UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ACATLÁN

MANEJO DE BASE DE DATOS CURSO CONALEP

ACTIVIDAD DE APOYO A LA DOCENCIA

QUE PARA OBTENER EL TITULO DE

LICENCIADO EN MATEMÁTICAS APLICADAS Y COMPUTACIÓN

PRESENTA

GABRIEL EMMANUEL FLORES BANDA

ASESOR: MTRA. MARÍA DEL CARMEN VILLAR PATIÑO

ABRIL, 2009

ÍNDICE

INTRODUCCIÓN

El presente trabajo tiene como finalidad el ser una herramienta teórico didáctica tanto para los alumnos así como para los profesores del Colegio Nacional de Educación Técnica (CONALEP) y otras instituciones de nivel medio superior, con el objetivo de apoyar la impartición de la materia “Manejo de Base de Datos”.

Como se desprende del párrafo que antecede, el correcto uso de este compendio, servirá para una mejor compresión y análisis de los temas relacionados con la materia de mérito.

Aunado a lo anterior, es preciso señalar que al desarrollar este tema de investigación se realizó un profundo análisis de los mapas conceptuales y temáticos impartidos en el CONALEP.

El capítulo 1 proporciona las definiciones de base de datos (BD) y sistema gestor de base de datos (SGBD), así como los elementos, funciones, objetivos y tareas que integran a los mismos. De la misma manera se manejan algunas otras definiciones de elementos de importancia para la buena realización de una base de datos.

Dentro del capítulo 2 se encuentra la definición, composición y uso de un modelo de base de datos. Definiendo los tres tipos, jerárquico, red y relacional; sus componentes, usos y la manera de diseñar cada modelo. Por lo que respecta al capítulo 3 se analizan los sistemas de gestión de base de datos, tomando en cuenta los distintos conceptos y usos. Se explican diferentes SGBD como son MySQL, ORACLE y SQL Server. Así mismo se menciona la diferencia existente entre un sistema gestor de bases de datos y una base de datos.

En relación con los capítulos 4 y 5, se estudian dos modelos de construcción de bases de datos, “El proceso de diseño para la realización de consultas de información” y “El diagrama E-R” respectivamente. Dando los aspectos generales y la manera de diseñar una base de datos.

Microsoft Access es un sistema de gestión de bases de datos Relacional creado y mantenido por Microsoft (DBMS) para uso personal de pequeñas organizaciones. Se expone en el capítulo 6 de una manera muy explícita el como utilizar este DBMS, el manual fue adquirido de la Universidad de Navarra a través de su página de Internet.

Por último el capítulo 7 es un complemento que enseñará al alumno que existen distintos SGBD, en este caso se trata de SQL. Se mencionan los componentes, funciones y consultas básicas para su utilización.

Nota: No se pretende proporcionar un manual de usuario completo para SQL. Ya que sólo se presentan las construcciones y conceptos fundamentales de éste. Las distintas implementaciones de SQL pueden diferenciarse en detalles, o pueden admitir sólo un subconjunto del lenguaje completo.

PARTE PRIMERA:

DISEÑO DE BASE DE DATOS

Objetivo: Al finalizar el alumno será capaz de construir juicios elaborados sobre diseño de Bases de Datos, para comprender el uso, aplicación y construcción de las mismas. Así como las diferentes herramientas que forman un SGBD.

CAPÍTULO 1: IDENTIFICAR BASES DE DATOS (BD) Y SISTEMAS GESTORES DE BASES DE DATOS (SGBD)

1.1. INTRODUCCIÓN

Las bases de datos son esenciales para el sistema de información de una organización, el cual soporta las funciones de la organización al mantener los datos para éstas y auxiliar a los usuarios a interpretar los datos para tomar decisiones. La base de datos juega un papel central en este proceso: es el depósito de los datos en el sistema de información.

Los usuarios que deben tomar decisiones obtienen los datos necesarios al acceder la base y registrar su decisión en ella. La localización de la base de datos y las facilidades para accederla tienen una gran relación con la efectividad del sistema de información. El fácil acceso de una variedad de datos desde cierto número de lugares permite que el sistema de información responda con rapidez a las necesidades de quienes toman decisiones en la organización, en tanto que un acceso deficiente puede, por supuesto, obstaculizar una rápida respuesta. Si no se dispone con facilidad de los datos, las decisiones quizá se retarden en forma innecesaria o se tomen con datos incompletos, lo que puede conducir a un mal desempeño posterior del sistema.

En el diseño de base de datos se usa el primero de los modelos conceptuales para lograr una descripción de alto nivel de la realidad luego se transforma el esquema conceptual en un esquema lógico. El motivo de realizar estas dos etapas es la dificultad de abstraer la estructura de una base de datos que presente cierta complejidad. Un esquema es un conjunto de representaciones lingüísticas o gráficas que describen la estructura de los datos de interés.

Los modelos conceptuales deben de ser buenas herramientas para representar la realidad por lo que debe poseer las siguientes cualidades:

Expresividad: Debe tener suficientes conceptos para expresar perfectamente la realidad.

Simplicidad: Debe ser simple para que los esquemas sean fáciles de entender.

Minimalidad: Cada concepto debe tener un significado distinto.

Formalidad: Todos los conceptos deben de tener una interpretación única, precisa y bien definida.

En general, un modelo no es capaz de expresar todas las propiedades de una realidad determinada por lo que hay que añadir secciones que complementen el esquema.

Las estructuras de bases de datos deben ser flexibles ante las necesidades de cambio de una organización. Cuando surge una nueva función en una organización, son necesarias nuevas decisiones. Como las bases de datos requieren almacenar nuevos datos y acomodar nuevas relaciones para apoyar las nuevas decisiones, se deben incluir opciones para permitir que estos cambios se hagan.

1.2. APOYO CONCEPTUAL

A continuación se dan algunas definiciones que nos ayudaran a introducirnos en el diseño de bases de datos.

Dato: La palabra dato (del latín data, plural de datum) significa simplemente “hechos”, entidades independientes sin evaluar. Los datos pueden ser numéricos o no numéricos.

Algunas otras definiciones de dato son:

- Un dato es la unidad o cantidad mínima de información no elaborada, sin sentido en sí misma, pero que convenientemente tratada se puede utilizar en la realización de cálculos o toma de decisiones.
- Unidad mínima que compone cualquier información.
- El termino que usamos para describir las señales con las cuales trabaja la computadora es dato. Aunque las palabras dato e información muchas veces son usadas indistintamente, sí existe una diferencia importante entre ellas. En un sentido estricto, los datos son las señales individuales en bruto y sin ningún significado que manipulan las computadoras para producir la información.

Información: Es un conjunto ordenado de datos los cuales pueden recuperarse de acuerdo con la necesidad del usuario. Para que un conjunto arbitrario de datos pueda ser procesado eficientemente y pueda dar lugar a información, primero se debe organizar lógicamente en archivos.

Algunas otras definiciones de información son:

- La información es un conjunto organizado de datos, que constituyen un mensaje sobre un determinado ente o fenómeno.
- Agrupación de datos con el objetivo de que lograr un significado específico más allá de cada uno de éstos. Un ejemplo 2, 0, 0 y 1 son datos; y 2001 es una información. La información ha sido siempre un recurso muy valioso, sobretodo hoy más aun por el desarrollo y la expansión de las Tecnologías de la Información y de las Comunicaciones.
- Es un conjunto ordenado de datos los cuales son manejados según la necesidad del usuario, para que un conjunto de datos pueda ser procesado eficientemente y pueda dar lugar a información, primero se debe guardar lógicamente en archivos.
- La información se define como: un sistema de datos o ideas sobre un tema determinado, datos que aumentan el conocimiento del investigador acerca del tema.

Archivo: Un archivo es una unidad de datos o información almacenada en algún medio que puede ser utilizada por aplicaciones de la computadora.

Cada archivo se diferencia del resto debido a que tiene un nombre propio y una extensión que lo identifica. Esta extensión sería como el apellido y es lo que permite diferenciar el formato del archivo y, asimismo, interpretar los caracteres que conforman el contenido del archivo. De esta manera, un archivo de texto, podrá tener la extensión .txt (el nombre completo sería: ARCHIVO.txt); un documento enriquecido, .doc, .pdf; uno de imágenes, .jpg, .gif; y lo mismo ocurre con cada formato.

Por otra parte, al estar conformados por caracteres, de la cantidad que contenga dependerá el tamaño del archivo, el que se podrá medir en Bytes, Kilobytes, Megabytes, etc.

Características generales de los archivos:

- Nombre y extensión: Cada archivo es individual y es identificable por un nombre y una extensión opcional que suele identificar su formato. El formato suele servir para identificar el contenido del archivo. Los nombres de archivos originalmente tenían un límite de ocho caracteres más tres caracteres de extensión, actualmente permiten muchos más caracteres dependiendo del sistema de archivos.

- Datos sobre el archivo: Además para cada archivo, según el sistema de archivos que se utilice, se guarda la fecha de creación, modificación y de último acceso. También poseen propiedades como oculto, de sistema, de solo lectura, etc.
- Tamaño: Los archivos tienen también un tamaño que se mide en bytes, kilobytes, megabytes, gigabytes y depende de la cantidad de caracteres que contienen.

- Ubicación: Todo archivo pertenece a un directorio o subdirectorio. La ruta de acceso a un archivo suele comenzar con la unidad lógica que lo contiene y los sucesivos subdirectorios hasta llegar al directorio contenedor, por ejemplo: "C:Archivos de programaMicrosoftarchivo.txt".

Existen diversos tipos de archivos, pero los dos grupos principales en los que se suelen dividir son ejecutables y no ejecutables. Los primeros, los ejecutables, se refieren a aquellos que corren por sí mismos. En cuanto a los no ejecutables, por el contrario, necesitan de algún programa para poder funcionar.

Registros Lógicos y Registros Físicos: Un registro lógico representa la percepción del programador de lo que es un registro de datos. Un registro físico puede consistir de varios registros lógicos, además de un control del sistema donde guarda información sobre el almacenamiento de los datos para facilitar la búsqueda. Esta parte se llama información sobre el sistema. Un registro físico es una unidad de transferencia de datos entre el dispositivo de almacenamiento de datos y la memoria principal.

Para el programador, los registros lógicos en un archivo están organizados uno detrás del otro, sin importar su verdadera posición en el dispositivo de almacenamiento.

En capítulos posteriores se abundará más sobre el diseño de las bases de datos que necesita un sistema de información, además de modelos de bases de datos, así como la construcción de una base de datos con un SGBD.

1.3. BASE DE DATOS Y SISTEMAS GESTORES DE BASE DE DATOS (SGBD O DBMS). DEFINICIÓN

Empezaremos dando las definiciones de lo que es una base de datos y un SGBD, para así poder abundar más sobre el diseño de una base de datos.

1.3.1. Base de datos

Base de Datos es un conjunto exhaustivo con redundancia controlada de datos estructurados organizados independientemente de su utilización y su implementación en máquina accesibles en tiempo real y compatibles con usuarios concurrentes con necesidad de información diferente y no predicable en tiempo.

Una definición mas sencilla es, la base de datos puede definirse como una colección de datos interrelacionados almacenados en conjunto sin redundancias perjudiciales o innecesarias su finalidad es la de servir a una aplicación o más, de la mejor manera posible; los datos se almacenan de modo que resulten independientes de los programas que los usan.

Otra definición, una base de datos es una colección de datos relacionados, y una descripción de estos datos, diseñados para cumplir con las necesidades de información de una organización.

La función básica de una base de datos es permitir el almacenamiento y la recuperación de la información necesaria, para que las personas de la organización puedan tomar decisiones.

A continuación se enlistan los objetivos que se quieren alcanzar al diseñar una base de datos.

Versatilidad para la representación de relaciones: El sistema de administración de datos debe ser capaz de representar relaciones de los datos almacenados, y crear los archivos lógicos que se requieren. Ejemplo:

La base de datos de un banco, representa a cada cliente con una unidad minima de almacenamiento, estos es:

$$\textbf{Cliente} \rightarrow C_{1...n}$$

Desempeño: La base de datos debe de ser eficiente en la función específica requerida por los usuarios. Las bases de datos deben asegurar un tiempo de respuesta adecuado para el dialogo entre el usuario y la terminal, además debe tener capacidad para manejar un adecuado caudal de transacciones. Ejemplo:

El presidente de una empresa dedicada a la manufactura de pantalones decide consultar las ventas de años anteriores, él nunca antes ha tenido contacto con la base de datos de la empresa, aún así, la consulta ha resultado rápida y eficiente debido al buen desempeño de la base.

Costo mínimo: La base de datos debe de tener un costo accesible para las organizaciones que la requieran, pero no por eso debe de ser ineficiente. Ejemplo:

Para llevar un mejor control de las ventas y el inventario de los productos, una miscelánea invierte $1,000 para el desarrollo de una base de datos.

Control sobre la redundancia de datos: Los sistemas de archivos almacenan varias copias de los mismos datos en archivos distintos. Esto hace que se desperdicie espacio de almacenamiento, además de provocar la falta de consistencia de datos. En los sistemas de bases de datos todos estos archivos están integrados, por lo que no se almacenan varias copias de los mismos datos. Sin embargo, en una base de datos no se puede eliminar la redundancia completamente, ya que en ocasiones es necesaria para modelar las relaciones entre los datos, o bien es necesaria para mejorar las prestaciones.

Capacidad de Búsqueda: La capacidad para explorar una base de datos rápidamente y con diferentes criterios de búsqueda depende mucho de la organización física de los datos. Con algunas organizaciones los tiempos de búsqueda son excesivamente prolongados para poder considerar las respuestas como de tiempo real. Uno de los objetivos de la organización es lograr capacidad para la búsqueda rápida y flexible.

Integridad: Toda la instalación debe garantizar la integridad de la información almacenada. Además de proteger los datos contra posibles problemas sistémicos deben incluirse también procedimientos que aseguren que los valores de los datos se ajusten a ciertas reglas prescritas de antemano. Estas pruebas podrán hacerse verificando las relaciones que existen entre varios valores de datos.

Reserva (privacidad) y seguridad: La reserva se refiere al derecho de los individuos y organismos para determinar por si mismos, cuándo, cómo y en qué medida se permitirá la transmisión a terceros de la información que les concierne.

La seguridad de los datos se refiere a la protección de éstos contra el acceso accidental o intencional por parte de personas no autorizadas y contra su indebida destrucción o alteración.

La interfase con el pasado: Cuando un organismo instala un nuevo software de base de datos, es importante que este pueda trabajar con los programas y procedimientos existentes y que los datos ya almacenados puedan ser convertidos a las nuevas formas.

La interfase con el futuro: Es importante en el diseño de una base de datos planearla de manera que sea modificable sin necesidad de cambiar los programas de aplicación en uso.

Afinación: Es necesario ajustes y cambiar fundamentalmente la organización del almacén de datos después de que el sistema ha entrado en servicio y se han aclarado suficientemente las pautas de uso. Este proceso de ajuste de la base de datos se llama afinación.

Migración de datos: Es conveniente mudar un conjunto de datos dentro del almacén de datos a posiciones accesibles de acuerdo con su actividad. Este proceso de ajuste del almacenamiento de los datos se llama migración de datos. En algunos sistemas esto se hace automáticamente en otros lo hacen los programadores del sistema o el administrador de datos.

El administrador de la base de datos se encarga de supervisar y mantener la vista lógica global de los datos.

Consistencia de datos: Eliminando o controlando las redundancias de datos se reduce en gran medida el riesgo de que haya inconsistencias. Si un dato está almacenado una sóla vez, cualquier actualización se debe realizar sólo una vez, y está disponible para todos los usuarios inmediatamente. Si un dato está duplicado y el sistema conoce esta redundancia, el propio sistema puede encargarse de garantizar que todas las copias se mantienen consistentes. Desgraciadamente, no todos los SGBD de hoy en día se encargan de mantener automáticamente la consistencia.

Compartición de datos: En los sistemas de archivos, los archivos pertenecen a las personas o a los departamentos que los utilizan. Pero en los sistemas de bases de datos, la base de datos pertenece a la empresa y puede ser compartida por todos los usuarios que estén autorizados. Además, las nuevas aplicaciones que se vayan creando pueden utilizar los datos de la base de datos existente.

Simplicidad: Los medios que se utilizan para representar la vista general de los datos deben ser concebidos de manera simple y nítida.

1.3.2. Sistemas gestores de base de datos (SGBD O DBMS)

Son programas de software para la administración de las bases de datos; que permiten: almacenar, manipular y recuperar datos en una computadora.

El SGBD también se encargará de la comunicación entre el usuario y la base de datos, proporcionándole al usuario, los medios para: obtener información, introducir nuevos datos y actualizar los ya existentes. El SGBD es responsable de las siguientes tareas:

Interacción con el Gestor de archivos: El SGBD es el responsable del verdadero almacenamiento, recuperación y actualización de los datos en la base de datos.

Integridad de los datos: La integridad de la base de datos se refiere a la validez y la consistencia de los datos almacenados. Normalmente, la integridad se expresa mediante restricciones o reglas que no se pueden violar. Estas restricciones se pueden aplicar tanto a los datos, como a sus relaciones, y es el SGBD quien se debe encargar de mantenerlas.

Seguridad: La seguridad de la base de datos es la protección de la base de datos frente a usuarios no autorizados. Sin unas buenas medidas de seguridad, la integración de datos en los sistemas de bases de datos hace que éstos sean más vulnerables que en los sistemas de archivos. Sin embargo, los SGBD permiten mantener la seguridad mediante el establecimiento de claves para identificar al personal autorizado a utilizar la base de datos. Las autorizaciones se pueden realizar a nivel de operaciones, de modo que un usuario puede estar autorizado a consultar ciertos datos pero no a actualizarlos.

Accesibilidad a los datos: Muchos SGBD proporcionan lenguajes de consultas o generadores de informes que permiten al usuario hacer cualquier tipo de consulta sobre los datos, sin que sea necesario que un programador escriba una aplicación que realice tal tarea.

Productividad: El SGBD proporciona muchas de las funciones estándar que el programador necesita escribir en un sistema de archivos. A nivel básico, el SGBD proporciona todas las rutinas de manejo de archivos típicas de los programas de

aplicación. El hecho de disponer de estas funciones permite al programador centrarse mejor en la función específica requerida por los usuarios, sin tener que preocuparse de los detalles de implementación de bajo nivel. Muchos SGBD también proporcionan un entorno de cuarta generación consistente en un conjunto de herramientas que simplifican, en gran medida, el desarrollo de las aplicaciones que acceden a la base de datos. Gracias a estas herramientas, el programador puede ofrecer una mayor productividad en un tiempo menor.

Mantenimiento gracias a la independencia de datos: En los sistemas de archivos, las descripciones de los datos se encuentran inmersas en los programas de aplicación que los manejan. Esto hace que los programas sean dependientes de los datos, de modo que un cambio en su estructura, o un cambio en el modo en que se almacena en disco, requiere cambios importantes en los programas cuyos datos se ven afectados. Sin embargo, los SGBD separan las descripciones de los datos de las aplicaciones. Esto es lo que se conoce como independencia de datos, gracias a la cual se simplifica el mantenimiento de las aplicaciones que acceden a la base de datos.

Aumento de la concurrencia: En algunos sistemas de archivos, si hay varios usuarios que pueden acceder simultáneamente a un mismo archivo, es posible que el acceso interfiera entre ellos de modo que se pierda información o, incluso, que se pierda la integridad. La mayoría de los SGBD gestionan el acceso concurrente a la base de datos y garantizan que no ocurran problemas de este tipo.

Copia de seguridad y de recuperación: Muchos sistemas de archivos dejan que sea el usuario quien proporcione las medidas necesarias para proteger los datos ante fallos en el sistema o en las aplicaciones. Los usuarios tienen que hacer copias de seguridad cada día, y si se produce algún fallo, utilizar estas copias para restaurarlos. En este caso, todo el trabajo realizado sobre los datos desde que se hizo la última copia de seguridad se pierde y se tiene que volver a realizar. Sin embargo, los SGBD actuales funcionan de modo que se minimiza la cantidad de trabajo perdido cuando se produce un fallo.

1.4. ELEMENTOS DE UNA BASE DE DATOS

Las siguientes son las definiciones que necesitamos manejar con claridad para comprender el resto de los conceptos en las bases de datos.

Campo: Es la unidad más pequeña a la cual uno puede referirse en un programa.

Registro: Es un conjunto de campos con relación entre sí.

Archivo: Es una colección de registros del mismo tipo (físico).

Tabla: Conjunto de registros homogéneos con la misma estructura (lógico)

Atributo: Es una columna en una tabla.

Entidad: Es un renglón en una tabla.

Dominio: Es el conjunto de valores de los cuales los atributos obtienen sus valores.

Tupla: Es una hilera o fila en una tabla.

Grado: Es el número de atributos en una tabla.

Cardinalidad: Es el número de tuplas en una tabla.

Figura 1.1. *Ejemplo de elementos de una base de datos*

Llave: Es un atributo con una característica de relevancia para identificar la tupla.

Llave primaria: Es una llave con valores únicos, es decir, no ocurren más de una vez en el atributo.

Vista: Es una relación virtual, que se construye a partir de tablas base o incluso otras vistas, formada por atributos de estas otras tablas de forma directa o como resultado de una consulta.

Relación: Una relación o vínculo entre dos o más entidades describe alguna interacción entre las mismas. Por ejemplo, una relación entre una entidad "Empleado" y una entidad "Sector" podría ser "trabaja_en", porque el empleado trabaja en un sector determinado.

1.5. ESTRUCTURA LÓGICA Y ESTRUCTURA FÍSICA

La estructura lógica corresponde con la idea que en principio tiene el programador sobre como están organizados los datos, y coincide aproximadamente con la forma en que son manipulados los datos por el programa de alto nivel.

En la concepción de la estructura lógica, el programador puede razonar más o menos en los siguientes términos: "Voy a crear un archivo de clientes donde los datos de cada cliente estarán agrupados en un registro. Posteriormente accederé los registros por número de cliente (que será único) o por nombre, para lo que estarán ordenados alfabéticamente (construiré un índice con el código de cliente y otro de nombres)...".

En uno u otro caso, la estructura (se llame "archivo" o "tabla") es una unidad lógica que se compone una multitud de elementos individuales (se llamen "registros" o "filas" -según la cultura del programador-). La estructura así concebida tiene un orden, ya que sus elementos estarán conceptualmente uno detrás de otro. Este orden será numérico, si el acceso se realiza por código de cliente, o alfabético de nombres si el acceso se realiza por nombre. A su vez, esta estructura lógica se divide aún más finamente: cada elemento se puede considerar dividido en multitud de campos. Aparte de los ya mencionados para código de cliente y nombre, pueden existir muchos mas:

dirección, teléfono, saldo, clasificación financiera, fecha última compra, vendedor asignado, etc. etc.

Por su parte, la estructura física corresponde a la forma en que están contenidos los datos en la máquina, de la que existen dos versiones: una corresponde a la que adoptan los datos en memoria; la otra a su almacenamiento externo (disco). Ambos esquemas son distintos.

Resulta evidente que la estructura física de datos en los almacenamientos externos no se corresponde exactamente con estructura lógica. En principio, el archivo o tabla de clientes antes mencionado, puede estar representado físicamente por varios archivos que pueden ser multi-volumen. Es decir: ocupar más de un volumen lógico en la máquina que los alberga. Si son aplicaciones de red, pueden estar incluso en máquinas remotas, distintas de la que ejecuta la aplicación. Además, aunque nos figuramos la estructura lógica es un todo continuo (suponemos que después de un cliente sigue otro), sabemos que la estructura física correspondiente, incluso si se trata solo de un archivo, está compuesta por trozos "clusters" que pueden estar dispersos en el disco.

La estructura lógica está ordenada (por números o por nombres en nuestro ejemplo). En cambio, la estructura física puede estar construida simplemente por el orden "natural" es decir, de creación de los propios registros. Generalmente, la "apariencia" de ordenación es el resultado de un proceso complejo que utiliza índices, tablas y punteros, para proporcionarnos un acceso ordenado a una estructura mucho más caótica.

Como queda dicho, los datos son manejados por el programador y el programa (que es la expresión concreta de las ideas de aquel) en términos de esta estructura lógica. En lo tocante a este aspecto, las herramientas que ofrezcan el lenguaje o entorno de programación, serán de mayor nivel cuanto mayor sea la distancia con que pueda ser manejada la estructura lógica de datos respecto de su verdadera estructura física.

Como resumen, podemos afirmar que el programador puede concentrarse en la estructura lógica, pero sin olvidar de vez en cuando mirar la estructura física con el rabillo del ojo. Como hemos señalado antes, existen distintos tipos de estructuras de datos (lógicas y físicas) que se diferencian grandemente en su grado de adecuación a diversas formas de almacenamiento y recuperación de la información, por lo que es conveniente que el programador tenga ciertas nociones al respecto.

1.5.1. Estructura lógica vs. Estructura física.

Es claro que la forma física como estén almacenados los datos es independiente del concepto que tengamos de ellos. Son el conjunto de programas que saben como traer, unir y mostrar los datos, así como aquellos encargados de almacenarlos, los que le dan coherencia al concepto Base de Datos.

Digamos que es como la diferencia entre harina, levadura, sal y agua por separado y una pieza de pan. Quién le da coherencia a esa pieza de pan es el proceso que se sigue para elaborarlo.

Es importante conceptualizar esto, porque del diseño de la estructura lógica depende toda la funcionalidad del sistema. Almacenar datos en una base de datos

aprovechando solamente la estructura física no ofrece, relativamente, ninguna ventaja. En cambio un buen diseño de acuerdo a la naturaleza de los datos y a la forma como serán explotados hace toda la diferencia.

Un gran problema es la inconsistencia de datos. Digamos que empleamos un archivo secuencial para almacenar la información de clientes. Supongamos que tenemos varios programas que utilizan esa información y que en un momento dado se pueden tener registros duplicados con atributos diferentes. Por ejemplo, una persona cambia de dirección y al no tener una estructura bien definida, no alteramos el registro, sino que lo damos de alta de nuevo con la nueva dirección. De esta manera, tendremos a la misma persona con dos datos diferentes y sin posibilidad de garantizar que todos los programas tendrán en cuenta que la dirección válida es la del segundo registro que aparece.

Puede ocurrir también que dos personas estén modificando simultáneamente atributos distintos del mismo registro. Sin un sistema de manejo de concurrencia, no podemos garantizar que ambos cambios permanezcan.

Bajo la misma suposición de uno o más archivos con la información y varios programas independientes que la explotan, es fácil ver que cualquier nueva explotación de la información implica un nuevo programa y que mantener un sistema como este conlleva toda la complejidad de mantener varios programas cuando se añade o elimina una columna a los registros.

Otro ejemplo, si tenemos un registro de personas donde almacenamos datos como: nombre, RFC, puesto, salario base, dirección, teléfono, fecha de nacimiento, gustos musicales, aficiones, nombre, fecha de nacimiento y ocupación del cónyuge, nombres y fechas de nacimiento de sus hijos, nombres de sus mascotas, autos que posee (con todas las características), etc.; y frecuentemente sólo utilizaremos nombre, RFC, puesto y salario base para generar una nómina, esto implica que en cada corrida, recuperaremos información que no necesitamos, con el agravante de que tenemos que traer registros inmensos para utilizar sólo cuatro campos. El problema no es el almacenaje en disco, sino el tiempo desperdiciado en recuperar registros de tal magnitud.

Un diseño un poco más inteligente tendría dos tablas, una con los datos más frecuentemente empleados de cada persona y la otra con el resto de la información que se emplea quizá sólo para fines estadísticos o para enviar tarjetas de felicitación. Por supuesto, ambas tablas estarán relacionadas por una llave primaria común.

Si tenemos un sistema donde por un lado hacemos un abono y por otro un cargo en una operación que está relacionada, es de esperar que no ocurra el cargo si no puede ocurrir el abono, o viceversa. Es decir, debemos de tener transacciones atómicas. En este caso, la pareja de transacciones debe de ocurrir por completo o no debe de ocurrir en lo absoluto.

Finalmente, un terrible problema es la exposición de los datos. En muchos casos nos interesa que ciertas personas tengan acceso sólo a parte de la información.

1.6. CONSISTENCIA

Dada la naturaleza de una consulta, de lectura o actualización, a veces no se puede simplemente reactivar la ejecución de una consulta, puesto que algunos datos pueden haber sido modificados. El no tomar en cuenta esos factores puede conducir a que la información en la base de datos contenga datos incorrectos.

El concepto fundamental aquí es la noción de "ejecución consistente" o "procesamiento confiable" asociada con el concepto de una consulta. El concepto de una transacción es usado dentro del dominio de la base de datos como una unidad básica de cómputo consistente y confiable.

Una transacción es una colección de acciones que hacen transformaciones consistentes de los estados de un sistema preservando la consistencia del sistema. Una base de datos está en un estado consistente si obedece todas las restricciones de integridad definidas sobre ella. Los cambios de estado ocurren debido a actualizaciones, inserciones, y supresiones de información. Por supuesto, se quiere asegurar que la base de datos nunca entra en un estado de inconsistencia. Sin embargo, durante la ejecución de una transacción, la base de datos puede estar temporalmente en un estado inconsistente. El punto importante aquí es asegurar que la base de datos regresa a un estado consistente al fin de la ejecución de una transacción.

Lo que se persigue con el manejo de transacciones es por un lado tener una transparencia adecuada de las acciones concurrentes a una base de datos y por otro lado tener una transparencia adecuada en el manejo de las fallas que se pueden presentar en una base de datos.

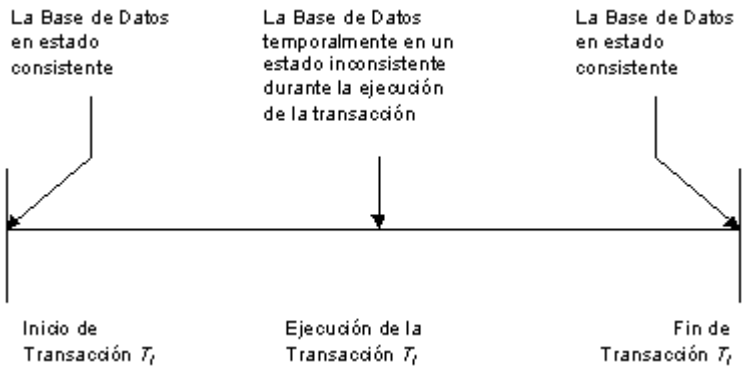


Figura 1.2. *Ejemplo de transacción*

En aquellos casos en los que no se tiene una redundancia nula, será necesario vigilar que aquella información que aparece repetida se actualice de forma coherente, es decir, que todos los datos repetidos se actualicen de forma simultánea.

1.7. COMPARACIÓN

El sistema manejador de bases de datos es la porción más importante del software de un sistema de base de datos. Un DBMS es una colección de numerosas rutinas de software interrelacionadas, cada una de las cuales es responsable de alguna tarea específica. Uno de los objetivos principales de un sistema de base de datos es proporcionar a los usuarios una visión abstracta de la información. Es decir, el sistema oculta ciertos detalles relativos a la forma como los datos se almacenan y mantienen. Sin embargo, para que el sistema sea útil, la información debe recuperarse en forma eficiente.

La búsqueda de la eficiencia conduce al diseño de estructuras de datos complejas para representar la información en la base de datos. Pero como los sistemas de base de datos muchas veces son utilizados por personal que no cuenta con conocimientos de computación, esta complejidad debe estar escondida para los usuarios. Para ocultarla, se definen varios niveles de abstracción en los que puede observarse la base de datos.

- **Nivel físico.** Este es el nivel más bajo de abstracción, en el que se describe cómo se almacenan realmente los datos. En este nivel se describen en detalle las estructuras de datos complejas del nivel más bajo.

- **Nivel conceptual.** Este es el nivel intermedio de abstracción, en el que se describe cuáles son los datos reales que están almacenados en la base de datos y qué relaciones existen entre los datos. Este nivel contiene toda la base de datos en términos de unas cuantas estructuras relativamente sencillas. Aunque es posible que la implantación de las estructuras simples del nivel conceptual requiera estructuras complejas en el nivel físico, no es forzoso que el usuario del nivel conceptual se dé cuenta de ello. El nivel conceptual de abstracción lo utilizan los administradores de base de datos; quienes deciden qué información se guarda en la base de datos.

- **Nivel de visión.** Este es el nivel de abstracción más alto, en el cual se describe solamente una parte de la base de datos. Aunque en el nivel conceptual se utilizan estructuras más simples, todavía queda una forma de complejidad que resulta del gran tamaño de la base de datos. Muchos usuarios de la base de datos no tendrán que ocuparse de toda esta información. Más bien, necesitarán solamente una parte de la base de datos. Para simplificar la interacción entre estos usuarios y el sistema, se define el nivel de abstracción de visión. El sistema puede proporcionar muchas vistas diferentes de la misma base de datos.

La interrelación entre estos tres niveles de abstracción se muestra en la siguiente figura:

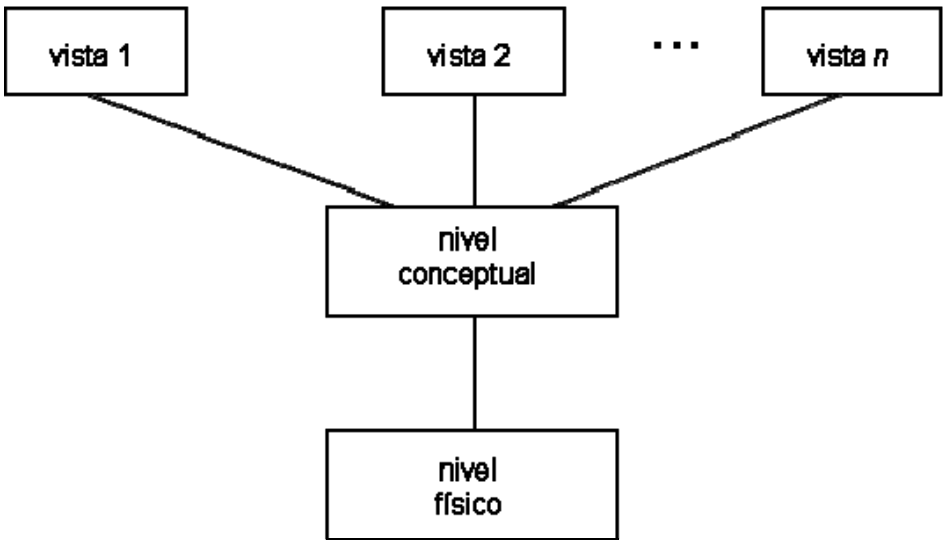


Figura 1.3. *Interrelación entre nivel físico, conceptual y de visión*

1.8. NORMAS

Con un control central de la base de datos el DBA puede garantizar que se cumplan todas las formas aplicables a la representación de los datos, las normas aplicables pueden comprender la totalidad o parte de lo siguiente: Normas de la compañía, de instalación, departamentales, industriales, etc. es muy deseable unificar los formatos de los datos almacenados como ayuda para el intercambio o migración de datos entre sistemas

1.9. RESTRICCIONES DE SEGURIDAD

La seguridad de los datos está fuertemente relacionada con la integridad de los mismos. La seguridad se refiere a la protección de la base contra accesos o modificaciones no autorizados. Como las bases de datos se comparten ampliamente, la información confidencial en la base es vulnerable a las intromisiones. Un acceso ilegal puede tener como resultado la destrucción accidental o maliciosa de la base. Sin control de seguridad, los usuarios no tendrán la privacía requerida en sus datos confidenciales y el sistema no podrá mantener la integridad de los mismos.

El administrador de base de datos (DBA) controla los derechos de los usuarios a ciertas porciones de la base declarando los elementos de datos apropiados en un subesquema. Los elementos de datos no incluidos en un subesquema serán inaccesibles a los programas que usen tal subesquema.

Debido a que la base se comparte extensamente, el DBMS protege los datos individuales de cada usuario contra la intrusión o destrucción no autorizada.

Para analizar los mecanismos de seguridad en el control de concurrencia, primero se introduce la noción de transacción. Una petición en línea a la base de datos se

considera como una transacción; generalmente involucra llamadas a rutinas del DBMS para operaciones de Entrada/Salida y alguna cantidad limitada de operaciones. El proceso de transacción empieza tan pronto como se propone la petición del usuario. Termina cuando se contesta la petición y en ese momento el sistema libera todos los recursos utilizados para la transacción.

El término granularidad se refiere al tamaño de las unidades aseguradas de datos compartidos. En la programación con lenguajes de alto nivel, el sistema operativo proporciona un seguro para cada archivo. Así, la granularidad del seguro en este caso es todo un archivo. Cuando un archivo se abre, se le asegura para excluir del acceso a otros programas, hasta que el archivo quede cerrado. En un sistema de manejo de base de datos, la granularidad del seguro puede ser un registro, un campo, o incluso ciertos valores de un campo.

No existen estándares en la industria para la implantación de los mecanismos de seguridad en un DBMS. Idealmente, un DBMS debería de prevenir automáticamente la interferencia de la simultaneidad sin necesidad de involucrar al programador.

1.10. BASE DE DATOS CENTRALIZADA Y DISTRIBUIDA

Una base de datos centralizada es una base de datos que está físicamente situada en un único lugar, controlado por una sola computadora, es decir, los datos y/o la información reside en una sola computadora. La mayoría de las funciones se llevan a cabo más fácilmente si la base de datos está centralizada.

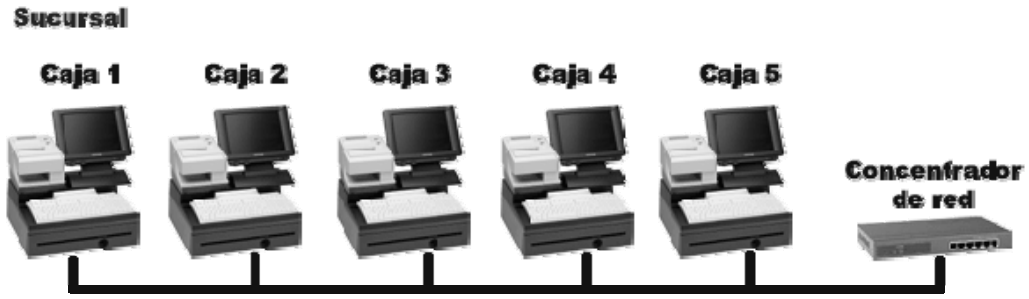


Figura 1.4. *Ejemplo de base de datos centralizada.*

Un sistema de base de datos distribuida (compuesto de varios sistemas de bases de datos operando en los sitios locales y conectados por líneas de comunicación), hace posible que los datos residan donde se necesitan con más frecuencia, mientras que al mismo tiempo puedan acceder a los mismos otros usuarios no locales. En otras palabras, en las bases de datos distribuidas: los datos y/o la información se almacenan en varias computadoras.

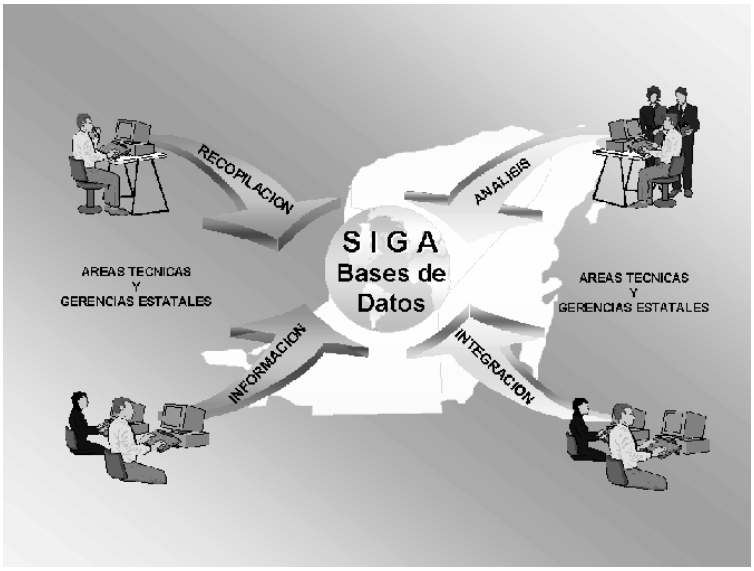


Figura 1.5. *Ejemplo de base de datos distribuida.*

1.11. ALMACENAMIENTO DE DATOS

Los métodos para desarrollar un sistema de información están evolucionando de varias formas con respecto al uso de nuevas herramientas y técnicas, como lo hemos venido analizando, y también, involucrando a los usuarios finales en el proceso de desarrollo. Deseamos esos avances.

El centro de muchas de las actividades de los desarrolladores está en la forma en que se almacenan los datos en el sistema para su recuperación cuando sea necesario. Aquí hay algo que permanece constante. Las computadoras, ya sean grandes o pequeñas, y los programas que utilizan lenguajes de tercera, cuarta o quinta generación, siguen almacenando datos en únicamente dos formas: secuencialmente, con un registro detrás del otro, o aleatoriamente, en lugares específicos de almacenamiento. Estos métodos son una característica arquitectónica fundamental de los dispositivos de almacenamiento de las computadoras.

En cierto momento, los datos deben traducirse a una estructura de almacenamiento, ya sea secuencial o aleatoria. Ahí es donde se llega al punto importante (o cuando el disco alcanza a los datos).

Los sistemas de información se organizan con base en archivos que acumulan y almacenan datos para su procesamiento. Los archivos contienen registros relacionados con datos, los cuales describen entidades de importancia para la organización. Los propios registros se pueden diseñar según un formato de longitud fija o variable, dependiendo de la naturaleza de la aplicación y la cantidad de espacio de almacenamiento disponible. Los registros de longitud fija pueden utilizar mayor espacio, pero a menudo simplifican los requerimientos de programación y procesamiento.

Los tipos principales de archivos usados en los sistemas de información son los archivos maestros, de transacciones, de tablas y de reportes. Los archivos maestros son archivos permanentes que existen durante toda la vida de un sistema, aunque deben mantenerse actualizados para que sean útiles. Cuando ocurren eventos que

afectan o involucran a la organización, los datos que los describen son capturados en los archivos de transacciones, que a su vez son procesados para actualizar el archivo maestro. Los archivos de tablas se usan para almacenar datos de referencia, los cuales se utilizan cuando se procesan las transacciones o se producen las salidas. Los archivos de reportes, por el contrario, acumulan las salidas que se pueden guardar temporalmente en una cola de espera en un disco magnético hasta que se puedan imprimir o producir en el dispositivo de salida apropiado.

Todos los archivos se almacenan utilizando un método de organización del archivo que se basa en la posición o dirección. El método más simple, la organización secuencial, es la única forma en que los registros se pueden guardar en una cinta magnética. Cada registro se almacena en la siguiente posición disponible en la cinta. Con los dispositivos de acceso directo, tales como el disco magnético, los registros se pueden almacenar secuencialmente o bajo acceso directo u organizaciones indexadas. Los métodos de acceso directo e indexado usan las direcciones de almacenamiento para identificar la localización específica de los registros o bloques de datos. Una dirección está formada por el número de cilindro, superficie y sector (áreas específicas del disco que el sistema pueda localizar rápidamente).

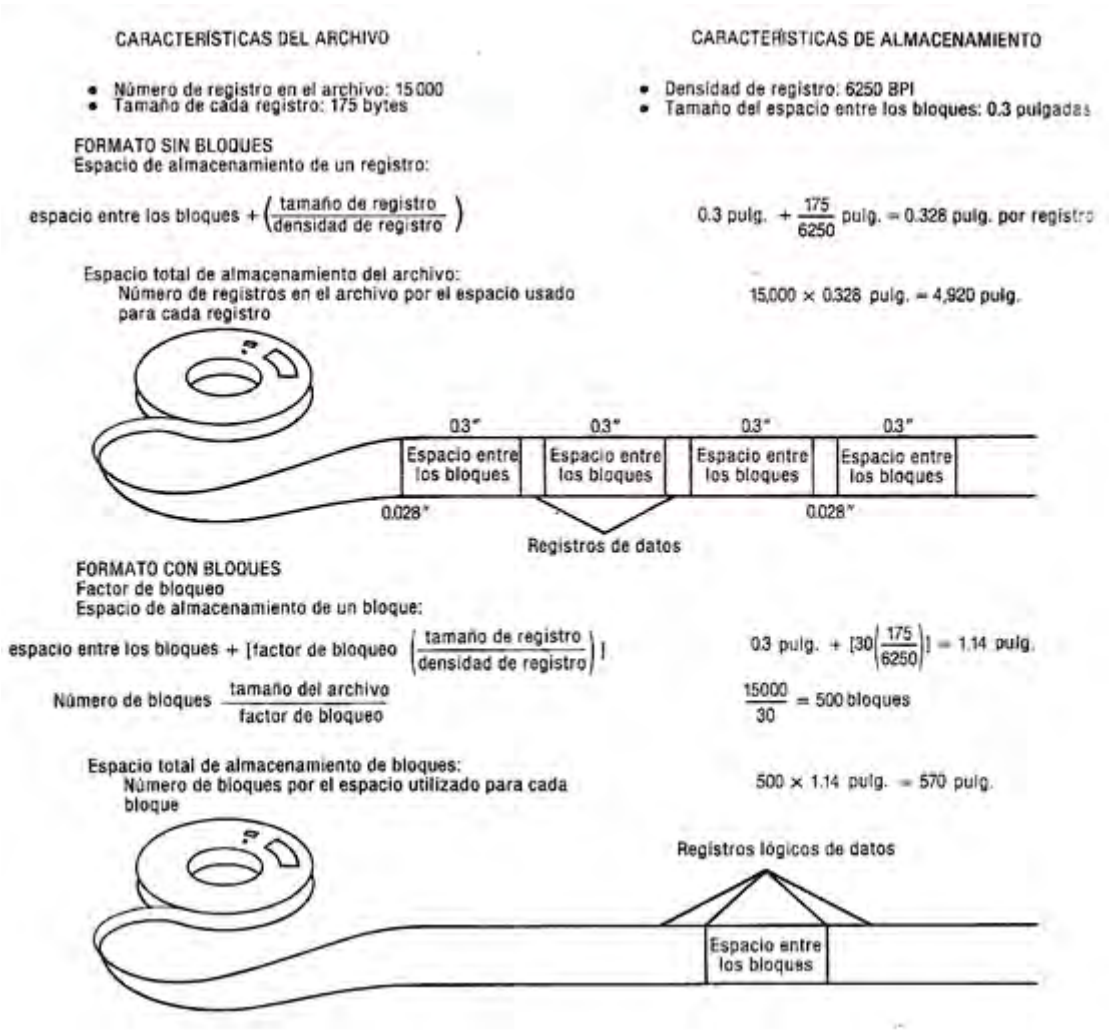


Figura 1.6. *Almacenamiento de datos en una cinta magnética.*

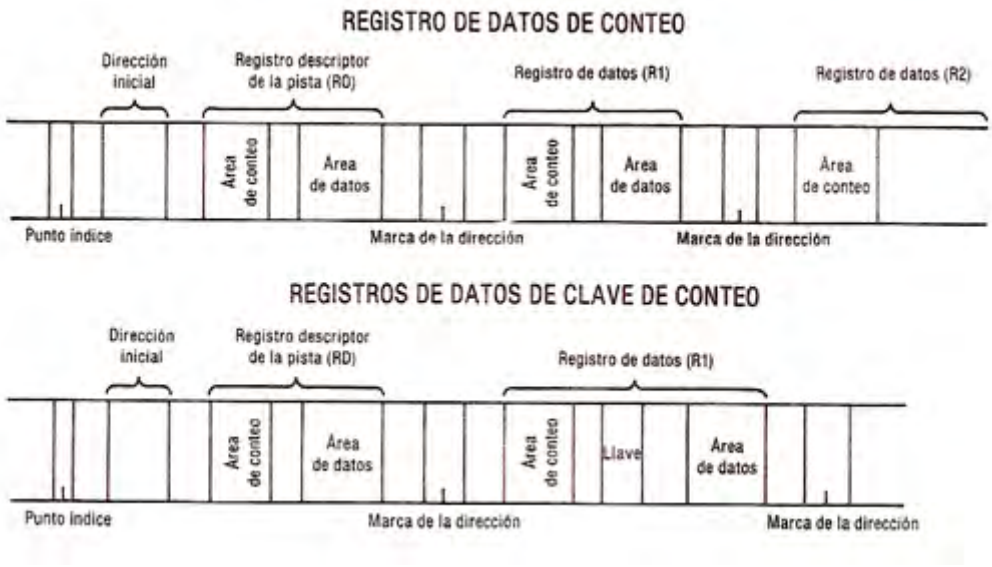


Figura 1.7. *Almacenamiento de datos en un disco magnético.*

Existen dos organizaciones de acceso directo. El direccionamiento directo significa que la llave registro corresponde exactamente a una dirección de almacenamiento. Sin embargo, ya que no es usual que las llaves de registro cumplan este requerimiento, a menudo se utiliza el hashing para proporcionar un medio de acceso directo. Con hashing, se obtiene una dirección de almacenamiento aplicando un algoritmo de transformación de la llave que divide, extrae, o dobla la llave del registro. En algunos casos, se desarrollan sinónimos; dos o más llaves producen la misma dirección. Se reserva un área aparte de almacenamiento, el área de overflow, para almacenar los sinónimos que aparecen en un archivo. El número de sinónimos puede controlarse desarrollando cuidadosamente el algoritmo de hashing.

Las organizaciones de archivo indexado secuencial y no secuencial utilizan un archivo separado, el índice, que sigue la pista de las llaves de registro y las direcciones de almacenamiento de los registros en el archivo maestro. Para hallar un registro, el programa examina primero el índice y recupera el registro del archivo maestro en el lugar especificado por el índice. El índice hace posible procesar el archivo en forma aleatoria o secuencial. Las organizaciones indexadas secuenciales son las más comunes.

Cuando se actualizan los archivos secuenciales, ya sea en cinta o disco magnéticos, se produce una nueva copia del archivo maestro. Sin embargo, cuando se actualizan los archivos de acceso directo o indexado, los cambios se hacen en el archivo y no se produce una nueva copia. Esto es posible debido a las llaves de registro físico que se utilizan dentro de los archivos de acceso directo e indexado; no existen llaves físicas en los archivos secuenciales.

El uso de cinta y discos magnéticos para el almacenamiento secundario ofrece un almacenamiento más conciso que el que es posible con los registros en papel. Sin embargo, ambos medios de almacenamiento están sujetos al uso y posibles daños.

Para protegerse contra la pérdida de datos, los analistas de sistemas diseñan procedimientos de respaldo. Se hacen copias duplicadas de los datos para garantizar que siempre existirá una copia de los registros, aun cuando el original se dañe o destruya; siempre existe la probabilidad de que esto ocurra debido al error, falla del equipo o del software o a un desastre natural. Al utilizar archivos secuenciales, las

generaciones de copias proporcionan el respaldo adecuado. Sin embargo, cuando los archivos se almacenan mediante una organización de acceso directo o indexado, se necesita un método de vaciado de los archivos o hacer copias antes y después. Los analistas deben suponer que las copias de respaldo se necesitarán tarde o temprano para mantener al sistema funcionando.

1.12. MANIPULACIÓN DE LOS DATOS

La determinación de los requerimientos de sistemas es un proceso en continua evolución. Los analistas aumentan de manera gradual su nivel de comprensión de un sistema o proceso existente, al añadir detalles conforme estudian grandes porciones del sistema en forma independiente.

El análisis estructurado es un enfoque general que permite a los analistas desarrollar en forma gradual la comprensión de los componentes de un sistema. El objetivo detrás de esto es organizar las tareas asociadas con la determinación de requerimientos en forma tal que sea posible documentar el sistema existente con exactitud. El término estructurar se aplica a la estructuración del proceso, a la intención de incluir todos los detalles relevantes relacionados con el sistema, a la capacidad de detectar la omisión de detalles, al desarrollo de la mejor solución posible (sin importar qué analista trabaje sobre el proyecto) y a la producción de documentos que faciliten la comunicación entre analistas y usuarios.

El análisis de flujo de datos está integrado por cuatro herramientas: diagramas de flujo de datos, diccionarios de datos, diagramas de estructuras de datos y gráficas de estructura. Las primeras dos se desarrollan durante la determinación de requerimientos mientras que las dos últimas tienen mayor utilidad durante el diseño de sistemas.

Un diagrama de flujo de datos es una descripción gráfica de un sistema o de una parte de él. Está formado por flujos de datos, procesos, fuentes, destinos y almacenes, todos ellos descritos por medio del uso de símbolos fáciles de comprender. Desde el punto de vista de los datos, todo el sistema puede describirse con sólo cuatro símbolos. Al mismo tiempo, los diagramas de flujo de datos son bastante poderosos como para señalar las actividades que ocurren en paralelo. Cuando los símbolos estándar limitan la comunicación se puede utilizar una gráfica de presentación que emplea símbolos de personas, archivos, terminales y documentos para discutir el sistema con los usuarios.

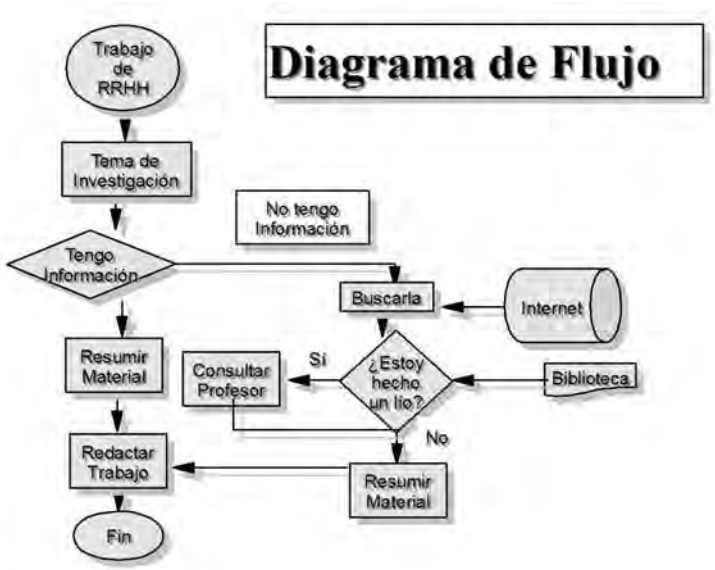


Figura 1.8. *Ejemplo de diagrama de flujo.*

Los diagramas físicos de flujo de datos dependen de la implantación. Muestran los dispositivos, departamentos y personas, entre otros, que tienen relación con el sistema en uso. Los diagramas lógicos de flujo de datos, en contraste, describen al sistema en una forma que es independiente de la implantación; es decir, indican lo que ocurre más que la forma como se realiza la actividad. Los dos tipos de diagramas de flujo de datos apoyan el enfoque descendente (top-down) para el análisis de sistemas, ya que los analistas comienzan desarrollando un conocimiento general del sistema que gradualmente extienden en componentes más detallados. Conforme se añaden detalles también es posible incluir información relacionada con los controles, aunque los diagramas de nivel superior se dibujan, en general, sin señalar ningún aspecto sobre controles específicos con la finalidad de asegurar que la atención se centre sobre los datos y procesos.

El diccionario de datos contiene descripciones de datos y estructuras así como de los procesos del sistema. Su finalidad es ayudar a los analistas a comprender el sistema ya que éstos recuperan las descripciones y detalles que contiene; por otra parte pone a disponibilidad de los que intervienen en el diseño de sistemas información sobre longitud de los datos, diferentes nombres para el mismo dato (alias), y los datos utilizados en procesos específicos. El diccionario de datos también guarda información sobre aspectos para validar que sirven como guía a los analistas al especificar los controles para aceptar datos por parte del sistema.

Los sistemas de diccionario de datos son importantes por las siguientes cinco razones: 1) manejar todos los detalles en sistemas grandes; 2) comunicar el mismo significado para todos los elementos del sistema; 3) documentar las características del sistema; 4) facilitar el análisis de los detalles para evaluar las características y determinar dónde deben realizarse los cambios, y 5) localizar errores y omisiones en el sistema. Durante el análisis se pone particular atención en comprender la naturaleza de las transacciones y consultas hechas por el sistema junto con los requerimientos para salidas y generación de reportes. Estos aspectos son significativos para determinar los requerimientos de archivos y bases de datos y las necesidades sobre la capacidad del sistema.

Los elementos dato forman el nivel más importante de datos contenidos en el diccionario; son los bloques básicos sobre los que se construyen los demás datos del sistema. Un conjunto de datos, denominado estructura de datos, describe la relación existente entre elementos individuales. Los datos se colocan de acuerdo con una de cuatro relaciones: secuencia, selección (uno u otro), iteración (repetición) y opcional. Sin importar cuál sea la relación específica, se debe describir cada detalle de los datos incluyendo el nombre, así como las especificaciones sobre su tipo y longitud.

El diccionario también contiene definiciones de flujos de datos, almacenes de datos y procesos. Estos últimos incluyen un resumen de la lógica de procesamiento.

Los diccionarios de datos pueden desarrollarse ya sea en forma manual o automatizada. Los sistemas automatizados ofrecen la ventaja de generar de manera automática elementos dato, estructuras de datos, y listados de los procesos. Asimismo, efectúan verificaciones con referencias cruzadas y detección de errores; todas éstas son ventajas importantes cuando se trabaja ante sistemas grandes que deben ser correctos. Los sistemas automatizados de diccionarios de datos se vuelven cada vez más la norma en el desarrollo de sistemas de información basados en computadora.

Algunos de los beneficios que reporta el diccionario de datos son los siguientes:

- La información sobre los datos se puede almacenar de un modo centralizado. Esto ayuda a mantener el control sobre los datos, como un recurso que son.
- El significado de los datos se puede definir, lo que ayudará a los usuarios a entender el propósito de los mismos.
- La comunicación se simplifica ya que se almacena el significado exacto. El diccionario de datos también puede identificar al usuario o usuarios que poseen los datos o que los acceden.
- Las redundancias y las inconsistencias se pueden identificar más fácilmente ya que los datos están centralizados.
- Se puede tener un historial de los cambios realizados sobre la base de datos.
- El impacto que puede producir un cambio se puede determinar antes de que sea implementado, ya que el diccionario de datos mantiene información sobre cada tipo de dato, todas sus relaciones y todos sus usuarios.
- Se puede hacer respetar la seguridad.
- Se puede garantizar la integridad.
- Se puede proporcionar información para auditorias.

Ejemplo:

El dueño de la compañía EYE dedicada a la fabricación de lentes decide automatizarla. Esta automatización es para apoyar a la producción de los productos así como eliminar el desperdicio en todas las actividades de la compañía.

El sistema propuesto por un desarrollador de base de datos es cumplir con las metas inmediatas de automatizar la tarea de preparar el plan diario de producción y apoyar la gerencia en la toma de decisiones, para lograr esto crea el siguiente diagrama de flujo mostrando el proceso que sigue la empresa para la fabricación de sus productos.

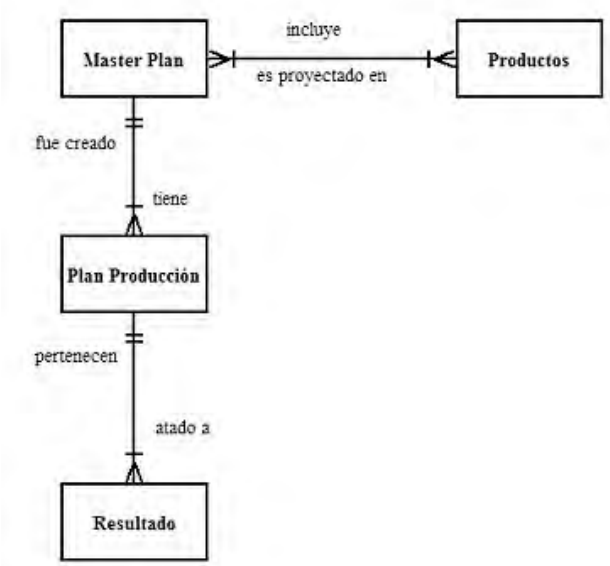


Figura 1.9. *Esquema relacional utilizado para la automatización de la empresa EYE.*

A continuación se muestra el diseño de la base de datos de la compañía EYE.

Nombre	Descripción	Estructura
Código del Producto	Identifica de forma única el producto por presentación.	Código del Producto = Base + Adicción + Ojo
Ensamblajes	Reporte de los ensamblajes que pudieron hacerse siguiendo	Ensamblajes = Nombre del Producto + Código del Producto + Fecha producción + Cantidad Ensamblada
Error en Figuras	Se recibió información cuantitativa del producto que se entiende no está correcta.	Error en Productos = Nombre del Reporte + Código del Producto + Explicación del Error + Fecha
Error en Pedidos	Explicación de errores encontrados en pedidos de productos.	Error en Pedidos = #Orden + Explicación del Error + Fecha
Error en Productos	Se recibió información descriptiva del producto que se entiende no está correcta.	Error en Productos = Nombre del Reporte + Código del Producto + Explicación del Error + Fecha
Productos Aceptados o Lentes Aceptados	Reporte de los lentes que fueron aceptados luego de un subproceso.	Productos Aceptados = Nombre del Producto + Código del Producto + Fecha producción + Cantidad aceptada
Lentes Rechazados	Reporte de los lentes que fueron rechazados de lo estimado en el Reporte de Producción Diaria	Lentes Rechazados = Nombre del Producto + Código del Producto + Fecha producción + Cantidad rechazada
Lentes Terminados	Reporte de los lentes que fueron terminados de lo estimado en el Reporte de Producción Diaria	Lentes Terminados = Nombre del Producto +Código del Producto + Fecha producción + Cantidad terminada

Ojo	Identifica ojo izquierdo o derecho.	Ojo = [L | R]
Pedidos de Productos	Ordenes de productos cuyos pedidos no se pueden completar utilizando el inventario.	Pedidos de Productos = #Orden + Código del Lente + Cantidad + Fecha
Pedidos Validados	Pedidos de Productos debidamente verificados y validados.	Pedidos Validados = #Orden + Código del Lente + Cantidad + Fecha de Aprobación
Plan Maestro	Contiene la proyección anual de producción por producto mensual resumida en forma anual.	Plan Maestro = Nombre-Producto + Año + Cantidad Proyectada Mensual + Cantidad Proyectada Anual + Por ciento de Resultado proyectado + Estatus
Plan Maestro con Productos Validados	Contiene la proyección anual de producción por producto	Plan Maestro con Productos Validados = Nombre-Producto + Año + Cantidad Proyectada Mensual + Cantidad Proyectada Anual + Por ciento de Resultado proyectado + Estatus
Plan Maestro de Producción	Contiene la proyección anual de producción por producto y presentación de producto mensual. Es exactamente igual al Plan Maestro, se diferencia en que su estatus es de Aprobado, significa que es el único que puede ser utilizado para determinar la producción diaria.	Plan Maestro de Producción = Nombre-Producto + Código del Producto + Cantidad Proyectada Mensual + Cantidad Proyectada Anual + Año + Por ciento de Resultado proyectado + Estatus
Plan Maestro Validado	El Plan Maestro con toda la información que contiene validada y confirmada.	Plan Maestro Validado = Nombre-Producto + Año + Cantidad Proyectada Mensual + Cantidad Proyectada Anual + Por ciento de Resultado proyectado + Estatus
Producción	Describe toda la información que se desea guardar de producción incluyendo estadísticas históricas, cálculos, ajuste, datos, etc.	Producción = El formato es libre.
Producción Diaria	Estimado de la producción del día que se calculó utilizando el Plan Maestro y los pedidos de productos.	Producción Diaria = Código del Producto + Fecha de producción + Cantidad
Reporte Producción Diaria	Reporte en forma de matriz con la producción diaria estimada.	Reporte Producción Diaria = Nombre del Producto + Código del Producto + Fecha producción + Cantidad a Producirse
Reportes Validados	Reportes cuyo contenido fu, verificado.	Reportes Validados = Código del Lente + Cantidad + Fecha
Resultados	Los resultados obtenidos por el departamento de producción al llevar a cabo el plan o estimado de producción diario.	Resultados = [Casted | Aceptados | Rechazados] + Fecha de producción + Cantidad
Resultados de Producción	Grupo de reportes que resumen los resultados	Resultados de Producción = Nombre del Producto + Mes + Año + Cantidad + Por ciento de Resultado actual
Totales Mensuales	Grupo de reportes que resumen los resultados mensualmente.	Totales Mensuales = Nombre del Producto + Mes + Año + Cantidad + Por ciento de Resultado actual

Como se puede ver el diccionario de datos es una herramienta muy útil en el desarrollo de una base de datos, ya que contiene las características lógicas de la misma.

1.13. CONTROL DE DATOS

Un esquema de base de datos se especifica por medio de una serie de definiciones que se expresan en un lenguaje especial llamado lenguaje de definición de datos (en inglés: DDL, data definition language). El resultado de la compilación de las proposiciones en DDL es un conjunto de tablas que se almacena en un archivo.

La estructura de almacenamiento y los métodos de acceso empleados por el sistema de base de datos se especifican por medio de un conjunto de definiciones de un tipo especial de DDL llamado lenguaje de almacenamiento y definición de los datos. El resultado de la compilación de estas definiciones es una serie de instrucciones que especifican los detalles de implantación de los esquemas de base de datos que normalmente no pueden ver los usuarios.

Los niveles de abstracción no solamente se aplican a la definición o estructuración de los datos, sino también al manejo los datos; esta manipulación consiste en:

- La recuperación de información almacenada en la base de datos.
- La inserción de información nueva en la base de datos.
- La eliminación de información de la base de datos.

En el nivel físico, deben definirse algoritmos que permitan tener acceso a los datos en forma eficiente. En los niveles de abstracción más altos lo importante es la facilidad de uso. El objetivo es lograr una interacción eficiente entre las personas y el sistema.

Un lenguaje de manejo de datos (en inglés: DML, data manipulation language) permite a los usuarios manejar o tener acceso a los datos que estén organizados por medio del modelo apropiado. Existen básicamente dos tipos de DML:

- **De procedimientos,** necesitan que el usuario especifique cuales datos quiere y cómo deben obtenerse.
- **Sin procedimientos,** requieren que el usuario especifique cuales datos quiere *sin* especificar cómo obtenerlos. .

Los DML sin procedimientos son, por lo general, más fáciles de aprender y utilizar que los de procedimientos. Sin embargo, ya que el usuario no tiene que especificar la forma de obtención de los datos, estos lenguajes podrían generar un código menos eficiente que el producido por los lenguajes de procedimientos. Tal problema puede resolverse empleando diversas técnicas de optimización.

Una consulta es una proposición que solicita la recuperación de información.

La parte de un DML que implica la recuperación de información se conoce como lenguaje de consultas. Aunque técnicamente es incorrecto, suelen utilizarse los términos lenguaje de consultas y lenguaje de manejo de datos como sinónimos.

1.14. RESUMEN

Un sistema de manejo de base de datos (DBMS) se compone de una serie de datos relacionados entre sí y de un conjunto de programas para tener acceso a esos datos. Los datos contienen información referente a determinada empresa. El objetivo principal de un DBMS es crear un ambiente en el que pueda almacenarse y recuperarse información en la base de datos en forma conveniente y eficiente.

Los sistemas de base de datos se diseñan para manejar grandes cantidades de información. El manejo de los datos implica tanto la definición de estructuras para el almacenamiento como la creación de mecanismos para manejar la información. Además, el sistema de base de datos debe cuidar la seguridad de la información almacenada en la base de datos, previniendo caídas del sistema o intentos de acceso no autorizados. Si se va a compartir la información entre varios usuarios, el sistema debe evitar posibles resultados anómalos.

Uno de los objetivos principales de una base de datos es proporcionar a los usuarios una visión abstracta de los datos. Es decir, el sistema oculta ciertos detalles relativos a la forma en que se almacenan y mantienen los datos. Esto se logra definiendo tres niveles de abstracción en los que puede considerarse la base de datos: físico, conceptual y de visión.

Para describir la naturaleza de una base de datos, se define el concepto de modelo de datos, que es un conjunto de herramientas conceptuales para describir los datos, las relaciones entre ellos, su semántica y sus limitantes. Se han propuesto varios modelos diferentes, los cuales se dividen en tres grupos: lógicos basados en objetos, lógicos basados en registros y los modelos físicos de datos.

Las bases de datos cambian con el tiempo al insertar información en ellas y eliminada. El conjunto de información almacenado en la base de datos en determinado momento se denomina instancia, de la base de datos. El diseño general de dicha base se conoce como esquema de la base de datos. La capacidad para modificar una definición de esquema en un nivel sin afectar la definición del esquema en el nivel inmediato superior se denomina independencia de los datos. Existen dos niveles de ésta: independencia física e independencia lógica de los datos.

Un esquema de base de datos se especifica por medio de una serie de definiciones que se expresa en un lenguaje de definición de datos (DDL). El resultado de la compilación de las proposiciones en DDL es un conjunto de tablas que se almacenan en un archivo especial llamado diccionario de datos que contiene metadatos, es decir, "datos acerca de los datos".

Un lenguaje de manejo de datos (DML) permite a los usuarios tener acceso a los datos o manejarlos. Existen básicamente dos tipos de DML: de procedimientos (requieren que el usuario especifique cuáles datos necesita y cómo se van a obtener) y sin procedimientos (requieren que el usuario especifique cuáles son los datos que necesita sin especificar la forma de obtención).

Un manejador de base de datos es un módulo de programa que constituye la interfaz entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicaciones y las consultas que se hacen al sistema. El manejador de base de datos se encarga de interactuar con el manejador de archivos, de conservar la integridad, de garantizar la seguridad, del respaldo y recuperación y del control de concurrencia.

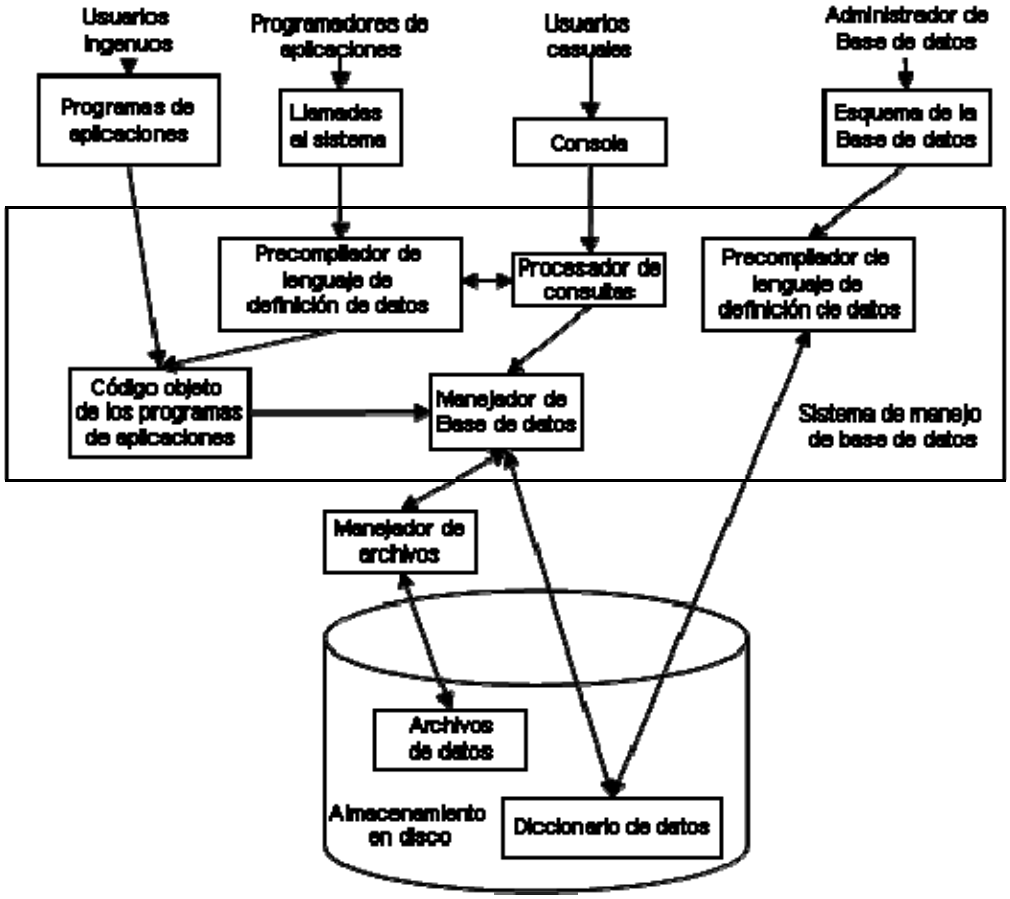


Figura 1.10. *Estructura de una base de datos*

CAPÍTULO 2: MODELOS DE BASES DE DATOS

2.1. INTRODUCCIÓN

Un modelo de datos es básicamente una "descripción" de algo conocido como contenedor de datos (algo en donde se guarda la información), así como de los métodos para almacenar y recuperar información de esos contenedores. Los modelos de datos no son cosas físicas: son abstracciones que permiten la implementación de un sistema eficiente de base de datos; por lo general se refieren a algoritmos, y conceptos matemáticos. Es formal pues los objetos del sistema se manipulan siguiendo reglas perfectamente definidas y utilizando exclusivamente los operadores definidos en el sistema, independientemente de lo que estos objetos y operadores puedan significar.

Un modelo de datos es una combinación de tres componentes:

- Una colección de estructuras de datos (los bloques constructores de cualquier base de datos que conforman el modelo).
- Una colección de operadores o reglas de inferencia, los cuales pueden ser aplicados a cualquier instancia de los tipos de datos listados para consultar o derivar datos de cualquier parte de estas estructuras en cualquier combinación deseada.
- Una colección de reglas generales de integridad, las cuales explícita o implícitamente definen un conjunto de estados consistentes, estas reglas algunas veces son expresadas como reglas de insertar-actualizar-borrar.

Un modelo de datos puede ser usado de las siguientes maneras:

- Como una herramienta para especificar los tipos de datos y la organización de los mismos que son permisibles en una base de datos específica.
- Como una base para el desarrollo de una metodología general de diseño para las bases de datos.
- Como una base para el desarrollo de familias de lenguajes de alto nivel para manipulación de consultas (querys) y datos.
- Como el elemento clave en el diseño de la arquitectura de un manejador de bases de datos.

El primer modelo de datos desarrollado con toda la formalidad que esto implica fue el modelo relacional, en 1969, mucho antes incluso que los modelos jerárquicos y de red. A pesar de que los sistemas jerárquicos y de red como software para manejar bases de datos son previos al modelo relacional, no fue sino hasta 1973 que los modelos de tales sistemas fueron definidos, apenas unos cuantos años antes de que estos sistemas empezaran a caer en desuso.

Algunos modelos con frecuencia utilizados en las bases de datos son:

Bases de datos jerárquicas: Éstas son bases de datos que, como su nombre indica, almacenan su información en una estructura jerárquica. En este modelo los datos se organizan en una forma similar a un árbol (visto al revés), en donde un nodo padre de información puede tener varios hijos. El nodo que no tiene padres es llamado raíz, y a los nodos que no tienen hijos se los conoce como hojas.

Las bases de datos jerárquicas son especialmente útiles en el caso de aplicaciones que manejan un gran volumen de información y datos muy compartidos permitiendo crear estructuras estables y de gran rendimiento.

Una de las principales limitaciones de este modelo es su incapacidad de representar eficientemente la redundancia de datos.

Ejemplo:

1.- El paciente Armando Martínez de 45 años tiene dos citas, la número 12 con el médico general y la número 3 con el neurólogo.

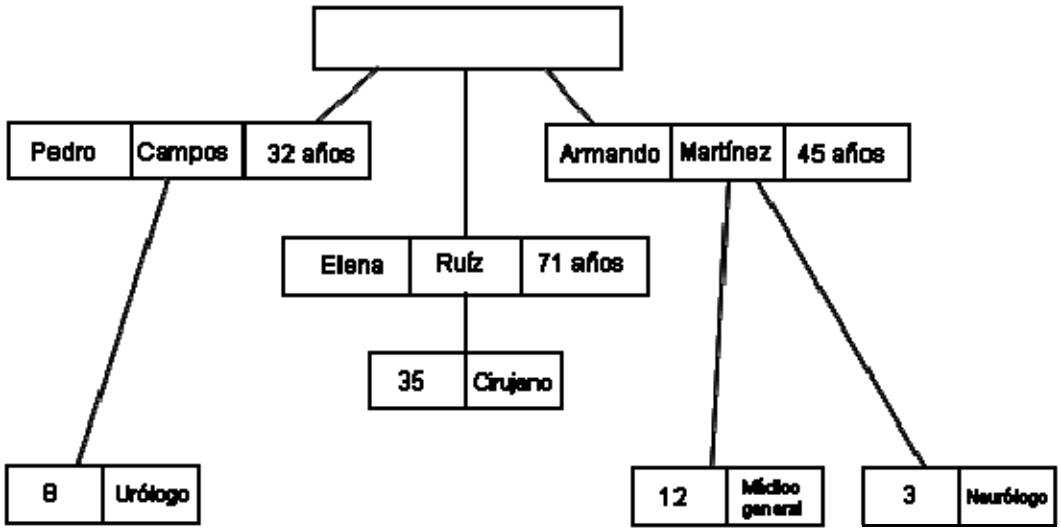


Figura 2.1. *Ejemplo de una base de datos jerárquica para citas médicas.*

2.- A continuación se muestran tres registros, uno de ellos, el cliente Cruz vive en la calle Acacias, en Naucalpan, que tiene dos cuentas, la número 647 con un saldo de $105, 366 y la número 801 con un saldo de $10, 533. Nótese que los clientes Pérez y Cruz comparten la cuenta número 647 (posiblemente son socios de la misma empresa).

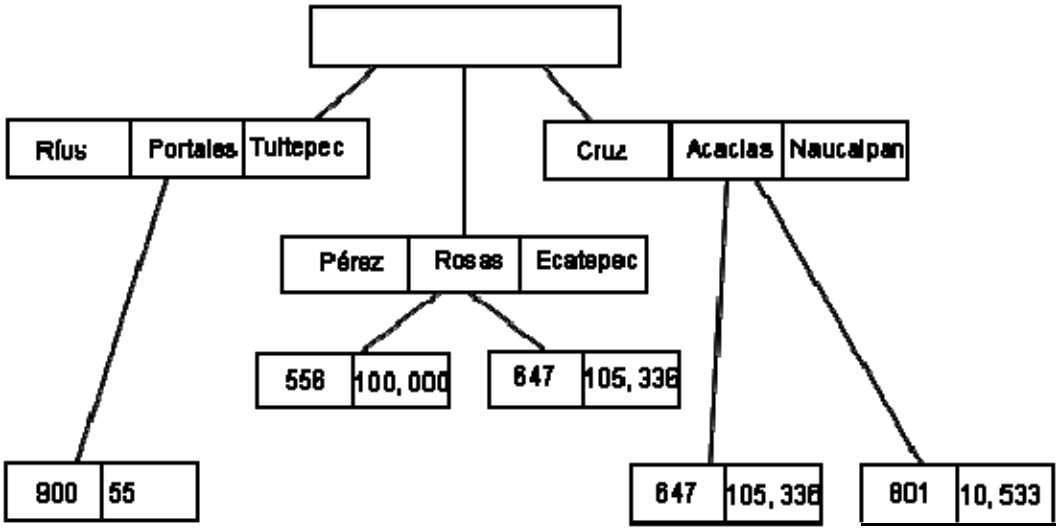


Figura 2.2. *Ejemplo de una base de datos jerárquica en cuentas de un banco.*

Base de datos de red: Éste es un modelo ligeramente distinto del jerárquico; su diferencia fundamental es la modificación del concepto de nodo: se permite que un mismo nodo tenga varios padres (posibilidad no permitida en el modelo jerárquico).

Fue una gran mejora con respecto al modelo jerárquico, ya que ofrecía una solución eficiente al problema de redundancia de datos; pero, aun así, la dificultad que significa administrar la información en una base de datos de red ha significado que sea un modelo utilizado en su mayoría por programadores más que por usuarios finales. Las figuras 2.3 y 2.4 muestran un ejemplo de una base de datos de red con la misma información de los ejemplos anteriores.

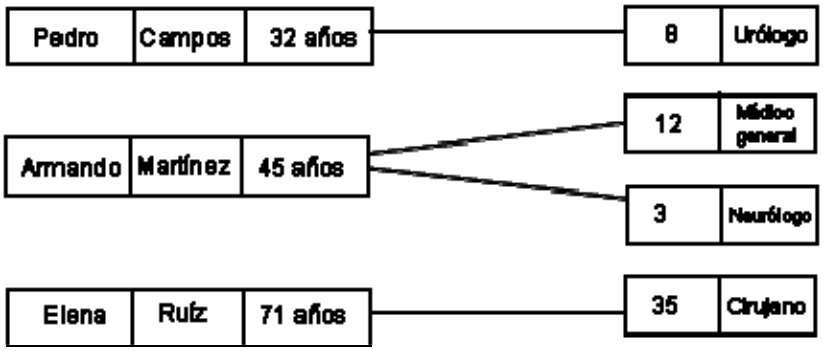


Figura 2.3. *Ejemplo de una base de datos de red para citas médicas.*

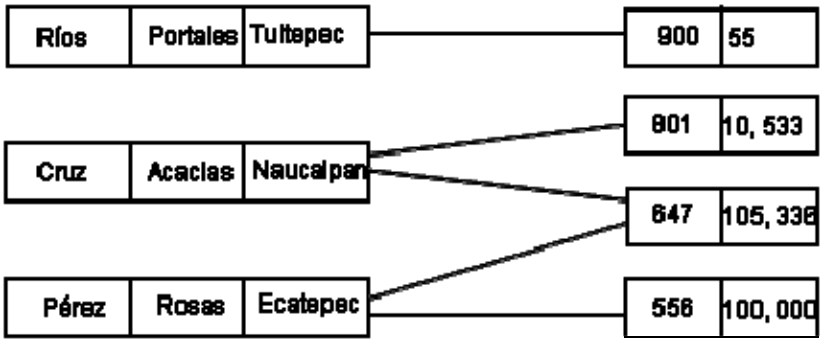


Figura 2.4. *Ejemplo de una base de datos de red para las cuentas de un banco.*

Base de datos relacional: Éste es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente. Tras ser postulados sus fundamentos en 1970 por Edgar Frank Codd, de los laboratorios IBM en San José (California), no tardó en consolidarse como un nuevo paradigma en los modelos de base de datos. Su idea fundamental es el uso de "relaciones". Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados "tuplas". Pese a que ésta es la teoría de las bases de datos relacionales creadas por Edgar Frank Codd, la mayoría de las veces se conceptualiza de una manera más fácil de imaginar. Esto es pensando en cada relación como si fuese una tabla que está compuesta por registros (las filas de una tabla), que representarían las tuplas, y campos (las columnas de una tabla).

En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia (a diferencia de otros modelos como el jerárquico y el de red). Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar para un usuario esporádico de la base de datos. La información puede ser recuperada o almacenada mediante "consultas" que ofrecen una amplia flexibilidad y poder para administrar la información.

A continuación se muestran dos bases de datos relacionales con la misma información de los ejemplos anteriores.

nombre	calle	ciudad	número
Ríos	Portales	Tultepec	900
Cruz	Acacias	Naucalpan	801
Cruz	Acacias	Naucalpan	647
Pérez	Rosas	Ecatepec	556
Pérez	Rosas	Ecatepec	647

número	saldo
900	55
556	100,000
647	105,336
801	10,533

Figura 2.5. *Ejemplo de una base de datos relacional para las cuentas de un banco.*

nombre	apellido	edad	médico
Pedro	Campos	32 años	Urólogo
Armando	Martínez	45 años	M. General
Armando	Martínez	45 años	Neurólogo
Elena	Ruiz	71 años	Cirujano

cita	médco
35	Cirujano
8	Urólogo
12	Médico General
2	Neurólogo

Figura 2.6. *Ejemplo de una base de datos relacional para citas médicas.*

El lenguaje más habitual para construir las consultas a bases de datos relacionales es SQL, Structured Query Language o Lenguaje Estructurado de Consultas, un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales.

Durante su diseño, una base de datos relacional pasa por un proceso al que se le conoce como normalización de una base de datos.

2.2. MODELO DE BASES DE DATOS JERÁRQUICO

2.2.1. Clases de objetos

En el modelo de red, la información se representa por medio de conjuntos de registros, y las relaciones entre los datos se representan por ligas. El modelo jerárquico es similar al de red en que los datos y las relaciones entre ellos se representan también por medio de registros y ligas, respectivamente. El modelo jerárquico difiere del de red en cuanto a que los registros se organizan para formar conjuntos de árboles, en vez de gráficas arbitrarias. Gráficamente, se muestra el modelo jerárquico de datos como un árbol donde el nivel más alto se conoce como la raíz y los nodos del árbol representan las entidades.

2.2.2. Relaciones padre-hijo

El modelo jerárquico relaciona entidades por medio de una relación superior/subordinado o padre/hijo. Por ejemplo, un diagrama de la organización muestra los niveles de ejecutivos, gerentes medios y personal de operación.

Las relaciones entre padre/hijo sólo pueden ser uno a muchos o uno a uno. En este tipo de relaciones, los hijos heredan comportamientos del padre.

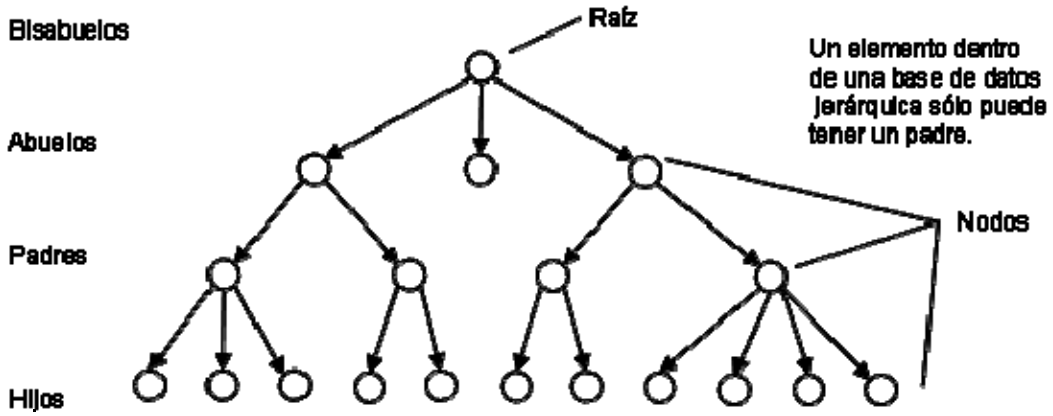


Figura 2.7. *Ejemplo de relación padre/hijo.*

2.2.3. Propiedades del esquema

- El árbol se organiza en un conjunto de niveles.
- El nodo raíz, el más alto de la jerarquía, se corresponde con el nivel 0 (algunos autores consideran que se trata del nivel 1).
- Los arcos representan las asociaciones jerárquicas entre dos entidades y no tienen nombre, ya que no es necesario porque entre dos conjuntos de datos sólo puede haber una interrelación.
- Mientras que un nodo de nivel superior (padre) puede tener un número ilimitado de nodos de nivel inferior (hijos), al nodo del nivel inferior sólo le puede corresponder un único nodo de nivel superior. En otras palabras, un progenitor (o padre) puede tener varios descendientes (o hijos), pero un hijo sólo tiene un padre.
- Todo nodo, a excepción del nodo raíz, ha de tener obligatoriamente un padre.
- Se llaman hojas los nodos que no tienen descendientes.
- Se llama altura al número de niveles de la estructura jerárquica.
- Se denomina momento al número de nodos.
- Sólo están permitidas las interrelaciones 1: 1 ó 1:N.
- Cada nodo no terminal y sus descendientes forman un subárbol, de forma que un árbol es una estructura recursiva.

La figura 2.8 muestra una estructura jerárquica, con las características antes mencionadas.

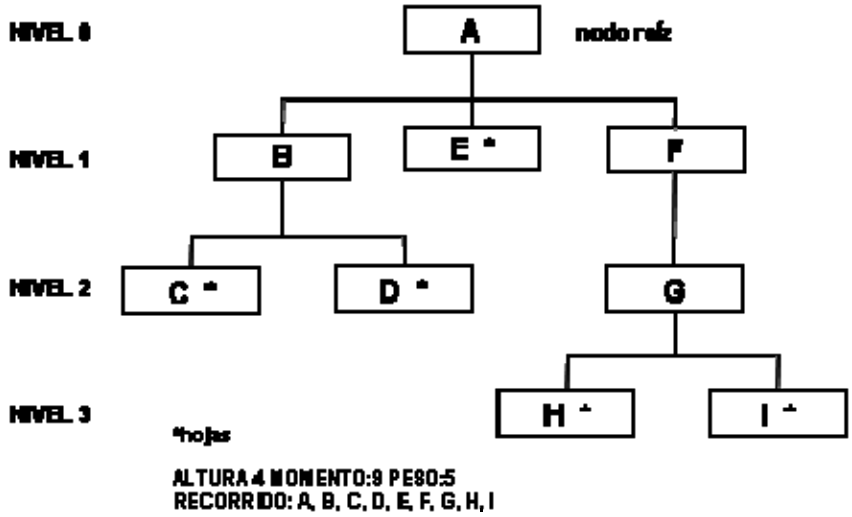


Figura 2.8. *Ejemplo de una estructura jerárquica*

2.2.4. Árboles

Un diagrama de estructura de árbol es el esquema de una base de datos jerárquica. Este tipo de diagrama está formado por dos componentes básicos:

- Cuadros, que corresponden a tipos de registro.
- Líneas, que corresponden a ligas.

Un diagrama de estructura de árbol tiene misma función que un diagrama de entidad-relación (o que uno de estructura de datos en el modelo de red); es decir, especificar la estructura lógica general de la base de datos. Un diagrama de estructura de árbol es similar a uno de estructura de datos. La diferencia principal radica en que en este último los tipos de registro se organizan en forma de una gráfica arbitraria, mientras que en el primero se organizan en forma de un árbol con raíz.

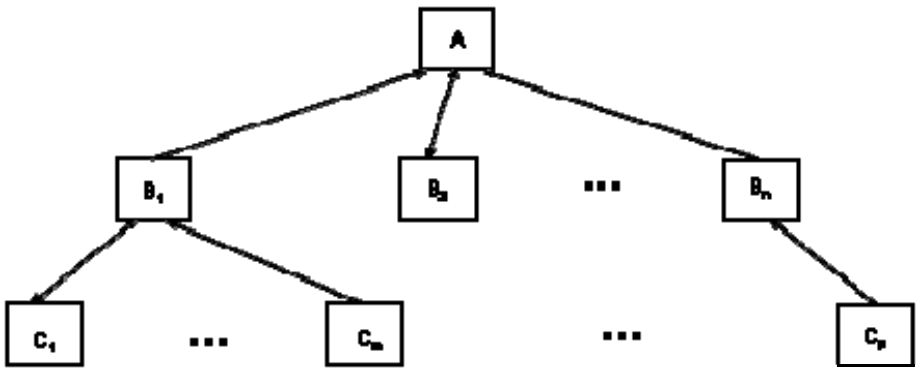


Figura 2.9. *Modelo general de un diagrama de estructura de árbol.*

Es necesario definir precisamente lo que quiere decir el término árbol con raíz. En primer lugar, la gráfica no puede contener ciclos. En segundo lugar, las relaciones entre padre e hijo sólo pueden ser uno a muchos o uno a uno.

- **Relación uno a uno:** Una entidad en un nivel se relaciona con una entidad en el siguiente nivel.

- **Relación uno a muchos:** Una entidad en un nivel se relaciona con una, muchas o ninguna entidad en el siguiente nivel.

El esquema de la base de datos se representa como un conjunto de diagramas de estructura de árbol. Para cada diagrama existe únicamente una instancia del árbol de base de datos. La raíz de este árbol es un nodo instrumental. Los hijos de ese nodo son instancias del tipo de registro correspondiente. Cada una de estas instancias puede tener a su vez varias instancias de diversos tipos de registro, según se especifica en el diagrama de estructura de árbol correspondiente.

2.2.5. Estructura de almacenamiento

El analista de sistemas se ve afectado por las decisiones hechas al diseñar una base de datos jerárquica. Durante el diseño, el administrador de la base de datos, quién es responsable del diseño, determina las entidades a incluir en la base de datos y la relación que existirá entre ellas. Los nodos representan ocurrencias de registros que contienen los datos tal como determina el administrador de los datos.

El diseño de una base de datos jerárquica afectará la facilidad de acceso a los datos (las notas de acceso no necesitan hacerse al utilizar bases de datos relacionales). Por ejemplo, la figura 2.10 indica que las citas siempre están relacionadas con un paciente específico. Es muy eficiente tener las citas de un cierto paciente o los médicos en un orden particular. La base de datos se diseña para soportar estas consultas ya que, son las que aparecen con más frecuencia.

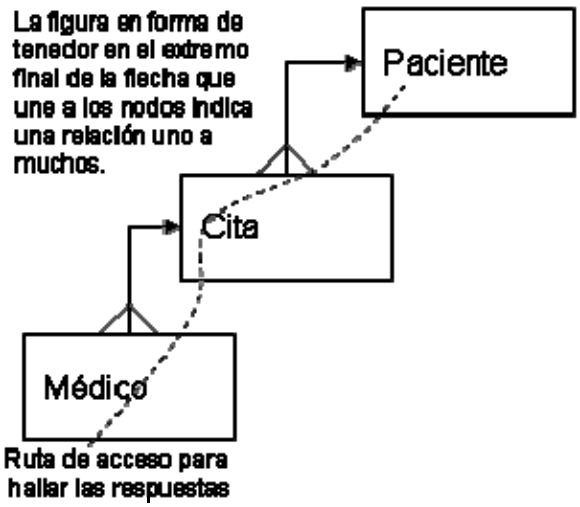


Figura 2.10. *Estructura jerárquica para la base de datos de citas médicas.*

El analista de sistemas debe trabajar con las restricciones de diseño que surjan. Por ejemplo, el modelo de datos estipula que los artículos son accesibles sólo mediante la aparición de un registro de pedido. Esta relación indica que se debe

buscar toda la lista de artículos para todos los pedidos para preparar un reporte de la venta histórica de los artículos.

Aparecen efectos colaterales indeseables bajo ciertos diseños de una base de datos. Las bases de datos jerárquicas involucran anomalías con respecto a lo siguiente:

- Inserción de registros: Un registro dependiente no se puede añadir a la base de datos sin un padre. Ejemplo: Los artículos no se pueden añadir sin incluirlos en un pedido.

- Borrado de registros: Al borrar un padre de la base de datos también se borran todos sus descendientes. Ejemplo: Al quitar un cliente también se quitan los pedidos pendientes.

Si estas situaciones ocurren con cierta probabilidad, en el marco de una aplicación particular, es necesario establecer múltiples copias de los registros e incluso bases de datos múltiples lo cual añade redundancia y complejidad adicional para evitar el problema.

2.2.6. Tipos de acceso de integridad y seguridad del modelo

La manipulación de datos jerárquicos, al igual que ocurre en todo modelo, necesita, al menos en un plano de abstracción, localizar (seleccionar) primero los datos sobre los que va a trabajar para realizar a continuación la acción de recuperación o actualización sobre dichos datos.

La función de selección jerárquica es de tipo navegacional, es decir, trabaja registro a registro, en oposición a otros modelos, como el relacional, en los que se seleccionan conjuntos de registros. Dada la sencillez del modelo, la función de selección es también muy sencilla, existiendo únicamente las siguientes formas básicas de búsqueda:

- Seleccionar un determinado registro que cumpla una cierta condición.

- Seleccionar el siguiente registro, que se encuentra perfectamente definido al existir un único camino jerárquico (preorden).

- Seleccionar el siguiente registro dentro de un padre. Esta sentencia es análoga a la anterior, pero la selección termina cuando no haya más descendientes de ese padre.

- Seleccionar el registro padre de otro lado (que ha sido activado previamente), se conoce como normalización jerárquica ascendente, mientras que la selección de descendientes se llama normalización jerárquica descendente.

Puesto que una instrucción jerárquica selecciona un único registro, será preciso que el lenguaje de datos esté embebido en un lenguaje de programación mediante el cual se describa el procedimiento necesario para ir recorriendo el árbol con el fin de buscar y manipular los datos.

Una vez seleccionado un registro, se tendrá que realizar sobre él una acción, sea de recuperación o de actualización.

La recuperación consiste, como en cualquier otro lenguaje de datos, en llevar el registro marcado como activo, en la selección realizada previamente, al área de entrada/salida.

En cuanto a la actualización, es preciso distinguir entre:

- Insertar un conjunto de datos.
- Borrar un conjunto de datos.
- Reemplazar -modificar- uno o varios campos de un registro.

Debido a la naturaleza jerárquica de las conexiones entre registros, las inserciones y borrados de registros requieren consideraciones especiales:

Cuando un nuevo registro se inserta en una base de datos jerárquica, excepto para la raíz, se conecta automáticamente a un nodo padre previamente localizado mediante alguna sentencia de selección. El nuevo registro se inserta como hijo del registro padre seleccionado.

2.2.7. Base de datos jerárquica

Como ya se menciono, una base de datos jerárquica consiste en un conjunto de registros que se conectan entre sí por medio de ligas. Para ilustrar lo anterior, piénsese en una base de datos que representa a una relación cuenta habiente-cuenta en un sistema bancario. Existen dos tipos de registro: cuenta habiente y cuenta. Consta de tres campos: nombre, calle y ciudad. De manera similar, el registro cuenta consiste en dos campos: número y saldo.

En la figura 2.11 se presenta un ejemplo de esta base de datos. Indica que el cuenta habiente Flores tiene la cuenta 305, Oca, las 226 y 177, Y Díaz, la 155.

Nótese que el conjunto de todos los registros de cuenta habientes y cuentas está organizado en forma de un árbol con raíz, en el cual esta última es un nodo de trabajo. Como ya se ha visto, una base de datos jerárquica está formada por un conjunto de árboles de este tipo, formando así un bosque. En este texto, un árbol de esta clase, dotado de raíz, se denominará árbol de base de·datos.

El contenido de un registro específico puede repetirse en varios lugares. Por ejemplo, en el sistema bancario cuenta habiente-cuenta, una cuenta puede pertenecer a varios clientes. La información correspondiente a esa cuenta, o la relativa a los cuenta habientes a los que puede pertenecer, tendrá que repetirse. Esta repetición puede darse tanto en el mismo árbol de base de datos corno en varios árboles distintos. La repetición de registros tiene dos desventajas principales:

- Puede producirse una inconsistencia de los datos al llevar a cabo la actualización.
- Será inevitable el desperdicio de espacio.

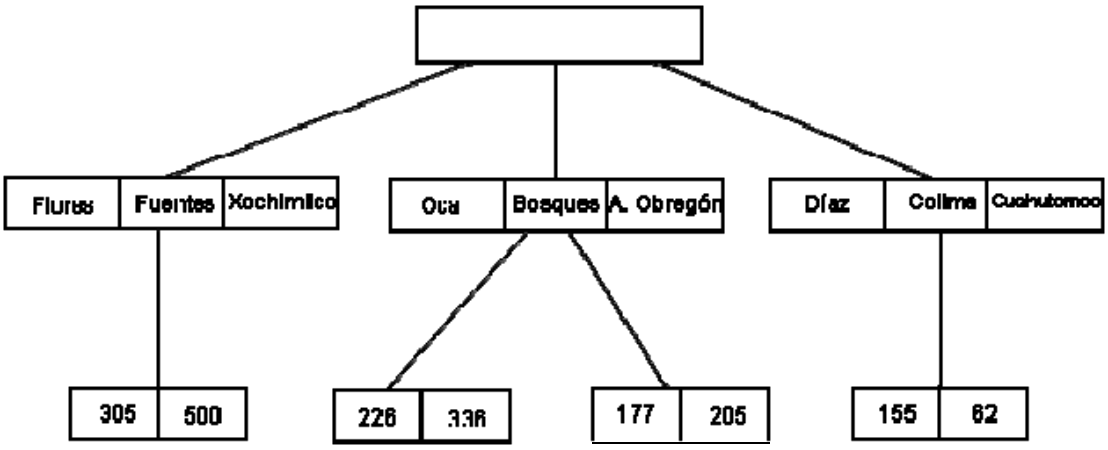


Figura 2.11. *Ejemplo de base de datos jerárquica.*

2.2.8. Problemas del modelo jerárquico

El modelo de datos jerárquico presenta importantes inconvenientes, que provienen principalmente de su rigidez, la cual deriva de la falta de capacidad de las organizaciones jerárquicas para representar sin redundancias ciertas estructuras muy difundidas en la realidad, como son las interrelaciones reflexivas y N:M, las redes, etc. En la figura 2.12 se muestran algunos ejemplos de estructuras no admitidas en el modelo jerárquico.

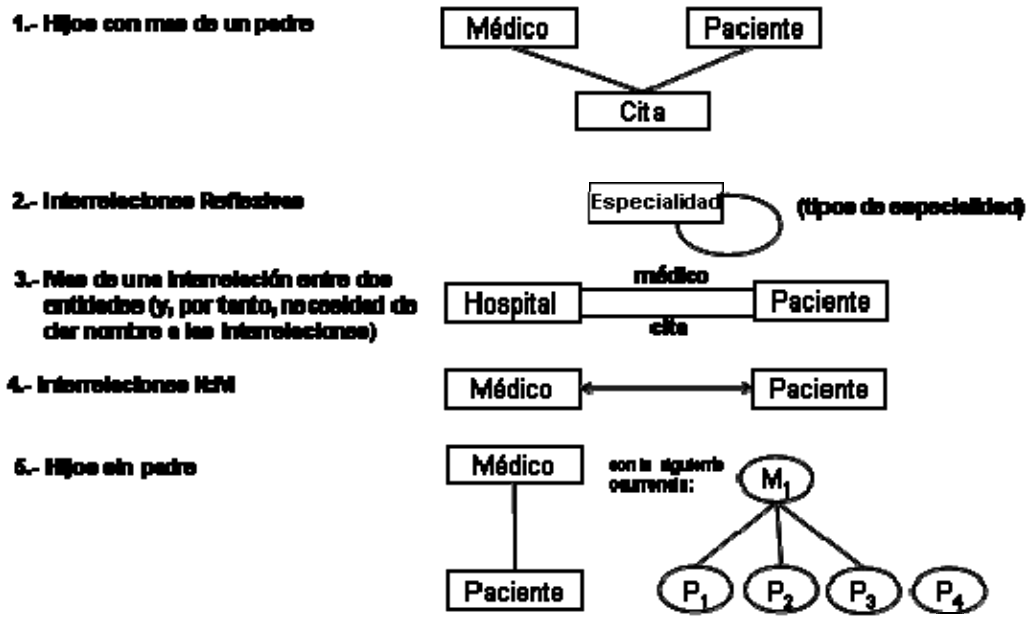


Figura 2.12. *Algunas estructuras no admitidas en el modelo jerárquico.*

La poca flexibilidad de este modelo puede obligar a la introducción redundancias cuando es preciso instrumentar, mediante el modelo jerárquico situaciones del mundo real que no responden a una jerarquía.

Se puede calcular un índice de redundancia (IR) mediante la siguiente fórmula:

IR = (No. nodos extra/ No. total de nodo) * 100

x 100

Este índice de redundancia permite determinar hasta qué punto una estructura del mundo real se adapta más o menos a un árbol (cuanto mayor es el índice de redundancia, más lejos se está de una estructura en árbol).

La redundancia que hemos calculado es una redundancia lógica que, en general, no coincide con la redundancia física. Esto es debido a que, al almacenar los datos en el dispositivo físico, no se suelen repetir los registros completos, sino sólo identificadores, o bien se introducen punteros. Se trata ya de soluciones de instrumentación propias de cada suministrador. En general, los productos ofrecen algún tipo de solución a estos problemas.

Otra limitación importante del modelo jerárquico es que no está preparado para representar interrelaciones N:M.

Además del grave problema que provocan estas redundancias no controladas por el sistema (con el aumento de almacenamiento y sobre todo las posibles inconsistencias), existe otro importante inconveniente en este tipo de solución como es la no conservación de las simetrías naturales existentes en el mundo real. Las actualizaciones en las bases de datos jerárquicas pueden también originar problemas debido a las restricciones inherentes al modelo.

En el caso de las relaciones muchos a muchos, es menester repetir registros si se desea conservar la organización de estructura de árbol de la base de datos. La repetición de registros tiene dos inconvenientes importantes: la actualización de registros puede producir inconsistencia de los datos, y se desperdicia espacio de almacenamiento. La solución es introducir el concepto de registro virtual. Un registro virtual no contiene valores, sino un apuntador lógico a un registro físico determinado. Cuando es necesario repetir un registro en varios árboles de base de datos, se conserva una sola copia del registro en uno de los árboles y se sustituyen las demás copias por registros virtuales que contienen un apuntador a ese registro físico. El lenguaje de manipulación de datos para esta nueva configuración sigue siendo el mismo que en el caso en que se permite la repetición de registros, por lo que no es necesario que el usuario se dé cuenta de estos cambios. Sólo se afecta la implantación interna.

Los SGBD basados en el modelo jerárquico suelen facilitar instrumentos que los dotan de una mayor flexibilidad para representar estructuras no estrictamente jerárquicas. Sin embargo, puede ocurrir que dicha implementación no proporcione la debida independencia física/lógica.

En cuanto a las ventajas proporcionadas por los SGBD de tipo jerárquico, cabe citar su sencillez de comprensión y la mayor facilidad de instrumentación en los soportes físicos, aunque esto depende en gran medida de los productos. Además, si se trata de estructuras del mundo real que sean de tipo jerárquico, este modelo puede ser muy idóneo. Por otra parte, los productos jerárquicos suelen ser muy eficientes.

2.2.9. Resumen

Una base de datos jerárquica consiste en un conjunto de registros que se conectan entre sí por medio de ligas. Un registro es un conjunto de campos, cada uno de los cuales contiene un sólo valor. Una liga es una asociación entre dos registros, exclusivamente. El modelo jerárquico es, pues, similar al de red, en cuanto que los datos y las relaciones entre ellos se representan por medio de registros y ligas, respectivamente. El modelo jerárquico difiere del de red en que los registros se organizan en árboles en vez de gráficas arbitrarias.

Un diagrama de estructura de árbol es el esquema de una base de datos jerárquica, y consta de dos componentes básicos: cuadros, que corresponden a tipos de registro, y líneas, que corresponden a ligas. Un diagrama de estructura de árbol tiene el mismo objetivo que uno de entidad-relación, es decir, especifica la estructura lógica general de la base de datos. Los diagramas de estructura de árbol son similares a los de estructura de datos del modelo de red. La diferencia principal es que en los últimos los tipos de registro se organizan en forma de gráficas arbitrarias, mientras que en los primeros se organizan en forma de árboles con raíz. Por cada diagrama de entidad-relación existe uno de estructura de árbol correspondiente.

Por tanto, el esquema de la base de datos se representa como un conjunto de diagramas de estructura de árbol. Por cada uno de estos diagramas, existe una sola instancia del árbol de base de datos. La raíz de este árbol es un nodo de trabajo. Los hijos de ese nodo son instancias reales del tipo de registro correspondiente. Cada una de estas instancias puede tener a su vez varias instancias de tipos de registro, según se especifique en el diagrama de estructura de árbol correspondiente.

Se puede localizar y manipular los elementos de la base de datos, y también variables definidas en forma local. El sistema mantiene un área de trabajo de programa para cada programa de aplicación, la cual contiene plantillas de registros, apuntadores de actualidad y una bandera de situación.

La recuperación de datos de la base de datos se lleva a cabo obteniendo o localizando un registro en la base de datos que cambia el apuntador de actualidad para que señale a ese registro. A continuación, copia ese registro en la plantilla apropiada en el área de trabajo del programa.

Se cuenta con varios mecanismos para actualizar la información de la base de, datos. Éstos incluyen la creación y eliminación de registros, así como la modificación del contenido de registros existentes.

En el caso de las relaciones muchos a muchos, es menester repetir registros si se desea conservar la organización de estructura de árbol de la base de datos. La repetición de registros tiene dos inconvenientes importantes: la actualización de registros puede producir inconsistencia de los datos, y se desperdicia espacio de almacenamiento. La solución es introducir el concepto de registro virtual. Un registro virtual no contiene valores, sino un apuntador lógico a un registro físico determinado. Cuando es necesario repetir un registro en varios árboles de base de datos, se conserva una sola copia del registro en uno de los árboles y se sustituyen las demás copias por registros virtuales que contienen un apuntador a ese registro físico. El lenguaje de manipulación de datos para esta nueva configuración sigue siendo el mismo que en el caso en que se permite la repetición de registros, por lo que no es necesario que el usuario se dé cuenta de estos cambios. Sólo se afecta la implantación interna.

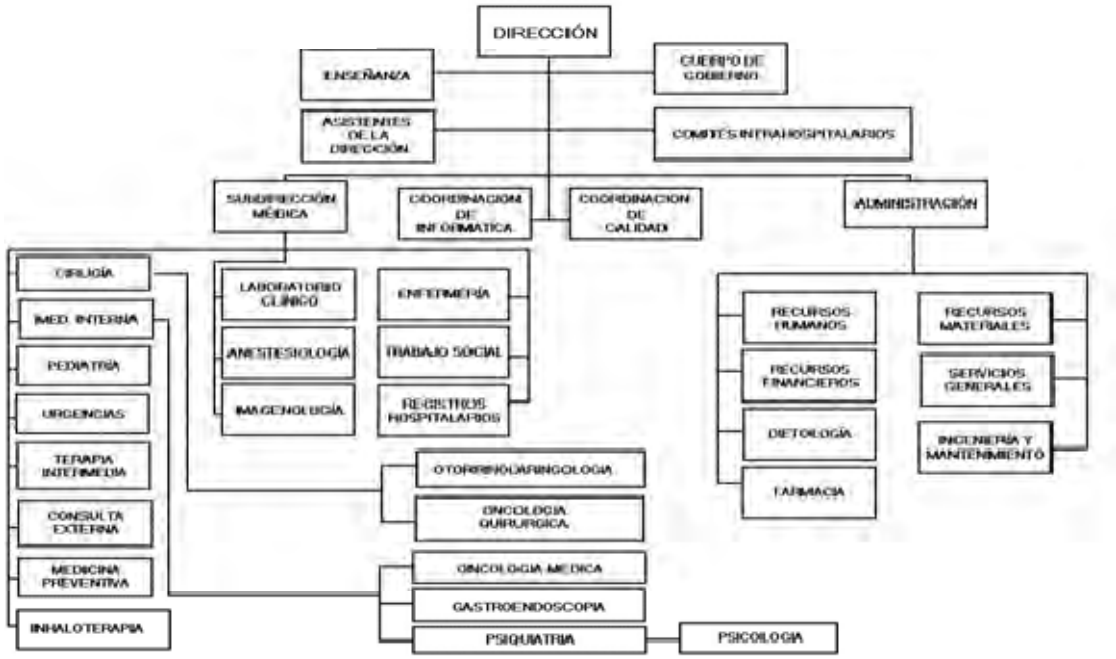


Figura 2.13. *Árbol jerárquico de un hospital.*

2.3. MODELO DE BASES DE DATOS DE RED

2.3.1. Estructura

El modelo de red es análogo al modelo jerárquico, excepto que una entidad puede tener más de un padre. Los miembros pueden pertenecer a más de una relación (es decir, tienen más de un poseedor).

Los modelos de datos en red representan las entidades en forma de nodos de un grafo, y las asociaciones o interrelaciones entre éstas, mediante los arcos que unen dichos nodos. Esta representación, que no impone en principio ninguna restricción ni al tipo ni al número de los arcos, permite el modelado de estructuras de datos tan complejas como se desee.

Podríamos definir un modelo en red general, con una mayor formalización como un conjunto finito de tipos de entidad:

$$\{E_1, E_2, \ldots, E_n\}$$

Donde cada entidad $E_¡$ tiene un conjunto de propiedades (atributos):

$$\{A_{i1}, A_{i2}, \ldots, A_{ik}\}$$

Y un conjunto finito de tipos de interrelación:

$$\{I^h_{j,k,\ldots,n}\}$$

Entre los elementos j, k, ... , n (entidades u otras interrelaciones), donde el superíndice h permite diferenciar dos interrelaciones distintas entre los mismos elementos, ya que se refiere al nombre de la interrelación (también las entidades y los atributos tiene nombre).

La forma de representación de estos modelos son los grafos.

Una base de datos de red consiste en una serie de registros que están conectados entre sí por medio de ligas (links). Un registro es, en muchos aspectos, similar a una entidad en el modelo de entidad-relación. Todo registro es un conjunto de campos (atributos), cada uno de éstos contiene únicamente el valor de un dato. Una liga es una asociación entre dos registros exclusivamente. Así pues, una liga puede considerarse como una forma restringida (binaria) de relación en el sentido del modelo E-R.

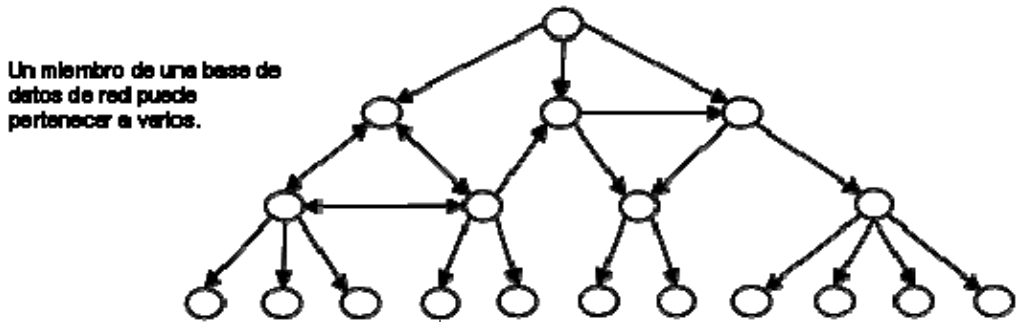


Figura 2.14. *Estructura básica del modelo de datos de red.*

La estructura de una base de datos de red que aparece en la figura 2.15 muestra las entidades en la base de datos de las citas médicas que usamos en el modelo jerárquico. Sin embargo, la relación entre las entidades es distinta.

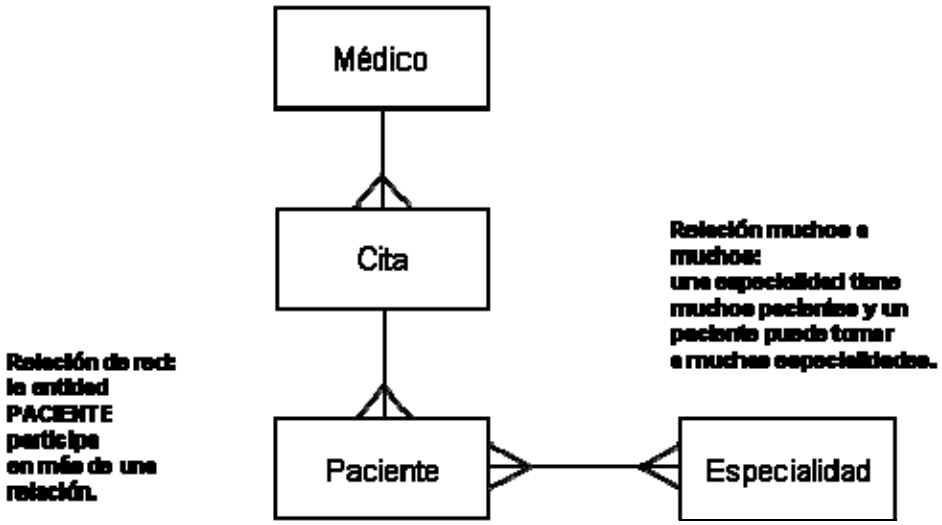


Figura 2.15. *Estructura de red para la base de datos de citas médicas.*

En el ejemplo de las citas médicas, se puede mostrar una relación entre los médicos y las citas, al igual que entre pacientes y especialidades. Esta capacidad introduce el uso de un tipo adicional de relación entre los datos:

- Muchos a muchos: Una entidad se puede relacionar con una, muchas o ninguna entidad en otro nivel.

En las bases de datos de tipo red, así como en las jerárquicas, se deben establecer las relaciones entre las entidades al mismo tiempo que se establece el modelo de los datos y se crea la base de datos (encontraste con el modelo relacional, el cual no requiere rutas de acceso predefinidas o relaciones entre las entidades). El analista de sistemas debe ajustarse a esos detalles cuando desarrolla aplicaciones que capturan o recuperan datos durante el procesamiento.

Las bases de datos jerárquicas y de red son conceptualmente sencillas y parecen no ser complicadas a primera vista. Sin embargo, en un ambiente de una base de datos grande, pueden evolucionar rápidamente hacia una telaraña complicada de interrelaciones que son difíciles de manejar al evolucionar la base de datos con el uso.

Un diagrama de estructura de datos es un esquema que representa el diseño de una base de datos de red. Estos diagramas están formados por dos componentes básicos:

- Cuadros, que corresponden a tipos de registro.
- Líneas, que corresponden a ligas.

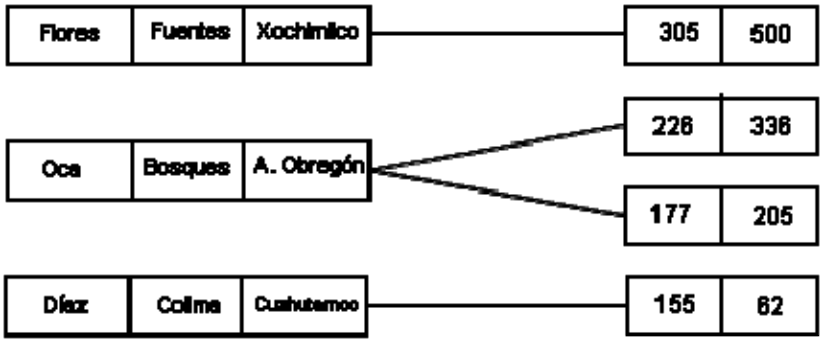


Figura 2.16. *Ejemplo de base de datos de red para un banco.*

Un diagrama de estructura de datos tiene el mismo objetivo general que uno de entidad-relación; a saber, especifica la estructura lógica general de la base de datos. Para comprender cómo se construyen estos diagramas, se mostrará cómo pueden transformarse los diagramas de entidad-relación en los de estructura de datos correspondientes.

Aparecen anomalías similares a las del modelo jerárquico de datos. Así, si un pedido se cancela, no queremos eliminar al cliente, aunque el modelo sugiere que esto podría ocurrir.

Existe un lado positivo de los modelos de datos jerárquico y de red, el cual deben observar los analistas de sistemas. Supongamos que se pueden predefinir las rutas de acceso y relaciones entre las entidades (cuando el esquema se desarrolla y se crea la base de datos). Si los requerimientos de acceso y recuperación del desarrollo de la aplicación se adecuan a las rutas de acceso predefinidas, será más rápido el procesamiento de consultas, actualizaciones y adiciones a la base de datos que cuando se utilizan bases de datos relacionales.

Los analistas de sistemas también reconocen que existen muchísimas bases de datos basadas en los modelos jerárquico y de red instaladas en la actualidad en la comunidad empresarial. Si estas bases de datos están cumpliendo los requerimientos operacionales, es improbable que las organizaciones las reemplacen. Así, el desarrollo de las aplicaciones de sistemas de información que tomen en cuenta las oportunidades y restricciones que ofrecen es una necesidad.

2.3.2. Registros, campos

Dado que solamente pueden utilizarse ligas del tipo uno a uno y muchos a uno, un diagrama de estructura de datos que consiste en dos tipos de registro ligados entre sí tiene la forma general que puede verse en la figura siguiente.

Figura 2.17. *Conjunto DBTG.*

El modelo DBTG en inglés; Data Base Task Group (Grupo de Trabajo Sobre Bases de Datos) es ya una implementaron de las reglas generales de operación del modelo de red; toma de este los aspectos generales operativos.

Solo pueden utilizarse ligas muchos a uno. Las ligas uno a uno se representan utilizando ligas muchos a uno.

Las ligas son representaciones mas concisas del modelo de red; se representan genéricamente entre registros. El nombre del conjunto generalmente es el mismo que el de la liga que conecta a los dos tipos de registro.

En todo conjunto DBTG de este tipo, el tipo de registro A se denomina el dueño o padre (en inglés: owner o parent) del conjunto. Y el tipo de registro B se denomina miembro o hijo (en inglés: member o child) del conjunto. Cada conjunto DBTG puede tener cualquier número de ocurrencias del conjunto, es decir, instancias reales de registros ligados.

Puesto que no se permiten ligas del tipo muchos a muchos, cada ocurrencia del conjunto tiene exclusivamente un dueño y cero o más registros miembro. Además, ningún registro miembro puede participar en más de una ocurrencia del conjunto en ningún momento. Sin embargo, un registro miembro sí puede participar simultáneamente en varias ocurrencias de diferentes conjuntos DBTG.

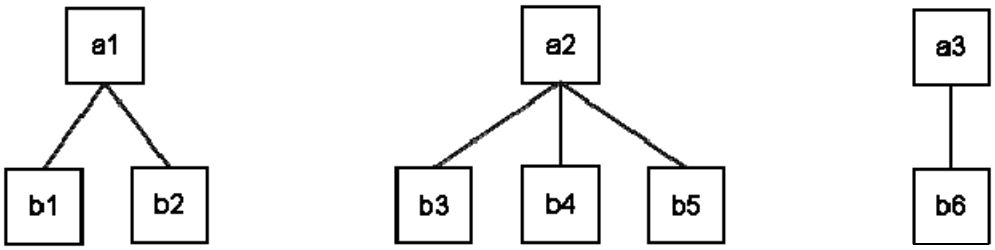


Figura 2.18. *Tres ocurrencias de un conjunto.*

Los registros miembro de una ocurrencia de un conjunto pueden ordenarse en diversas formas. El modelo DBTG permite estructuras de conjuntos más complejas en las que existe un sólo tipo dueño y varios tipos miembro.

El modelo DBTG también prevé la definición de un conjunto especial, denominado conjunto singular (o conjunto de sistema). En un conjunto de este tipo, el dueño es un tipo de registro único definido por el sistema, llamado system, que no tiene campos.

Este conjunto tiene una sola ocurrencia de conjunto. Este esquema es útil para buscar registros de un tipo determinado.

Grupos repetidos: El modelo DBTG cuenta con un mecanismo que permite a un campo (o grupo de campos) tener un conjunto de valores, en vez de un solo valor. Por ejemplo, supóngase que un cuenta habiente tiene varios domicilios. En este caso, se definirá la pareja de campos (calle, ciudad) del tipo de registro cuenta habiente como un grupo repetido. De esta manera, el registro cuenta habiente de Kahn puede ser como el que se muestra en la figura 2.19.

Díaz	Colima	Cuauhtémoc
	Olivos	A. Obregón

Figura 2.19. *Ejemplo de grupos repetidos.*

La construcción de grupos repetidos es otra forma de representar el concepto de entidades débiles en el modelo E-R.

2.3.4. Resumen

Los modelos de datos en red representan las entidades en forma de nodos de un grafo, y las asociaciones o interrelaciones entre éstas, mediante los arcos que unen dichos nodos. Esta representación, que no impone en principio ninguna restricción ni al tipo ni al número de los arcos, permite el modelado de estructuras de datos tan complejas como se desee.

Podríamos definir un modelo en red general, con una mayor formalización como un conjunto finito de tipos de entidad:

$$\{E_1, E_2, \ldots, E_n\}$$

Donde cada entidad $E_¡$ tiene un conjunto de propiedades (atributos):

$$\{A_{i1}, A_{i2}, \ldots, A_{ik}\}$$

Y un conjunto finito de tipos de interrelación:

$$\{I^h_{j,k,\ldots,n}\}$$

Entre los elementos j, k, ... , n (entidades u otras interrelaciones), donde el superíndice h permite diferenciar dos interrelaciones distintas entre los mismos elementos, ya que se refiere al nombre de la interrelación (también las entidades y los atributos tiene nombre).

La forma de representación de estos modelos son los grafos.

El modelo relacional describe los aspectos estáticos, es decir, la parte estructural de los datos (tipos de entidades, tipos de interrelaciones, etc.), representadas en forma de grafo, y las restricciones; mientras que una ocurrencia del esquema (base de datos) son los valores que toman los elementos del esquema en un determinado momento, los cuales irán variando a lo largo del tiempo por el efecto de aplicar los operadores de manipulación de datos a una ocurrencia del mismo.

En cuanto a la dinámica, los modelos en red se caracterizan por ser navegacionales, es decir, la recuperación y la actualización de la base de datos se lleva a cabo registro a registro.

Otra característica de este tipo de modelos es que su implementación se lleva a cabo por medio de punteros.

Este modelo en red general es muy flexible, debido a la inexistencia de restricciones inherentes; pero también, por esta misma razón, su instrumentación física resulta difícil y poco eficiente. Ésta es la causa de que el modelo en red general que hemos presentado sea teórico, y que al llevado a la práctica se introduzcan restricciones.

2.4. MODELO DE BASES DE DATOS RELACIONAL

2.4.1. Conceptos básicos

El modelo relacional es en la actualidad el más popular de los sistemas de manejo de una base de datos, puesto que es conceptualmente sencillo y comprensible por los profesionales de los sistemas de información y muchos otros usuarios finales; puede evolucionar, ya que las relaciones entre los datos no necesitan estar predefinidas, además utiliza valores de los datos para implicar las relaciones. Las bases de datos relacionales utilizan un modelo para mostrar cómo se relacionan lógicamente los datos de un registro.

El orden de los datos en la tabla no es significativo y tampoco implica un orden cuando los registros están incluidos en la relación. Análogamente, los detalles físicos de almacenamiento (ya sea una organización aleatoria, indexada o secuencial) no son de interés para el analista. Las tablas relacionales muestran las relaciones lógicas, no físicas. Lo atractivo de las bases de datos relacionales es que el modelo relacional se puede comprender rápidamente.

Al hacer una solicitud de información, el sistema produce una tabla que contiene la información. En el ejemplo de las citas médicas, si se desea determinar quién tiene cita con el dermatólogo, el sistema producirá una tabla que contenga los nombres de todos los pacientes de dermatología.

El modelo relacional cumple las siguientes características:

- Independencia física: El modo en que se almacenan los datos no debe influir en su manipulación lógica y, por tanto, los usuarios que acceden a esos datos no han de modificar sus programas por cambios en el almacenamiento físico.

- Independencia lógica: Añadir, eliminar o modificar cualquier elemento de la base de datos no debe repercutir en los programas y/o usuarios que están accediendo a subconjuntos parciales de los mismos (vistas).

- Flexibilidad: En el sentido de poder ofrecer a cada usuario los datos de la forma más adecuada a la correspondiente aplicación.

- Uniformidad: Las estructuras lógicas de los datos presentan un aspecto uniforme (tablas), lo que facilita la concepción y manipulación de la base de datos por parte de los usuarios.

- Sencillez: Las características anteriores, así como unos lenguajes de usuario muy sencillos, producen como resultado que el modelo de datos relacional sea fácil de comprender y de utilizar por parte del usuario final.

2.4.2. Dominios y atributos

Un dominio D es un conjunto finito de valores homogéneos y atómicos V_1, V_2, ..., V_n, caracterizado por un nombre; decimos valores homogéneos porque son todos del mismo tipo, y atómicos porque son indivisibles en lo que al modelo se refiere, es decir, si se descompusiesen, perderían la semántica a ellos asociada. Por ejemplo, el dominio de nacionalidades tiene los valores: Mexicana, Española, Francesa, Norteamericana, etc., que son todos del mismo tipo y no se pueden dividir sin que se pierda su semántica; así, si descompusiéramos el valor "MEXICANA" en las letras "M", "E", "X", etc., se perdería la semántica, ya que estas letras consideradas aisladamente han dejado de tener el significado que tiene "MEXICANA" como un valor de la nacionalidad.

Todo dominio ha de tener un nombre, por el cual nos podemos referir a él, y un tipo de datos; así, el tipo de datos del dominio de nacionalidades es una tira de caracteres de longitud diez. También se le puede asociar una unidad de medida, como metros, kilos, etc., y ciertas restricciones.

Los dominios pueden definirse por intensión o por extensión. Por ejemplo, el dominio de las edades de las personas activas se puede definir por intensión como entero de longitud dos comprendido entre 18 y 65, mientras que la definición del dominio de nacionalidades por intensión sería muy pobre semánticamente, ya que permitiría toda combinación de 10 letras aún cuando no constituyesen un nombre válido de nacionalidad; por ello, sería preferible definir este dominio por extensión con los nombres de las distintas nacionalidades que admitiésemos en nuestra base de datos.

Se podría pensar que un dominio es igual que una relación de grado 1 (con un único atributo). Sin embargo esto no es cierto, ya que el dominio contiene todos los posibles valores que puede tomar un atributo y es estático (estos valores no varían en el transcurso del tiempo), en cambio la relación es dinámica por su misma naturaleza; además, los dominios representan un papel importante y característico en ciertas operaciones.

Un atributo A es el papel que juega un determinado dominio D en una relación; se dice que D es el dominio de A y se denota como dom(A). Así, el atributo Nacionalidad de la tabla AUTOR, definido sobre el dominio de Nacionalidades, nos indica que dicho dominio tiene el papel de nacionalidad del autor en la referida tabla.

El universo del discurso de una base de datos relacional, representado por U, está compuesto por un conjunto finito y no vacío de atributos, A_1, A_2,…,A_n, estructurados en relaciones; cada atributo toma sus valores de un único dominio (dominio subyacente) y varios atributos pueden tener el mismo dominio subyacente.

Es muy usual dar el mismo nombre al atributo y al dominio subyacente. En el caso de que sean varios los atributos de una misma tabla definidos sobre el mismo dominio, habrá que darles nombres distintos, ya que una tabla no puede tener dos atributos con el mismo nombre.

Además de los dominios y atributos simples, se introduce el concepto de dominio compuesto el cual todavía no se encuentra en el modelo relacional.

Un dominio compuesto se puede definir como una combinación de dominios simples a la que se pueden aplicar ciertas restricciones de integridad. Por ejemplo, un

usuario puede necesitar manejar, además de los tres dominios Día, Mes y Año, un dominio compuesto por ellos denominado Fecha, al que podríamos aplicar las adecuadas restricciones de integridad a fin de que no aparecieran valores no válidos para la fecha; algo análogo ocurre con el nombre y los apellidos, que, según las aplicaciones, puede ser conveniente tratarlos en conjunto o por separado.

Al igual que es posible definir dominios compuestos, existen también atributos compuestos; así, el atributo Fecha tomaría sus valores del dominio compuesto de igual nombre.

Tanto los atributos compuestos como los dominios compuestos pueden ser tratados, si así lo precisa el usuario, como "piezas únicas" de información, es decir, como valores atómicos.

2.4.3. Relaciones

Matemáticamente, una relación definida sobre los n dominios D_1, D_2, …, D_n no necesariamente distintos, es un subconjunto del producto cartesiano de estos dominios, donde cada elemento de la relación (tupla) es una serie de n valores ordenados.

En esta definición matemática de relación no se alude a los atributos, es decir, al papel que representan los dominios en la relación y, además, en ella el orden de los valores dentro de una tupla es significativo. A fin de evitar estos inconvenientes se puede dar otra definición de relación más adecuada desde el punto de vista de las bases de datos, para lo cual es preciso distinguir en la noción de relación los siguientes elementos:

- Nombre: Las relaciones se identifican por un nombre; si bien ciertas relaciones que no necesitan identificarse (por ejemplo, resultados intermedios) pueden no tener nombre.

- Cabecera de relación: Conjunto de n pares atributo-dominio subyacente $\{ (A_i: D_j) \}^n_{i=0}$ donde n es el grado; se corresponde con la primera fila cuando la relación se percibe como una tabla. El conjunto A de atributos sobre los que se define la relación se llama contexto de la misma.

- Cuerpo de la relación: Conjunto de m tuplas $\{ t_1, t_2, \ldots, t_m \}$, donde cada tupla es un conjunto de n pares atributo-valor $\{ (A_i: V_{ij}) \}$, siendo V_{ij} el valor j del dominio Di asociado al atributo A_i; el número de tuplas m es la cardinalidad. Así como la cabecera de relación es invariante, su cuerpo varía en el transcurso del tiempo, al igual que la cardinalidad.

- El esquema de relación estará constituido por el nombre R -si existe- y la cabecera, denotándose:

$$\mathbf{R (\{ A_i : D_i \; H=1)}$$

El esquema de relación representa la parte definitoria y estática y se denomina también intención, corresponde con lo que hemos llamado tipo (de entidad) en el modelo Entidad/Interrelación.

El estado de relación r(R), al que denominaremos simplemente relación, está constituido por el esquema y el cuerpo de relación:

<esquema, cuerpo>

Siendo el cuerpo el conjunto de tuplas que, es un instante dado, satisface el correspondiente esquema de relación.

ESQUEMA DE RELACIÓN (INTENSIÓN):

PACIENTE (Nombre: Nombres, Especialidad: Especialidades, Médico: Médicos)

RELACIÓN (EXTENSIÓN, ESTADO U OCURRENCIA):

PACIENTE

Nombre	Especialidad	Médico
Flores, Isaac	Ortopedia	Luna, Laura
Pérez, Francisco	Urología	Sánchez, Pedro
Ruíz, Elena	Neurología	Solís, Teresa

Figura 2.20. *Ejemplo de esquema y relación.*

Una base de datos relacional es una base de datos percibida por los usuarios como una colección de relaciones que varían en el tiempo, es decir, una colección de variables de relación.

2.4.4. Clases de relación

Existen diversas clasificaciones de las relaciones. En primer lugar, dividiremos las relaciones en nominadas y sin nombre.

Las relaciones nominadas, a su vez, pueden ser:

- Persistentes: Son aquellas relaciones cuya definición (esquema de relación) permanece en la base de datos, borrándose solamente mediante una acción explícita del usuario. Las relaciones persistentes se dividen en:
    - Relaciones base (se corresponden con el nivel conceptual de la arquitectura ANSI). Existen por sí mismas, no en función de otras relaciones, y se crean especificando explícitamente su esquema de relación (nombre y conjunto de pares: atributo/dominio). Sus extensiones (ocurrencias de relación), al igual que su definición, también se encuentran almacenadas.
    - Vistas (se corresponden con el nivel externo de la arquitectura ANSI). Son relaciones derivadas que se definen dando un nombre

a una expresión de consulta. Se podría decir que son relaciones virtuales (como ventanas sobre otras relaciones), en el sentido de que no tienen datos almacenados, sino que lo único que se almacena es su definición en términos de otras relaciones con nombre, las cuales pueden ser relaciones base, otras vistas o instantáneas.

- Instantáneas (corresponden con el nivel interno de la arquitectura ANSI). Son relaciones derivadas al igual que las vistas, es decir, se definen en términos de otras relaciones nominadas, pero tienen datos propios almacenados, los cuales son el resultado de ejecutar la consulta especificada o de guardar una relación base. A veces, se les llama vistas materializadas. Las instantáneas no se actualizan cuando cambian los datos de las relaciones sobre las que están definidas, pero se "refrescan" (es decir, se renuevan sus datos), cada cierto tiempo, de acuerdo con lo indicado por el usuario en el momento de su creación. Son, por tanto, sólo de lectura, no pudiendo ser actualizadas por el usuario, sino únicamente "refrescadas" por el sistema.

- Temporales. A diferencia de las relaciones persistentes, una relación temporal desaparece de la base de datos en un cierto momento sin necesidad de una acción de borrado específica del usuario; por ejemplo, al terminar una sesión o una transacción. En general, una relación temporal suele ser una relación autónoma (en el sentido de que no se deriva de otra, al igual que las relaciones base), pero también podría ser una vista o una instantánea.

- Relaciones sin nombre: Las relaciones sin nombre son los resultados de las consultas que no se materializan sino que se entregan al usuario que ha realizado la consulta, y pueden ser tanto resultados intermedios como finales; en consecuencia, las relaciones no nominadas son siempre temporales.

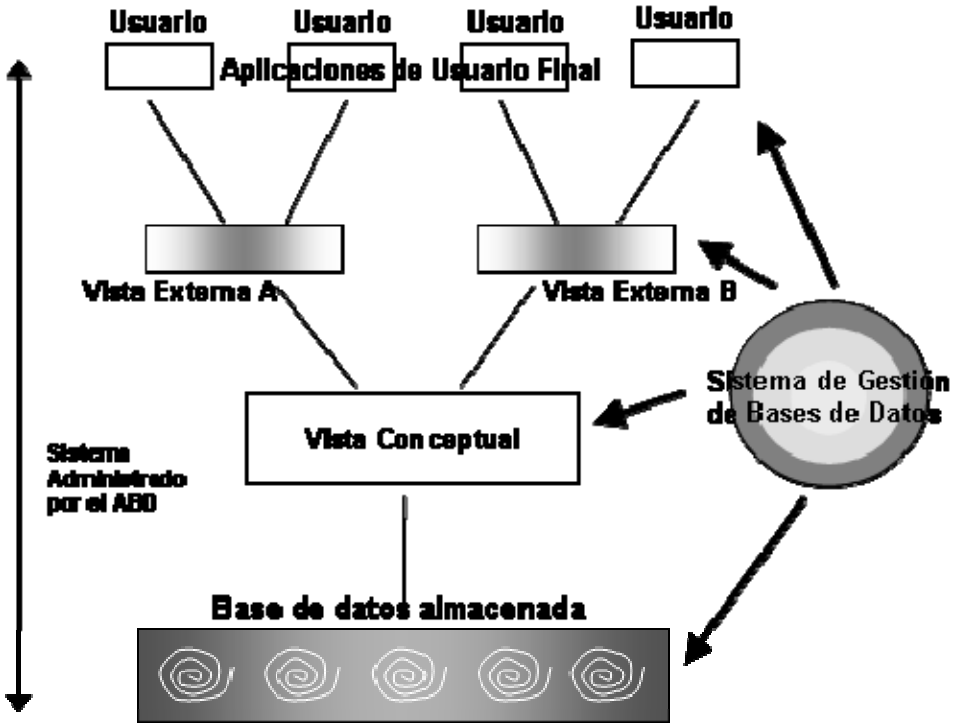


Figura 2.21. *Ejemplo de relaciones "Vistas".*

Tampoco existe, respecto a la clasificación de las relaciones, una terminología unánimemente aceptada. Así, por ejemplo se emplean relaciones "de expresión" (expressible), que son todas aquellas relaciones (sean o no nominadas) que se derivan de otras, más las relaciones almacenadas, que son las que tienen una representación directa en el almacenamiento físico (para nosotros, estas últimas no son realmente relaciones, y las llamaremos datos almacenados). El SQL92 tampoco se ajusta a la clasificación que acabamos de presentar; en ISO (1992) las tablas se dividen en persistentes y temporales. Las tablas persistentes, como su nombre indica, se almacenan en memoria secundaria y permanecen allí cuando termina la sesión en la que fueran creadas, mientras que las temporales sólo se materializan y tienen existencia durante un período de tiempo específico (una sesión o una transacción); las tablas temporales se dividen a su vez en globales, creadas localmente y declaradas localmente. El SQL92 no contempla las instantáneas. En la figura se puede ver un resumen de la clasificación anterior, donde se han señalado con un asterisco las relaciones derivadas.

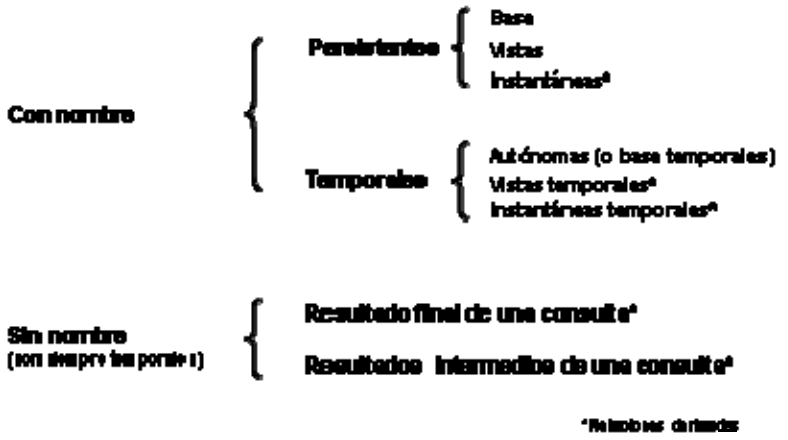


Figura 2.22. *Clasificación de las relaciones.*

2.4.5. Atributos llave (clave)

Una clave candidata de una relación es un conjunto de atributos que identifican unívoca y mínimamente cada tupla de la relación. Por la propia definición de relación, siempre hay, al menos, una clave candidata, ya que al ser una relación un conjunto no existen dos tuplas iguales y, por tanto, el conjunto de todos los atributos siempre tiene que identificar unívocamente a cada tupla; si no se cumpliera la condición de minimalidad se eliminarían aquellos atributos que lo impidiesen.

Una relación puede tener más de una clave candidata, entre las cuales se debe distinguir:

- Clave primaria: es aquella clave candidata que el usuario escogerá, por consideraciones ajenas al modelo relacional, para identificar las tuplas de la relación. Cuando sólo existe una clave candidata, ésta será la clave primaria.

- Claves alternativas: son aquellas claves candidatas que no han sido escogidas como clave primaria.

Se denomina clave ajena de una relación R2 a un conjunto no vacío de atributos cuyos valores han de coincidir con los valores de la clave candidata[18] de una relación R1 (R1 y R2 no son necesariamente distintas). Cabe destacar que la clave ajena y la correspondiente clave candidata han de estar definidas sobre el mismo dominio.

Los conceptos de claves, tanto candidatas (en especial primaria) como ajenas, son muy importantes en el estudio de la integridad del modelo relacional.

2.4.6. Representación del modelo mediante diagramas

Una base de datos relacional consiste en un conjunto de tablas, que tienen asignado un nombre único. Las tablas tienen una estructura similar a las bases de datos E-R. Una columna de una tabla representa una relación entre un conjunto de valores. Puesto que una tabla es un conjunto de estas relaciones, existe una

correspondencia entre el concepto de tabla y el concepto matemático de relación, del cual recibe su nombre el modelo de datos relacional.

La relación es el elemento básico del modelo relacional, y se puede representar como una tabla, ver la figura 2.23.

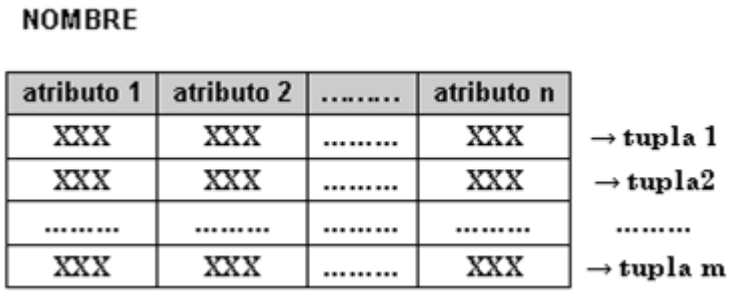


Figura 2.23. *Representación de una relación en forma de tabla.*

En ella podemos distinguir su nombre, un conjunto de columnas, denominadas atributos, que representan propiedades de la tabla y que también están caracterizadas por su nombre, y un conjunto de filas llamadas tuplas, que contienen los valores que toma cada uno de los atributos para cada elemento de la relación. En la figura representación de la relación paciente, se representa la relación PACIENTE, en donde aparece la estructura del modelo relacional. En ella podemos observar el nombre de la relación (PACIENTE); los atributos (Nombre, Especialidad y Médico); los dominios (de donde los atributos toman sus valores; varios atributos pueden tomar valores del mismo dominio); las tuplas (cada una de las cuales contiene los valores que toma el nombre, la especialidad y el médico para un determinado paciente); el grado (número de atributos); y la cardinalidad (número de tuplas).

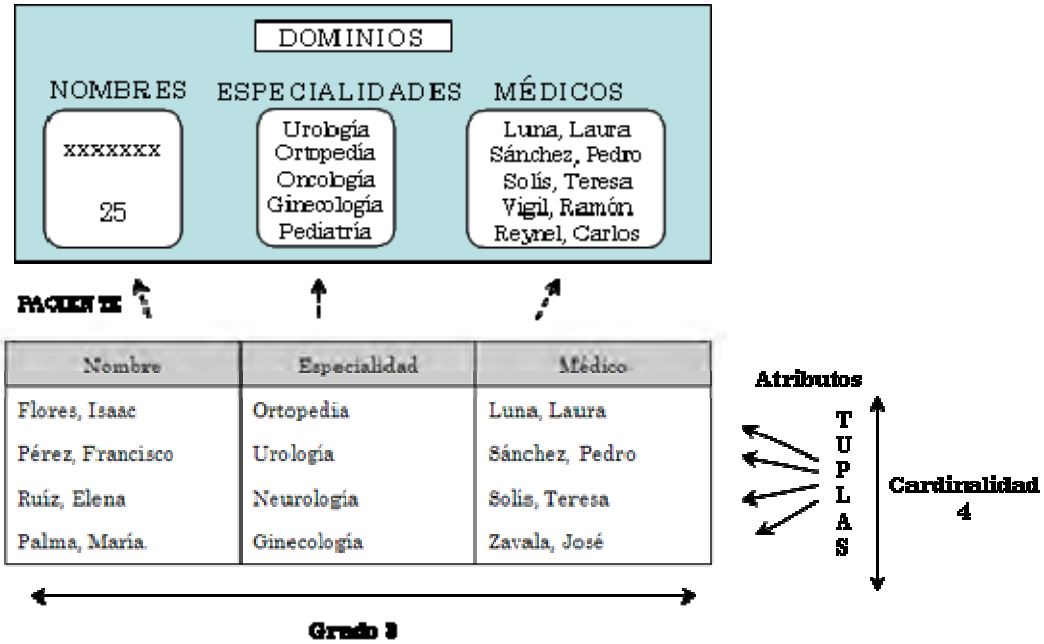


Figura 2.24. *Representación de la relación paciente.*

Insistimos en que una relación se puede representar en forma de tabla, aunque tiene una serie de elementos característicos que la distinguen de la tabla, ya que no se admiten filas duplicadas, las filas y las columnas no están ordenadas y es plana, es decir, que en el cruce de una fila y de una columna sólo puede haber un valor (no se admiten atributos multivaluados). Se trata de restricciones inherentes al modelo.

En una tabla se puede distinguir una cabecera que define la estructura de la tabla. Es decir, sus atributos con los dominios subyacentes, y un cuerpo que está formado por un conjunto de tuplas que varían en el tiempo.

Esta representación de la relación como una tabla ha sido el origen de que los productos relacionales y los usuarios utilicen habitualmente el nombre de tabla (en principio ajeno a la teoría relacional) para denominar las relaciones y, como consecuencia de ello, se llame filas a las tuplas y columnas a los atributos; si bien, la terminología es irrelevante y un producto más o menos relacional una u otra terminología.

En la figura 2.25 se compara la terminología relacional con la que corresponde a las tablas y a los archivos.

Figura 2.25. *Comparación de la terminología de relación, tabla y archivo.*

2.4.7. Álgebra Relacional

El álgebra relacional es un lenguaje de consulta procedimental. Consta de un conjunto de operaciones que toman como entrada una o dos relaciones y producen como resultado una nueva relación.

El álgebra relacional es similar al álgebra aritmética pero con una diferencia importante, el álgebra aritmética utiliza variables que representan números y operadores que actúan sobre elementos numéricos. En álgebra relacional, sin embargo, las variables son tablas y los operadores actúan sobre estas dando como resultado nuevas tablas. De hecho el álgebra relacional es cerrada, lo que significa que los resultados de la aplicación de una o más operaciones siempre producen tablas. El álgebra relacional es muy importante por varias razones:

1. Proporciona un fundamento formal para las operaciones del modelo relacional.

2. Quizá la más importante, es que se utiliza como base para la implementación y optimización de consultas en los SGBD.

3. Algunos de sus conceptos se han incorporado al lenguaje estándar de consultas SQL.

4. Describe el aspecto de la manipulación de datos del modelo relacional.

5. Se usa como una representación intermedia de una consulta a una base de datos y sirve para obtener una versión más optimizada y eficiente de dicha consulta.

Las operaciones fundamentales del álgebra relacional son selección, proyección, unión, diferencia de conjuntos, producto cartesiano y renombramiento. Además de las operaciones fundamentales hay otras operaciones, por ejemplo, intersección de conjuntos, reunión natural, división y asignación. Estas operaciones se definirán en términos de las operaciones fundamentales.

2.4.7.1. Operaciones fundamentales

Las operaciones selección, proyección y renombramiento se denominan operaciones unarias porque operan sobre una sola relación. Las otras tres operaciones operan sobre pares de relaciones y se denominan, por lo tanto, operaciones binarias.

2.4.7.2. La operación selección

La operación selección selecciona tuplas que satisfacen un predicado dado. Se utiliza la letra griega sigma minúscula (σ) para denotar la selección. El predicado aparece como subíndice de σ. La relación del argumento se da entre paréntesis a continuación de σ.

En general, se permiten las comparaciones que utilizan =, ≠, <, ≤, > o en el predicado de selección. Además, se pueden combinar varios predicados en uno mayor utilizando las conectivas y ($\wedge$) y/o ($\vee$). El predicado de selección puede incluir comparaciones entre dos atributos. Dado que el valor especial nulo indica «valor desconocido o inexistente», cualquier comparación que implique a un valor nulo se evalúa como falsa.

En la figura 2.26 aparece la relación resultante de aplicar el operador de selección sobre el atributo “especialidad” a la tabla Médicos.

Médicos

Nombre	Especialidad	Edad
Juan Pérez	Urología	35
Pedro Méndez	Alergología	40
Rosa Santos	Neurología	57
Francisco Ortiz	Medicina General	44
Luis Robles	Urología	51
Sofía González	Nefrología	39

$\sigma_{especialidad="urología"}$ (Médicos)

Nombre	Especialidad	Edad
Juan Pérez	Urología	35
Luis Robles	Urología	51

Figura 2.26. *Ejemplo de operación de restricción.*

2.4.7.3. La operación proyección

La operación proyección es una operación unaria que devuelve su relación de argumentos, excluyendo algunos argumentos. Dado que las relaciones son conjuntos, se eliminan todas las filas duplicadas. La proyección se denota por la letra griega mayúscula pi (Π). Se crea una lista de los atributos que se desea que aparezcan en el resultado como subíndice de Π. La relación de argumentos se escribe a continuación entre paréntesis.

La figura siguiente muestra un ejemplo de la operación proyección al aplicar este operador sobre el atributo “especialidad” a la tabla Médicos, eliminando las tuplas duplicadas que hubieran podido resultar.

Médicos

Nombre	Especialidad	Edad
Juan Pérez	Urología	35
Pedro Méndez	Alergología	40
Rosa Santos	Neurología	57
Francisco Ortiz	Medicina General	44
Luis Robles	Urología	51
Sofía González	Nefrología	39

$\Pi_{especialidad}$(Médicos)

Especialidad
Urología
Alergología
Neurología
Medicina General
Nefrología

Figura 2.27. *Ejemplo de operación de proyección.*

2.4.7.4. La operación unión

La operación unión se denota por $\cup$. Para que una operación unión r $\cup$ s sea válida hay que exigir que se cumplan dos condiciones:

1. Las relaciones r y s deben tener el mismo número de atributos.

2. Los dominios de los atributos i-ésimos de r y de s deben ser iguales para todo i.

Téngase en cuenta que r y s pueden ser, en general, relaciones temporales que sean resultado de expresiones del álgebra relacional.

En la figura 2.28 se encuentra representado un ejemplo de unión entre las relaciones Médicos y Médicos 1.

Médicos

Nombre	Especialidad	Edad
Juan Pérez	Urología	35
Rosa Santos	Neurología	57
María Fernández	Medicina Preventiva	32

Médicos 1

Nombre	Especialidad	Edad
Pedro Méndez	Alergología	40
María Fernández	Medicina Preventiva	32
Luis Robles	Urología	51
Sofía González	Nefrología	39

Médicos **U** Médicos 1

Nombre	Especialidad	Edad
Juan Pérez	Urología	35
Rosa Santos	Neurología	57
María Fernández	Medicina Preventiva	32
Pedro Méndez	Alergología	40
Luis Robles	Urología	51
Sofía González	Nefrología	39

Figura 2.28. *Ejemplo de operación unión.*

2.4.7.5. La operación diferencia de conjuntos

La operación diferencia de conjuntos, denotada por -, permite buscar las tuplas que estén en una relación pero no en la otra. La expresión r - s da como resultado una relación que contiene las tuplas que están en r pero no en s.

Como en el caso de la operación unión, hay que asegurarse de que las diferencias de conjuntos se realicen entre relaciones compatibles. Por tanto, para que una operación diferencia de conjuntos r - s sea válida hay que exigir que las relaciones r y

s sean de la misma aridad (deben tener el mismo número de atributos) y que los dominios de los atributos i-ésimos de r y s sean iguales.

La figura 2.29 muestra un ejemplo de diferencia entre las relaciones Médicos y Médicos 1.

Médicos

Nombre	Especialidad	Edad
Juan Pérez	Urología	35
Rosa Santos	Neurología	57
María Fernández	Medicina Preventiva	32

Médicos 1

Nombre	Especialidad	Edad
Pedro Méndez	Alergología	40
María Fernández	Medicina Preventiva	32
Luis Robles	Urología	51
Sofía González	Nefrología	39

Médicos — Médicos 1

Nombre	Especialidad	Edad
Juan Pérez	Urología	35
Rosa Santos	Neurología	57

Figura 2.29. *Ejemplo de operación diferencia de conjuntos.*

2.4.7.6. La operación producto cartesiano

La operación producto cartesiano, denotada por un aspa (x), permite combinar información de cualesquiera dos relaciones. El producto cartesiano de las relaciones r_1 y r_2 como r_1 x r_2.

Recuérdese que las relaciones se definen como subconjuntos del producto cartesiano de un conjunto de dominios. A partir de esta definición ya se debe tener una intuición sobre la definición de la operación producto cartesiano. Sin embargo, dado que el mismo nombre de atributo puede aparecer tanto en r_1 como en r_2, hay que crear un esquema de denominaciones para distinguir entre ambos atributos. En este caso se logra adjuntando al atributo el nombre de la relación de la que proviene originalmente.

El acuerdo de denominaciones precedente exige que las relaciones que sean argumentos de la operación producto cartesiano tengan nombres diferentes. Esta exigencia causa problemas en algunos casos, como cuando se desea calcular el producto cartesiano de una relación consigo misma. Se produce un problema similar si se utiliza el resultado de una expresión del álgebra relacional en un producto cartesiano, dado que hará falta un nombre para la relación para poder hacer referencia a sus atributos.

En general, si se tienen las relaciones r_1 (R_1) y r_2 (R_2), r_1 x r_2 es una relación cuyo esquema es la concatenación de R_1 y de R_2. La relación R contiene todas las tuplas t para las que hay unas tuplas t_1 en r_1 y t_2 en r_2 para las que $t[R_1] = t_1[R_1]$ y $t[R_2] = t_2[R_2]$.

En la siguiente figura aparece un ejemplo de producto cartesiano entre las relaciones Médicos y Pacientes.

Médicos

Nombre	Especialidad	Edad
Juan Pérez	Urología	35
Rosa Santos	Neurología	57
María Fernández	Medicina Preventiva	32

Pacientes

Código de Paciente	Nombre	Sexo
82	Carlos Soto	M
121	Ramón Cabello	M
3	Fernanda Estudillo	F

Médicos X Pacientes

Nombre	Especialidad	Edad	Código de Paciente	Nombre	Sexo
Juan Pérez	Urología	35	82	Carlos Soto	M
Juan Pérez	Urología	35	121	Ramón Cabello	M
Juan Pérez	Urología	35	3	Fernanda Estudillo	F
Rosa Santos	Neurología	57	82	Carlos Soto	M
Rosa Santos	Neurología	57	121	Ramón Cabello	M
Rosa Santos	Neurología	57	3	Fernanda Estudillo	F
María Fernández	Medicina Preventiva	32	82	Carlos Soto90	M
María Fernández	Medicina Preventiva	32	121	Ramón Cabello	M
María Fernández	Medicina Preventiva	32	3	Fernanda Estudillo	F

Figura 2.30. *Ejemplo de operación producto cartesiano.*

2.4.7.7. La operación renombramiento

A diferencia de las relaciones de la base de datos, los resultados de las expresiones de álgebra relacional no tienen un nombre que se pueda utilizar para referirse a ellas. Resulta útil poder ponerles nombre; el operador renombramiento, denotado por la letra griega rho minúscula (ρ), permite realizar esta tarea. Dada una expresión E del álgebra relacional, la expresión

$$\rho_x(E)$$

devuelve el resultado de la expresión E con el nombre x.

Las relaciones r por sí mismas se consideran expresiones (triviales) del álgebra relacional. Por tanto, también se puede aplicar la operación renombramiento a una relación r para obtener la misma relación con un nombre nuevo.

Otra forma de la operación renombramiento es la siguiente. Supóngase que una expresión del álgebra relacional E tiene aridad n. Por tanto, la expresión

$$\rho_x(A_1, A_2, ..., A_n)(E)$$

devuelve el resultado de la expresión E con el nombre x y con los atributos con el nombre cambiado a $A_1, A_2, ..., A_n$

La operación renombramiento no es estrictamente necesaria, dado que es posible utilizar una notación posicional para los atributos. Se pueden nombrar los atributos de una relación de manera implícita utilizando una notación posicional, donde $1, $2, ... hagan referencia al primer atributo, al segundo, etcétera. La notación posicional también se aplica a los resultados de las operaciones del álgebra relacional. La siguiente expresión del álgebra relacional ilustra el uso de la notación posicional con el operador unario σ:

$$\sigma_{\$2=\$3}(RxR)$$

Si una operación binaria necesita distinguir entre las dos relaciones que son sus operandos, se puede utilizar una notación posicional parecida para los nombres de las relaciones. Por ejemplo, $R1 puede hacer referencia al primer operando y $R2, al segundo. Sin embargo, la notación posicional no resulta conveniente para las personas, dado que la posición del atributo es un número en vez de un nombre de atributo fácil de recordar.

2.4.7.8. Definición formal del álgebra relacional

Las operaciones que se vieron anteriormente permiten dar una definición completa de las expresiones del álgebra relacional. Las expresiones fundamentales del álgebra relacional se componen de alguna de las siguientes:

- Una relación de la base de datos
- Una relación constante

Una relación constante se escribe listando sus tuplas entre llaves ({}).

Las expresiones generales del álgebra relacional se construyen a partir de subexpresiones menores. Sean E_1 y E_2 expresiones de álgebra relacional. Todas las siguientes son expresiones del álgebra relacional:

- $E_1 \cup E_2$
- $E_1 - E_2$.
- $E_1 \times E_2$
- $\sigma_\rho(E_1)$, donde P es un predicado de atributos de E_1

- $\Pi_s(E_1)$, donde S es una lista que se compone de algunos de los atributos de E_1
- $\rho_x(E_1)$, donde x es el nuevo nombre del resultado de E_1

2.4.7.9. Otras operaciones

Las operaciones fundamentales del álgebra relacional son suficientes para expresar cualquier consulta del álgebra relacional. Sin embargo, si uno se limita únicamente a las operaciones fundamentales, algunas consultas habituales resultan de expresión intrincada. Por tanto, se definen otras operaciones que no añaden potencia al álgebra, pero que simplifican las consultas habituales. Para cada operación nueva se facilita una expresión equivalente utilizando sólo las operaciones fundamentales.

2.4.7.9.1. La operación intersección de conjuntos

La primera operación adicional del álgebra relacional que se definirá es la intersección de conjuntos ($\cap$).

Se puede volver a escribir cualquier expresión del álgebra relacional utilizando la intersección de conjuntos sustituyendo la operación intersección por un par de operaciones de diferencia de conjuntos, de la manera siguiente:

$$r \cap s = r - (r - s)$$

Por tanto, la intersección de conjuntos no es una operación fundamental y no añade potencia al álgebra relacional. Sencillamente, es más conveniente escribir $r \cap s$ que $r - (r - s)$.

En la figura 2.31 se encuentra representado un ejemplo de intersección entre las relaciones Médicos y Médicos 1.

Médicos

Nombre	Especialidad	Edad
Juan Pérez	Urología	35
Rosa Santos	Neurología	57
María Fernández	Medicina Preventiva	32

Médicos 1

Nombre	Especialidad	Edad
Pedro Méndez	Alergología	40
María Fernández	Medicina Preventiva	32
Luis Robles	Urología	51
Sofía González	Nefrología	39

Médicos ∩ Médicos 1

Nombre	Especialidad	Edad
María Fernández	Medicina Preventiva	32

Figura 2.31. *Ejemplo de operación intersección de conjuntos.*

2.4.7.9.2. La operación reunión natural

Suele resultar deseable simplificar ciertas consultas que exigen un producto cartesiano. Generalmente, las consultas que implican un producto cartesiano incluyen un operador selección sobre el resultado del producto cartesiano.

La reunión natural es una operación binaria que permite combinar ciertas selecciones y un producto cartesiano en una sola operación. Se denota por el símbolo de la reunión $\bowtie$. La operación reunión natural forma un producto cartesiano de sus dos argumentos, realiza una selección forzando la igualdad de los atributos que aparecen en ambos esquemas de relación y, finalmente, elimina los atributos duplicados.

Considérense dos esquemas de relación R y S que son, por supuesto, listas de nombres de atributos. Si se consideran los esquemas como conjuntos, en vez de como listas, se pueden denotar los nombres de los atributos que aparecen tanto en R como en S mediante $R \cap S$, y los nombres de los atributos que aparecen en R, en S o en ambos mediante $R \cup S$. De manera parecida, los nombres de los atributos que aparecen en R pero no en S se denotan por R - S, mientras que S - R denota los nombres de los atributos que aparecen en S pero no en R. Obsérvese que las operaciones unión, intersección y diferencia aquí operan sobre conjuntos de atributos, y no sobre relaciones.

Ahora se está preparado para una definición formal de la reunión natural. Considérense dos relaciones r(R) y s(S). La reunión natural de r y de s, denotada por $r \bowtie s$ es una relación del esquema $R \cup S$ definida formalmente de la manera siguiente:

$$r \bowtie s = \Pi_{R \cup S}(\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n} r \times s)$$

donde $R \cap S = \{A_1, A_2, ..., A_n\}$.

La operación reunión zeta es una extensión de la operación reunión natural que permite combinar una selección y un producto cartesiano en una sola operación. Considérense las relaciones r(R) y s(S), y sea θ un predicado de los atributos del esquema R $\cup$ S. La operación reunión zeta r $\bowtie_\theta$ s se define de la manera siguiente:

$$r \bowtie_\theta s = \sigma_\theta (r \times s)$$

La figura 2.32 representa un ejemplo de reunión natural, donde se han eliminado en la relación resultante uno de los atributos idénticos.

Médicos

Nombre	Especialidad	Edad
Juan Pérez	Urología	35
Rosa Santos	Neurología	57
María Fernández	Medicina Preventiva	32
Sofía González	Nefrología	39

Citas

No. De Cita	Especialidad	Fecha
9	Medicina Preventiva	5/10/2008
22	Urología	30/09/2008
85	Neurología	13/11/2008
42	Ginecología	01/10/2008

Médicos ⋈ Citas

Nombre	Especialidad	Edad	No. De Cita	Fecha
Juan Pérez	Urología	35	22	30/09/2008
Rosa Santos	Neurología	57	85	13/11/2008
María Fernández	Medicina Preventiva	32	9	5/10/2008

Figura 2.32. *Ejemplo de operación reunión natural.*

2.4.7.9.3. La operación división

La operación división, denotada por $\div$, resulta adecuada para las consultas que incluyen la expresión para todos.

Formalmente, sean r(R) y s(S) relaciones y S $\subseteq$ R; es decir, todos los atributos del esquema S están también en el esquema R. La relación r $\div$ s es una relación del esquema R - S (es decir, del esquema que contiene todos los atributos del esquema R que no están en el esquema S). Una tupla t está en r $\div$ s si y sólo si se cumplen estas dos condiciones:

1. t está en $\Pi_{R-S}(r)$

2. Para cada tupla t_s de s hay una tupla t_r de r que cumple las dos condiciones siguientes:

   a) $t_r[S] = t_s[S]$

   b) $t_r[R-S] = t$

Puede resultar sorprendente descubrir que, dados una operación división y los esquemas de las relaciones, se puede, de hecho, definir la operación división en términos de las operaciones fundamentales. Sean r(R) y s(S) dadas, con $S \subseteq R$:

$$r \div s = \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

Para comprobar que esta expresión es verdadera, obsérvese que $\Pi_{R-S}(r)$ da todas las tuplas t que cumplen la primera condición de la definición de la división. La expresión del lado derecho del operador diferencia de conjuntos,

$$\Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

sirve para borrar esas tuplas que no cumplen la segunda condición de la definición de la división. Esto se logra de la manera siguiente. Considérese $\Pi_{R-S}(r) \times s$. Esta relación está en el esquema R y empareja cada tupla de $\Pi_{R-S}(r)$ con cada tupla de s. La expresión $\Pi_{R-S,S}(r)$ sólo reordena los atributos de r.

Por tanto, $\Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$ genera los pares de tuplas de $\Pi_{R-S}(r)$ y de s que no aparecen en r. Si una tupla t_j está en

$$\Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

hay alguna tupla t_s de s que no se combina con la tupla t_j para formar una tupla de r. Por tanto, t_j guarda un valor de los atributos R - S que no aparece en $r \div s$. Estos valores son los que se eliminan de $\Pi_{R-S}(r)$.

En el ejemplo siguiente, se desea obtener a los pacientes cuyas citas son en la especialidad “Medicina General” y “Alergología”.

Pacientes

Código de Paciente	Nombre	Cita en Especialidad
82	Carlos Soto	Medicina Preventiva
121	Ramón Cabello	Urología
3	Fernanda Estudillo	Alergología
5	Julio García	Neurología
94	Esperanza Martínez	Alergología
21	José Pardo	Neurología
87	Tomás Torres	Medicina General
94	Esperanza Martínez	Medicina General
106	Teresa Rosales	Ginecología
3	Fernanda Estudillo	Medicina General

Especialidad

Especialidad
Alergología
Medicina General

Pacientes ÷ Especialidad

Código de Paciente	Nombre
3	Fernanda Estudillo
94	Esperanza Martínez

Figura 2.33. *Ejemplo de operación división.*

2.4.7.9.4. La operación asignación

En ocasiones resulta conveniente escribir una expresión del álgebra relacional por partes utilizando la asignación a una variable de relación temporal. La operación asignación, denotada por ←, actúa de manera parecida a la asignación de los lenguajes de programación. Para ilustrar esta operación, considérese la definición de la división. Se puede escribir r -;- s como

$$temp1 \leftarrow \Pi_{R-S}(r)$$

$$temp2 \leftarrow \Pi_{R-S}(r)((temp1 \times s) - \Pi_{R-S,S}(r)$$

$$resultado = temp1 - temp2$$

La evaluación de una asignación no hace que se muestre ninguna relación al usuario. Por el contrario, el resultado de la expresión a la derecha de ← se asigna a la variable relación a la izquierda de ←. Esta variable relación puede utilizarse en expresiones posteriores.

Con la operación asignación se pueden escribir las consultas como programas secuenciales consistentes en una serie de asignaciones seguida de una expresión cuyo valor se muestra como resultado de la consulta. En las consultas del álgebra relacional la asignación siempre debe hacerse a una variable de relación intermedia. Las asignaciones a relaciones permanentes constituyen una modificación de la base de datos. Obsérvese que la operación asignación no añade potencia alguna al

álgebra. Resulta, sin embargo, una manera conveniente de expresar las consultas complejas.

2.4.8. Resumen

El modelo de datos relaciona está basado en una serie de tablas. El usuario del sistema de base de datos puede consultar estas tablas, insertar tuplas nuevas, eliminarlas y actualizarlas (modificadas). Existen varios lenguajes para expresar estas operaciones, como el cálculo relacional de tuplas y el cálculo relacional de dominios los cuales son lenguajes sin procedimientos que representan la capacidad básica requerida en un lenguaje de consulta relacional. El álgebra relacional es un lenguaje de procedimentos que define las operaciones básicas empleadas en los lenguajes de consulta relacionales.

El álgebra relacional es un lenguaje conciso y formal que no es apropiado para usuarios casuales del sistema de base de datos. Es por esto que los sistemas comerciales de base de datos han utilizado lenguajes con más "azúcar sintáctica". Estos lenguajes incluyen construcciones para la actualización, inserción y eliminación de información además para consultar la base de datos.

CAPÍTULO 3: MANEJADORES Y SISTEMAS DE GESTIÓN DE BASES DE DATOS

3.1. CONCEPTOS, UTILIDAD

Los sistemas de bases de datos están diseñados para gestionar grandes volúmenes de información. Generalmente, las bases de datos requieren gran cantidad de espacio de almacenamiento, por lo que las bases de datos de las organizaciones se miden en términos de gigabytes o terabytes de datos. Un gigabyte equivale a 1000 megabytes (un billón de bytes), y un terabyte equivale a un millón de megabytes (un trillón de bytes). Un sistema de bases de datos tiene como objetivo simplificar y facilitar el acceso a los datos y hacer que los tiempos de respuesta a las solicitudes de los usuarios sean muy reducidos.

De forma sencilla, un sistema de gestión de bases de datos se puede definir como una colección de datos interrelacionados y un conjunto de programas para acceder a esos datos. También se define como conjunto coordinado de programas, procedimientos, lenguajes, etc. que suministra, tanto a los usuarios no informáticos como a los analistas, programadores o al administrador, los medios necesarios para describir, recuperar y manipular los datos almacenados en la base, manteniendo su integridad, confidencialidad y seguridad.

Existen dos grandes modelos de sistemas de gestión de bases de datos:

- **Sistemas de Gestión de Bases de Datos Relacionales (SGBDR):** Las bases de datos que generan se construyen con información muy estructurada (datos) acerca de una organización o empresa determinada. Cuando un usuario realiza una consulta en una base de datos relacional, el sistema presenta como resultado la respuesta exacta a lo que se busca. A este tipo de bases de datos se les denomina bases de datos relacionales, y a los sistemas que las gestionan, Sistemas de Gestión de Bases de Datos Relacionales (SGBDR).

  Ejemplos de bases de datos relacionales son las bases de datos de cuentas y clientes de un banco o las bases de datos de productos creadas por los centros comerciales o las librerías para llevar un control de sus ventas. Access es un ejemplo de sistemas de gestión de bases de datos relacionales.

- **Sistemas de Gestión de Bases de Datos Documentales (SGBDD) o Sistemas de Recuperación de Información (SRI):** Las bases de datos que generan se construyen con información no estructurada tipo texto (documentos) sobre uno o varios temas. Cuando un usuario realiza una consulta en una base de datos documental, el sistema presenta como resultado, no una respuesta exacta, sino documentos útiles para satisfacer la pregunta del usuario. A este tipo de bases de datos se les denomina bases de datos documentales, y a los sistemas que las gestionan, Sistemas de Gestión de Bases de Datos Documentales (SGBDD) o Sistemas de Recuperación de Información (SRI).

Ejemplos de bases de datos documentales son las bases de datos bibliográficas, bases de datos de prensa, bases de datos de informes de una empresa, etc. Ejemplos de sistemas de gestión de bases de datos documentales son Knosys, Inmagic, ISIS, BRS, entre otros.

Pese a que las bases de datos relacionales y los sistemas que las gestionan (SGBDR) son los más utilizados y por tanto los más populares, las bases de datos documentales han experimentado un fuerte auge durante estos dos últimos años, impulsado sobre todo por la popularización de Internet y la consiguiente saturación de información textual que ha traído la World Wide Web, así como por el reciente interés de las grandes empresas por gestionar el conocimiento almacenado en documentos.

La siguiente lista contiene los ocho servicios que debe ofrecer todo SGBD.

1. Un SGBD debe proporcionar a los usuarios la capacidad de almacenar datos en la base de datos, acceder a ellos y actualizarlos. Ésta es la función fundamental de un SGBD y por supuesto, el SGBD debe ocultar al usuario la estructura física interna (la organización de los archivos y las estructuras de almacenamiento).

2. Un SGBD debe proporcionar un mecanismo que garantice que todas las actualizaciones correspondientes a una determinada transacción se realicen, o que no se realice ninguna. Una transacción es un conjunto de acciones que cambian el contenido de la base de datos. Una transacción en el sistema informático de la empresa inmobiliaria sería dar de alta a un empleado o eliminar un inmueble. Una transacción un poco más complicada sería eliminar un empleado y reasignar sus inmuebles a otro empleado. En este caso hay que realizar varios cambios sobre la base de datos. Si la transacción falla durante su realización, por ejemplo porque falla el hardware, la base de datos quedará en un estado inconsistente. Algunos de los cambios se habrán hecho y otros no, por lo tanto, los cambios realizados deberán ser deshechos para devolver la base de datos a un estado consistente.

3. Un SGBD debe proporcionar un mecanismo que asegure que la base de datos se actualice correctamente cuando varios usuarios la están actualizando concurrentemente. Uno de los principales objetivos de los SGBD es el permitir que varios usuarios tengan acceso concurrente a los datos que comparten. El acceso concurrente es relativamente fácil de gestionar si todos los usuarios se dedican a leer datos, ya que no pueden interferir unos con otros. Sin embargo, cuando dos o más usuarios están accediendo a la base de datos y al menos uno de ellos está actualizando datos, pueden interferir de modo que se produzcan inconsistencias en la base de datos. El SGBD se debe encargar de que estas interferencias no se produzcan en el acceso simultáneo.

4. Un SGBD debe proporcionar un mecanismo capaz de recuperar la base de datos en caso de que ocurra algún suceso que la dañe. Como se ha comentado antes, cuando el sistema falla en medio de una transacción, la base de datos se debe devolver a un estado consistente. Este fallo puede ser a causa de un fallo en algún dispositivo hardware o un error del software, que hagan que el SGBD se interrumpa, o puede ser a causa de que el usuario detecte un error durante la transacción y la interrumpa antes de que finalice. En todos estos casos, el SGBD debe

proporcionar un mecanismo capaz de recuperar la base de datos llevándola a un estado consistente.

5. Un SGBD debe proporcionar un mecanismo que garantice que sólo los usuarios autorizados pueden acceder a la base de datos. La protección debe ser contra accesos no autorizados, tanto intencionados como accidentales.

6. Un SGBD debe ser capaz de integrarse con algún software de comunicación. Muchos usuarios acceden a la base de datos desde terminales. En ocasiones estos terminales se encuentran conectados directamente a la máquina sobre la que funciona el SGBD. En otras ocasiones los terminales están en lugares remotos, por lo que la comunicación con la máquina que alberga al SGBD se debe hacer a través de una red. En cualquiera de los dos casos, el SGBD recibe peticiones en forma de mensajes y responde de modo similar. Todas estas transmisiones de mensajes las maneja el gestor de comunicaciones de datos. Aunque este gestor no forma parte del SGBD, es necesario que el SGBD se pueda integrar con él para que el sistema sea comercialmente viable.

7. Un SGBD debe proporcionar los medios necesarios para garantizar que tanto los datos de la base de datos, como los cambios que se realizan sobre estos datos, sigan ciertas reglas. La integridad de la base de datos requiere la validez y consistencia de los datos almacenados. Se puede considerar como otro modo de proteger la base de datos, pero además de tener que ver con la seguridad, tiene otras implicaciones. La integridad se ocupa de la calidad de los datos. Normalmente se expresa mediante restricciones, que son una serie de reglas que la base de datos no puede violar. Por ejemplo, se puede establecer la restricción de que cada empleado no puede tener asignados más de diez inmuebles. En este caso sería deseable que el SGBD controlara que no se sobrepase este límite cada vez que se asigne un inmueble a un empleado.

Además, de estos ocho servicios, es razonable esperar que los SGBD proporcionen un par de servicios más:

1. Un SGBD debe permitir que se mantenga la independencia entre los programas y la estructura de la base de datos. La independencia de datos se alcanza mediante las vistas o subesquemas. La independencia de datos física es más fácil de alcanzar, de hecho hay varios tipos de cambios que se pueden realizar sobre la estructura física de la base de datos sin afectar a las vistas. Sin embargo, lograr una completa independencia de datos lógica es más difícil. Añadir una nueva entidad, un atributo o una relación puede ser sencillo, pero no es tan sencillo eliminarlos.

2. Un SGBD debe proporcionar una serie de herramientas que permitan administrar la base de datos de modo efectivo. Algunas herramientas trabajan a nivel externo, por lo que habrán sido producidas por el administrador de la base de datos. Las herramientas que trabajan a nivel interno deben ser proporcionadas por el distribuidor del SGBD. Algunas de ellas son:

- Herramientas para importar y exportar datos.
- Herramientas para monitorizar el uso y el funcionamiento de la base de datos.
- Programas de análisis estadístico para examinar las prestaciones o las estadísticas de utilización.
- Herramientas para reorganización de índices.
- Herramientas para aprovechar el espacio dejado en el almacenamiento físico por los registros borrados y que consoliden el espacio liberado para reutilizarlo cuando sea necesario.

El SGBD incorpora como herramienta fundamental dos lenguajes, para la definición y la manipulación de los datos. El lenguaje de definición de datos (DDL, Data Definition Language, por sus siglas en inglés) provee de los medios necesarios para definir los datos con precisión, especificando las distintas estructuras. Acorde con el modelo de arquitectura de tres niveles, habrá un lenguaje de definición de la estructura lógica global, otro para la definición de la estructura interna, y un tercero para la definición de las estructuras externas.

El lenguaje de manipulación de datos (DML, Data Manipulation/ Management Language, por sus siglas en inglés), que es el encargado de facilitar a los usuarios el acceso y manipulación de los datos. Pueden diferenciarse en procedimentales (aquellos que requieren qué datos se necesitan y cómo obtenerlos) y no procedimentales (que datos se necesitan, sin especificar como obtenerlos), y se encargan de la recuperación de los datos almacenados, de la inserción y supresión de datos en la base de datos, y de la modificación de los existentes.

3.2. SMBD (SISTEMAS MANEJADORES DE BASE DE DATOS)

Son programas de software para la administración de las bases de datos; que permiten: almacenar, manipular y recuperar datos en una computadora.

3.2.1. MySQL

MySQL es un sistema de gestión de bases de datos relacional, licenciado bajo la GNU GPL; por sus siglas en inglés General Public License (Licencia Pública General de GNU). Su diseño multihilo le permite soportar una gran carga de forma muy eficiente. MySQL fue creada por la empresa sueca MySQL AB, que mantiene el copyright del código fuente del servidor SQL, así como también de la marca.

Aunque MySQL es software libre, MySQL AB distribuye una versión comercial de MySQL, que no se diferencia de la versión libre más que en el soporte técnico que se ofrece, y la posibilidad de integrar este gestor en un software propietario, ya que de no ser así, se vulneraría la licencia GPL.

Este gestor de bases de datos es, probablemente, el gestor más usado en el mundo del software libre, debido a su gran rapidez y facilidad de uso. Esta gran aceptación es debida, en parte, a que existen infinidad de librerías y otras

herramientas que permiten su uso a través de gran cantidad de lenguajes de programación, además de su fácil instalación y configuración.

3.2.1.1. Historia de MySQL

MySQL surgió como un intento de conectar el gestor mSQL a las tablas propias de MySQL AB, usando sus propias rutinas a bajo nivel. Tras unas primeras pruebas, vieron que mSQL no era lo bastante flexible para lo que necesitaban, por lo que tuvieron que desarrollar nuevas funciones. Esto resultó en una interfaz SQL a su base de datos, con una interfaz totalmente compatible a mSQL.

Se comenta en el manual que no se sabe con certeza de donde proviene su nombre. Por un lado dicen que sus librerías han llevado el prefijo 'my' durante los diez últimos años. Por otro lado, la hija de uno de los desarrolladores se llama My. No saben cuál de estas dos causas (aunque bien podrían tratarse de la misma), han dado lugar al nombre de este conocido gestor de bases de datos.

La versión estable de este gestor a días de hoy es la 3.23.49. Se puede encontrar más información sobre este gestor en el manual.

3.2.1.2. Características de MySQL

Las principales características de este gestor de bases de datos son las siguientes:

1. Aprovecha la potencia de sistemas multiprocesador, gracias a su implementación multihilo.
2. Soporta gran cantidad de tipos de datos para las columnas.
3. Gran portabilidad entre sistemas.
4. Soporta hasta 32 índices por tabla.
5. Gestión de usuarios y passwords, manteniendo un muy buen nivel de seguridad en los datos.

3.2.1.3. ¿Qué es lo que le falta?

MySQL surgió cómo una necesidad de un grupo de personas sobre un gestor de bases de datos rápido, por lo que sus desarrolladores fueron implementando únicamente lo que precisaban, intentando hacerlo funcionar de forma óptima. Es por ello que, aunque MySQL se incluye en el grupo de sistemas de bases de datos relacionales, carece de algunas de sus principales características:

1. Subconsultas: tal vez ésta sea una de las características que más se echan en falta, aunque gran parte de las veces que se necesitan, es posible reescribirlas de manera que no sean necesarias.

2. SELECT INTO TABLE: Esta característica propia de Oracle, todavía no está implementada.

3. Gatillos y Procedimientos de almacenado: Se tiene pensado incluir el uso de procedimientos de almacenado en la base de datos, pero no el de gatillos, ya que los gatillos reducen de forma significativa el rendimiento de la base de datos, incluso en aquellas consultas que no los activan.

4. Transacciones: a partir de las últimas versiones ya hay soporte para transacciones, aunque no por defecto (se ha de activar un modo especial).

5. Integridad referencial: aunque admite la declaración de claves ajenas en la creación tablas, internamente no las trata de forma diferente al resto de campos.

Los desarrolladores comentan en la documentación que todas estas carencias no les resultaban un problema, ya que era lo que ellos necesitaban. De hecho, MySQL fue diseñada con estas características, debido a que lo que buscaban era un gestor de bases de datos con una gran rapidez de respuesta. Pero ha sido con la distribución de MySQL por Internet, cuando más y más gente les están pidiendo estas funcionalidades, por lo que serán incluidas en futuras versiones del gestor.

3.2.2. ORACLE

Oracle es un sistema de gestión de base de datos relacional (o RDBMS por el acrónimo en inglés de Relational Data Base Management System), fabricado por Oracle Corporation.

Se considera a Oracle como uno de los sistemas de bases de datos más completos, destacando su:

- Soporte de transacciones.
- Estabilidad.
- Escalabilidad.
- Es multiplataforma.

Un aspecto que ha sido criticado por algunos especialistas es la seguridad de la plataforma, y las políticas de suministro de parches de seguridad, modificadas a comienzos de 2005 y que incrementan el nivel de exposición de los usuarios. En los parches de actualización provistos durante el primer semestre de 2005 fueron corregidas 22 vulnerabilidades públicamente conocidas, algunas de ellas con una antigüedad de más de 2 años.

Oracle es básicamente una herramienta cliente/servidor para la gestión de Bases de Datos. Es un producto vendido a nivel mundial, aunque la gran potencia que tiene y su elevado precio hace que sólo se vea en empresas muy grandes y multinacionales, por norma general. En el desarrollo de páginas web pasa lo mismo: como es un sistema muy caro no está tan extendido como otras bases de datos, por ejemplo, Access, MySQL, SQL Server, etc.

3.2.3. ACCESS

Microsoft Access es un programa Sistema de gestión de base de datos relacional creado y modificado por Microsoft para uso personal de pequeñas organizaciones. Es un componente de la suite Microsoft Office aunque no se incluye en el paquete "básico". Una posibilidad adicional es la de crear ficheros con bases de datos que pueden ser consultados por otros programas.

Es un software de gran difusión entre pequeñas empresas (PYMES) cuyas bases de datos no requieren de excesiva potencia, ya que se integra perfectamente con el resto de aplicaciones de Microsoft y permite crear pequeñas aplicaciones con unos pocos conocimientos del Programa. Tiene un sistema de seguridad de cifrado bastante primitivo y puede ser la respuesta a proyectos de programación de pequeños y medianos tamaños.

Entre sus mayores inconvenientes figuran que no es multiplataforma, pues sólo está disponible para sistemas operativos de Microsoft, y que no permite transacciones. Su uso es inadecuado para grandes proyectos de software que requieren tiempos de respuesta críticos o muchos accesos simultáneos a la base de datos.

3.2.4. Microsoft SQL Server

Microsoft SQL Server es un sistema de gestión de bases de datos relacionales (SGBD) basado en el lenguaje Transact-SQL, y específicamente en Sybase IQ, capaz de poner a disposición de muchos usuarios grandes cantidades de datos de manera simultánea.

Microsoft SQL Server constituye la alternativa de Microsoft a otros potentes sistemas gestores de bases de datos como son Oracle, Sybase ASE, PostgreSQL, Interbase, Firebird o MySQL.

3.2.4.1. Características de Microsoft SQL Server

- Soporte de transacciones.
- Escalabilidad, estabilidad y seguridad.
- Soporta procedimientos almacenados.
- Incluye también un potente entorno gráfico de administración, que permite el uso de comandos DDL y DML gráficamente.
- Permite trabajar en modo cliente-servidor, donde la información y datos se alojan en el servidor y las terminales o clientes de la red sólo acceden a la información.
- Además permite administrar información de otros servidores de datos.

Este sistema incluye una versión reducida, llamada MSDE con el mismo motor de base de datos pero orientado a proyectos más pequeños, que en sus versiónes 2005 y 2008 pasa a ser el SQL Express Edition, que se distribuye en forma gratuita.

Es común desarrollar completos proyectos complementando Microsoft SQL Server y Microsoft Access a través de los llamados ADP; por sus siglas en inglés Access Data Project (Proyecto de Acceso de Datos). De esta forma se completa la base de datos (Microsoft SQL Server), con el entorno de desarrollo (VBA Access), a través de la implementación de aplicaciones de dos capas mediante el uso de formularios Windows.

Para el desarrollo de aplicaciones más complejas (tres o más capas), Microsoft SQL Server incluye interfaces de acceso para varias plataformas de desarrollo, entre ellas .NET, pero el servidor sólo está disponible para Sistemas Operativos Windows.

3.3. DIFERENCIA ENTRE UN SGBD Y UNA BD

Base de Datos es un conjunto exhaustivo no redundante de datos estructurados organizados independientemente de su utilización y su implementación en máquina accesibles en tiempo real y compatibles con usuarios concurrentes con necesidad de información diferente y no predicable en tiempo.

Un Sistema Manejador de Bases de Datos (SMBD) es un programa o conjunto de aplicaciones para almacenar, manipular y recuperar información en una BD. Muchos de los SMBD operan en un lenguaje común como el SQL. Entre los principales programas de bases de datos integradas a los SMBD figuran ORACLE, SQL SERVER y MS ACCESS.

3.4. RESUMEN

Para plasmar los tres niveles en el enfoque o modelo de datos seleccionado, es necesaria una aplicación que actúe de interfaz entre el usuario, los modelos y el sistema físico. Esta es la función que desempeñan los SGBD, ya reseñados, y que pueden definirse como un paquete generalizado de software, que se ejecuta en un sistema computacional anfitrión, centralizando los accesos a los datos y actuando de interfaz entre los datos físicos y el usuario. Las principales funciones que debe cumplir un SGBD se relacionan con la creación y mantenimiento de la base de datos, el control de accesos, la manipulación de datos de acuerdo con las necesidades del usuario, el cumplimiento de las normas de tratamiento de datos, evitar redundancias e inconsistencias y mantener la integridad. Se han señalado como componentes de un sistema ideal de gestión de bases de datos los siguientes. Un lenguaje de definición de esquema conceptual.

1. Un sistema de diccionario de datos.
2. Un lenguaje de especificación de paquetes de entrada/salida.
3. Un lenguaje de definición de esquemas de base de datos.
4. Una estructura simétrica de almacenamiento de datos.
5. Un módulo de transformación lógica a física.
6. Un subsistema de privacidad de propósito general.
7. Un subsistema de integridad de propósito general
8. Un subsistema de reserva y recuperación de propósito general.
9. Un generador de programas de aplicación.
10. Un generador de programas de informes.
11. Un lenguaje de consulta de propósito general.

El SGBD incorpora como herramienta fundamental dos lenguajes, para la definición y la manipulación de los datos. El lenguaje de definición de datos (DDL, Data Definition Language, por sus siglas en inglés) provee de los medios necesarios para definir los datos con precisión, especificando las distintas estructuras. Acorde con el modelo de arquitectura de tres niveles, habrá un lenguaje de definición de la estructura lógica global, otro para la definición de la estructura interna, y un tercero para la definición de las estructuras externas.

El lenguaje de manipulación de datos (DML, Data Manipulation/ Management Language, por sus siglas en inglés), que es el encargado de facilitar a los usuarios el acceso y manipulación de los datos. Pueden diferenciarse en procedimentales (aquellos que requieren qué datos se necesitan y cómo obtenerlos) y no procedimentales (que datos se necesitan, sin especificar como obtenerlos), y se encargan de la recuperación de los datos almacenados, de la inserción y supresión de datos en la base de datos, y de la modificación de los existentes.

CAPÍTULO 4: CONSTRUIR MODELOS DE BASES DE DATOS SIGUIENDO EL PROCESO DE DISEÑO PARA LA REALIZACIÓN DE CONSULTAS DE INFORMACIÓN

4.1. CONSTRUCCIÓN DEL MODELO

Una consulta es una instrucción de solicitud para recuperar información. La parte de un lenguaje de manipulación de datos (DML) que implica recuperación de información se llama lenguaje de consultas.

4.2. RELACIONES

Es una correspondencia o asociación entre dos o más entidades. Cada relación tiene un nombre que describe su función. Las relaciones se representan gráficamente mediante rombos y su nombre aparece en el interior.

Las entidades que están involucradas en una determinada relación se denominan entidades participantes. El número de participantes en una relación es lo que se denomina grado de la relación. Por lo tanto, una relación en la que participan dos entidades es una relación binaria; si son tres las entidades participantes, la relación es ternaria; etc.

4.3. CONECTIVIDAD Y CARDINALIDAD

La conectividad se utiliza para describir la clasificación de relaciones.

- Uno a uno
- Uno a muchos a muchos

La cardinalidad con la que una entidad participa en una relación especifica el número mínimo y el número máximo de correspondencias en las que puede tomar parte cada ocurrencia de dicha entidad. La participación de una entidad en una relación es obligatoria (total) si la existencia de cada una de sus ocurrencias requiere la existencia de, al menos, una ocurrencia de la otra entidad participante. Si no, la participación es opcional (parcial). Las reglas que definen la cardinalidad de las relaciones son las reglas de negocio.

4.4. ATRIBUTOS

Es una característica de interés o un hecho sobre una entidad o sobre una relación. Los atributos representan las propiedades básicas de las entidades y de las relaciones. Toda la información extensiva es portada por los atributos. Gráficamente, se representan mediante bolitas que cuelgan de las entidades o relaciones a las que pertenecen.

Cada atributo tiene un conjunto de valores asociados denominado dominio. El dominio define todos los valores posibles que puede tomar un atributo. Puede haber varios atributos definidos sobre un mismo dominio.

Los atributos pueden ser simples o compuestos. Un atributo simple es un atributo que tiene un sólo componente, que no se puede dividir en partes más pequeñas que tengan un significado propio. Un atributo compuesto es un atributo con varios componentes, cada uno con un significado por sí mismo. Un grupo de atributos se representa mediante un atributo compuesto cuando tienen afinidad en cuanto a su significado, o en cuanto a su uso. Un atributo compuesto se representa gráficamente mediante un óvalo.

Los atributos también pueden clasificarse en monovalentes o polivalentes. Un atributo monovalente es aquel que tiene un sólo valor para cada ocurrencia de la entidad o relación a la que pertenece. Un atributo polivalente es aquel que tiene varios valores para cada ocurrencia de la entidad o relación a la que pertenece. A estos atributos también se les denomina multivaluados, y pueden tener un número máximo y un número mínimo de valores. La cardinalidad de un atributo indica el número mínimo y el número máximo de valores que puede tomar para cada ocurrencia de la entidad o relación a la que pertenece. El valor por omisión es (1,1).

Por último, los atributos pueden ser derivados. Un atributo derivado es aquel que representa un valor que se puede obtener a partir del valor de uno o varios atributos, que no necesariamente deben pertenecer a la misma entidad o relación.

4.5. TABLAS

Existe una diferencia fundamental entre una tabla en el diseño para la realización de consultas de información y una en el modelo relacional, ya que mientras que ésta se define como un conjunto de tuplas, una tabla es en realidad un multiconjunto de filas, por lo que admite filas repetidas.

Los elementos de la tabla pueden ser tanto definiciones de columnas como definiciones de restricciones de tabla. Una columna debe definirse sobre un dominio o bien directamente con un tipo de datos.

4.6. DIAGRAMAS ENTIDAD-RELACIÓN

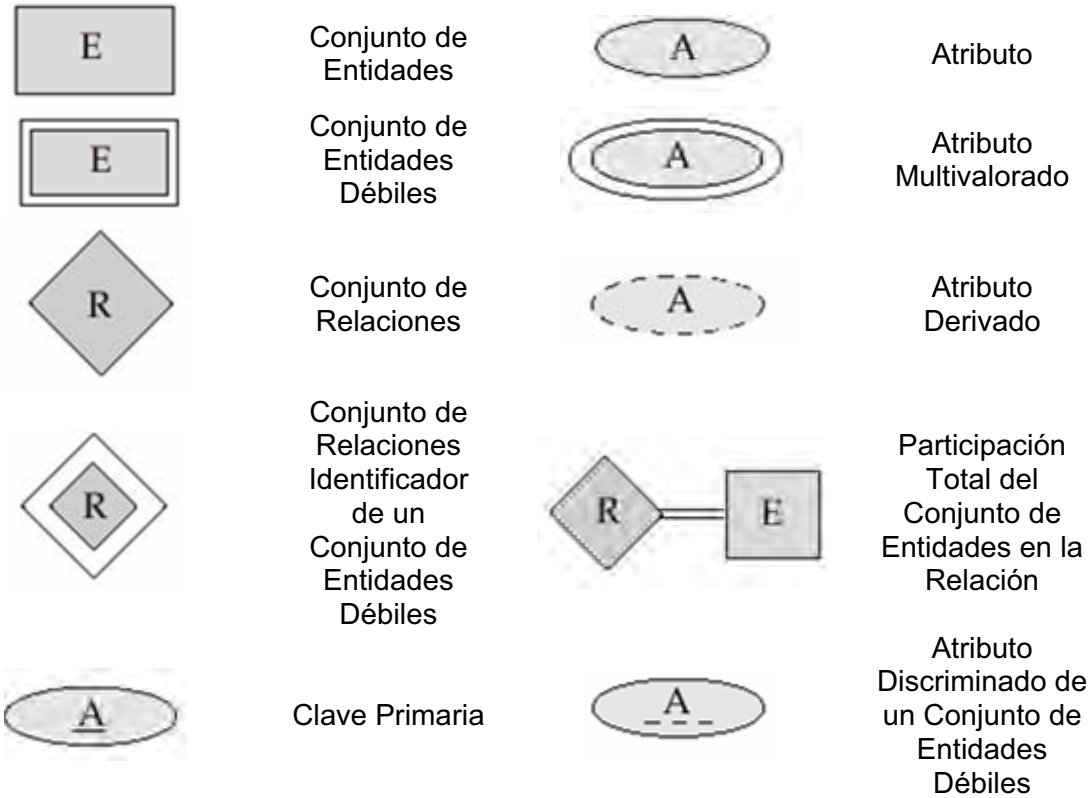
El modelo de datos entidad-relación (E-R) está basado en una percepción del mundo real que consta de una colección de objetos básicos, llamados entidades, y de relaciones entre estos objetos. Una entidad es una cosa u objeto en el mundo real que es distinguible de otros objetos. Por ejemplo, cada persona es una entidad, y las cuentas bancarias pueden ser consideradas entidades.

Las entidades se describen en una base de datos mediante un conjunto de atributos. Una relación es una asociación entre varias entidades. El conjunto de todas las entidades del mismo tipo, y el conjunto de todas las relaciones del mismo tipo, se denominan respectivamente conjunto de entidades y conjunto de relaciones.

La estructura lógica general de una base de datos se puede expresar gráficamente mediante un diagrama E-R, que consta de los siguientes componentes:

- Rectángulos, que representan conjuntos de entidades.
- Elipses, que representan atributos.
- Rombos, que representan relaciones entre conjuntos de entidades.
- Líneas, que unen los atributos con los conjuntos de entidades y los conjuntos de entidades con las relaciones.

La figura 4.1 muestra el conjunto de símbolos que se usan en los diagramas E-R. no hay ningún estándar universal para la notación de diagramas E-R.



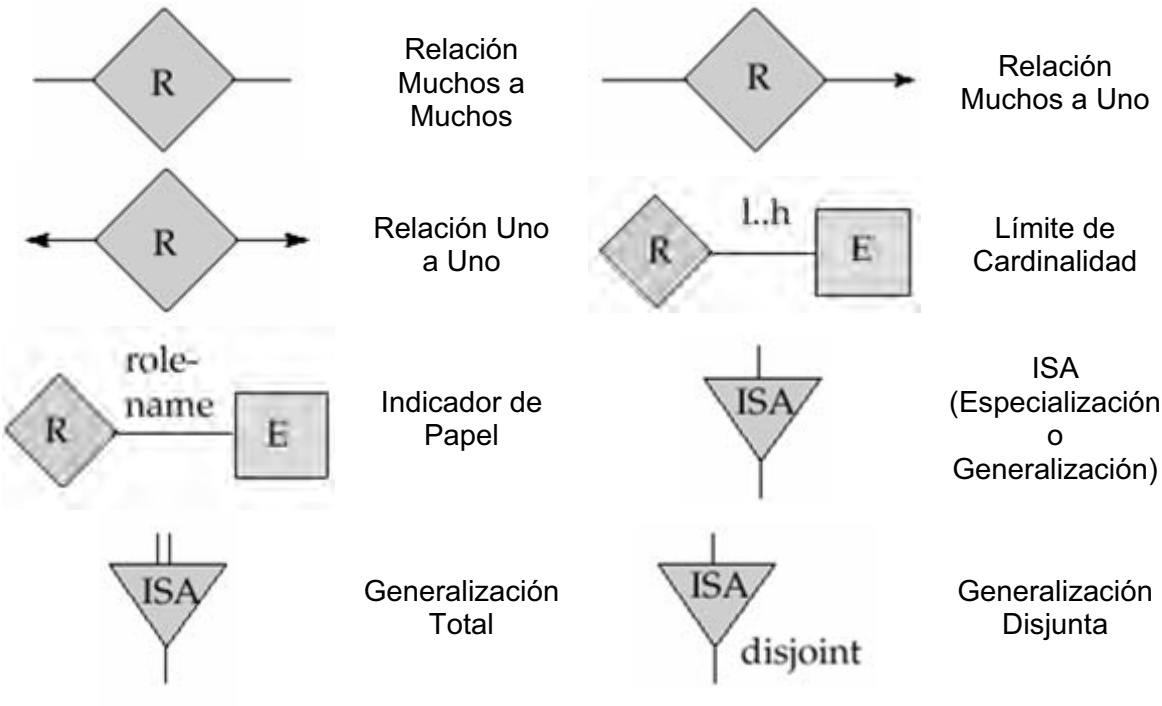


Figura 4.1. *Notación empleada para elaborar modelos E-R*

Cada componente se etiqueta con la entidad o relación que representa. Además de entidades y relaciones, el modelo E-R representa ciertas restricciones que los contenidos de la base de datos deben cumplir. Una restricción importante es la correspondencia de cardinalidades, que expresa el número de entidades con las que otra entidad se puede asociar a través de un conjunto de relaciones.

4.7. RELACIONES REDUNDANTES Y RELACIONES RECURSIVAS

Relaciones Redundantes: Una relación es redundante cuando se puede obtener la misma información que ella aporta mediante otras relaciones. El hecho de que haya dos caminos diferentes entre dos entidades no implica que uno de los caminos corresponda a una relación redundante, eso dependerá del significado de cada relación. La relación redundante se puede mantener si nos obligan a almacenar información de la relación.

En la figura 4.2 podemos observar un ejemplo de relaciones redundantes, dado que un médico tiene una especialidad ya definida pertenece a un cierto departamento, el cual está adscrito a la misma.

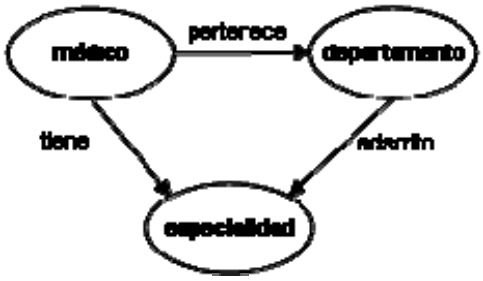


Figura 4.2. *Ejemplo de relación redundante.*

Relaciones Recursivas: Una relación recursiva es una relación donde la misma entidad participa más de una vez en la relación con distintos papeles. El nombre de estos papeles es importante para determinar la función de cada participación.

La figura 4.3 muestra un ejemplo de una relación recursiva donde un cuentahabiente es propietario de varias cuentas de un banco.

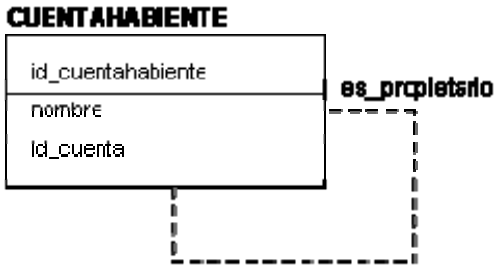


Figura 4.3. *Ejemplo de relación recursiva.*

4.8. TIPOS DE LLAVES (CLAVES): SIMPLES/COMPUESTAS

Una clave candidata de una relación es un conjunto de atributos que identifican unívoca y mínimamente cada tupla de la relación. Por la propia definición de relación. siempre hay, al menos, una clave candidata, ya que al ser una relación un conjunto no existen dos tuplas iguales y, por tanto, el conjunto de todos los atributos siempre tiene que identificar unívocamente a cada tupla; si no se cumpliera la condición de minimalidad se eliminarían aquellos atributos que lo impidiesen.

Una relación puede tener más de una clave candidata, entre las cuales se pueden distinguir:

Clave simple: es aquella clave candidata que el usuario escogerá, consideraciones ajenas al modelo relacional, para identificar las tuplas de la relación. Cuando sólo existe una clave candidata, ésta será la clave primaria

Claves compuestas: son aquellas claves candidatas que no han sido escogido como clave primaria

Se denomina clave ajena de una relación R2 a un conjunto no vacío de atributos cuyos valores han de coincidir con los valores de la clave candidata de una relación Rl

(Rl Y R2 no son necesariamente distintas). Cabe destacar que la clave ajena correspondiente clave candidata han de estar definidas sobre el mismo dominio.

4.9. RESUMEN

Una consulta es una instrucción de solicitud para recuperar información. La parte de un lenguaje de manipulación de datos (LMD) que implica recuperación de información se llama lenguaje de consultas.

Las relaciones son una correspondencia o asociación entre dos o más entidades. Cada relación tiene un nombre que describe su función. Las relaciones se representan gráficamente mediante rombos y su nombre aparece en el interior. Los atributos representan las propiedades básicas de las entidades y de las relaciones. Toda la información extensiva es portada por los atributos. Gráficamente, se representan mediante bolitas que cuelgan de las entidades o relaciones a las que pertenecen.

Existe una diferencia fundamental entre una tabla en el diseño para la realización de consultas de información y una en el modelo relacional, ya que mientras que ésta se define como un conjunto de tuplas, una tabla es en realidad un multiconjunto de filas, por lo que admite filas repetida.

Una clave candidata de una relación es un conjunto de atributos que identifican unívoca y mínimamente cada tupla de la relación. Por la propia definición de relación. siempre hay, al menos, una clave candidata, ya que al ser una relación un conjunto no existen dos tuplas iguales y, por tanto, el conjunto de todos los atributos siempre tiene que identificar unívocamente a cada tupla; si no se cumpliera la condición de minimalidad se eliminarían aquellos atributos que lo impidiesen.

CAPÍTULO 5: DIAGRAMA ENTIDAD-RELACIÓN

5.1. CONCEPTO

El modelo de datos entidad-relación (E-R) está basado en una percepción del mundo real que consta de una colección de objetos básicos, llamados entidades, y de relaciones entre estos objetos.

5.1.1. Entidad

Una entidad es una «cosa» u «objeto» en el mundo real que es distinguible de otros objetos. Por ejemplo, cada persona es una entidad, y las cuentas bancarias pueden ser consideradas entidades.

Las entidades se describen en una base de datos mediante un conjunto de atributos. Por ejemplo, los atributos número-cuenta y saldo describen una cuenta particular de un banco y pueden ser atributos del conjunto de entidades cuenta. Análogamente, los atributos nombre-cliente, calle-cliente y ciudad-cliente pueden describir una entidad cliente.

Un atributo extra, id-cliente, se usa para identificar unívocamente a los clientes (dado que puede ser posible que haya dos clientes con el mismo nombre, dirección y ciudad. Se debe asignar un identificador único de cliente a cada cliente. En México, muchas empresas utilizan el número de la seguridad social de una persona (un número único que el Gobierno Mexicano asigna a cada persona) como identificador.de cliente *.

5.1.2. Relación

Una relación es una asociación entre varias entidades. Por ejemplo, una relación impositor asocia un cliente con cada cuenta que tiene. El conjunto de todas las entidades del mismo tipo, y el conjunto de todas las relaciones del mismo tipo, se denominan respectivamente conjunto de entidades y conjunto de relaciones.

5.2. CONSISTENCIA DE DATOS

Eliminando o controlando las redundancias de datos se reduce en gran medida el riesgo de que haya inconsistencias. Si un dato está almacenado una sola vez, cualquier actualización se debe realizar sólo una vez, y está disponible para todos los usuarios inmediatamente. Si un dato está duplicado y el sistema conoce esta redundancia, el propio sistema puede encargarse de garantizar que todas las copias se mantienen consistentes. Desgraciadamente, no todos los SGBD de hoy en día se encargan de mantener automáticamente la consistencia.

5.3. INTEGRIDAD DE DATOS

La integridad de los datos consiste en mantener la precisión y consistencia de los valores de los datos. Los mecanismos de seguridad protegen la integridad de los datos. También se pueden mantener en el diccionario de datos restricciones sobre los valores, aunque es una tarea que resulta complicada.

Por último, resaltar que los mecanismos de copias de seguridad y restauración soportados por el SGBD deben preservar los datos de cualquier fallo del sistema.

5.4. INTEGRIDAD REFERENCIAL

La integridad referencial es una propiedad deseable en las bases de datos. Gracias a la integridad referencial se garantiza que una entidad (fila o registro) siempre se relaciona con otras entidades válidas, es decir, que existen en la base de datos. Implica que en todo momento dichos datos sean correctos, sin repeticiones innecesarias, datos perdidos y relaciones mal resueltas.

Todas las bases de datos relacionales gozan de esta propiedad gracias a que el software gestor de base de datos vela por su cumplimiento. En cambio, las bases de datos jerárquicas requieren que los programadores se aseguren de mantener tal propiedad en sus programas.

5.5. DIAGRAMAS

La estructura lógica general de una base de datos se puede expresar gráficamente mediante un diagrama E-R, que consta de los siguientes componentes:

- Rectángulos, que representan conjuntos de entidades.
- Elipses, que representan atributos.
- Rombos, que representan relaciones entre conjuntos de entidades.
- Líneas, que unen los atributos con los conjuntos de entidades y los conjuntos de entidades con las relaciones.
- Elipses dobles, que representan atributos multivalorados
- Elipses discontinuas, que denotan atributos derivados
- Líneas dobles, que indican participación total de una entidad en un conjunto de relaciones
- Rectángulos dobles, que representan conjuntos de entidades débiles

Cada componente se etiqueta con la entidad o relación que representa.

Como ilustración, considérese parte de una base de datos de un sistema bancario consistente en clientes y cuentas que tienen esos clientes. En la siguiente figura se muestra el diagrama E-R correspondiente.

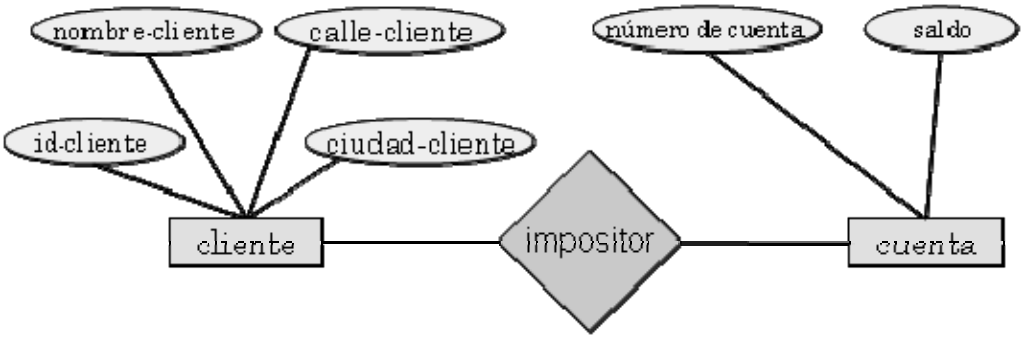


Figura 5.1. *Ejemplo de diagrama E-R*

El diagrama E-R indica que hay dos conjuntos de entidades cliente y cuenta, con distintos atributos. El diagrama también muestra la relación impositor entre cliente y cuenta.

Además de entidades y relaciones, el modelo E-R representa ciertas restricciones que los contenidos de la base de datos deben cumplir. Una restricción importante es la correspondencia de cardinalidades, que expresa el número de entidades con las que otra entidad se puede asociar a través de un conjunto de relaciones. Por ejemplo, si cada cuenta puede pertenecer sólo a un cliente, el modelo puede expresar esta restricción.

El modelo entidad-relación se utiliza habitualmente en el proceso de diseño de bases de datos.

5.6. DISEÑO

El modelo de datos E-R da una flexibilidad sustancial en el diseño de un esquema de bases de datos para modelar una empresa dada. En este apartado se considera cómo un diseñador de bases de datos puede seleccionar entre el amplio rango de alternativas. Entre las decisiones que se toman están las siguientes:

- Si se usa un atributo o un conjunto de entidades para representa un objeto
- Si un concepto del mundo real se expresa más exactamente mediante un conjunto de entidades o mediante un conjunto de relaciones
- Si se usa una relación ternaria o un par de relaciones binaras
- Si se usa un conjunto de entidades fuertes o débiles; un conjunto de entidades fuertes y sus conjuntos de entidades débiles dependientes se pueden considerar como un «objeto» en la base de datos, debido a que

la existencia de las entidades débiles depende de la entidad fuerte.

- Si el uso de la generalización es apropiado; la generalización, o una jerarquía de relaciones ES, contribuye a la modularidad por permitir que los atributos comunes de conjuntos de entidades similares se representen en un único lugar en un diagrama E-R.

- Si el uso de la agregación es apropiado; la agregación agrupa una parte de un diagrama E-R en un único conjunto de entidades, permitiendo tratar el conjunto de entidades de la agregación como una unidad única sin importar los detalles de su estructura interna.

Se verá que el diseñador de bases de datos necesita un buen entendimiento de la empresa que se modela para tomar estas decisiones.

5.6.1. Fases de diseño

Un modelo de datos de alto nivel sirve al diseñador de la base de datos para proporcionar un marco conceptual en el que especificar de forma sistemática los requisitos de datos de los usuarios de la base de datos que existen, y cómo se estructurará la base de datos para completar estos requisitos. La fase inicial del diseño de bases de datos, por tanto, es caracterizar completamente las necesidades de datos esperadas por los usuarios de la base de datos. El resultado de esta fase es una especificación de requisitos del usuario.

A continuación, el diseñador elige un modelo de datos y, aplicando los conceptos del modelo de datos elegido, traduce estos requisitos a un esquema conceptual de la base de datos. El esquema desarrollado en esta fase de diseño conceptual proporciona una visión detallada del desarrollo. Debido a que sólo se ha estudiado el modelo E-R hasta ahora, se usará éste para desarrollar el esquema conceptual. En términos del modelo E-R, el esquema especifica todos los conjuntos de entidades, conjuntos de relaciones, atributos y restricciones de correspondencia. El diseñador revisa el esquema para confirmar que todos los requisitos de datos se satisfacen realmente y no hay conflictos entre sí. También se examina el diseño para eliminar características redundantes. Lo importante en este punto es describir los datos y las relaciones, más que especificar detalles del almacenamiento físico.

Un esquema conceptual completamente desarrollado indicará también los requisitos funcionales de la empresa. En una especificación de requisitos funcionales los usuarios describen los tipos de operaciones (o transacciones) que se realizarán sobre los datos. Algunos ejemplos de operaciones son la modificación o actualización de datos, la búsqueda y recuperación de datos específicos y el borrado de datos. En esta fase de diseño conceptual se puede hacer una revisión del esquema para encontrar los requisitos funcionales.

El proceso de trasladar un modelo abstracto de datos a la implementación de la base de datos consta de dos fases de diseño finales. En la fase de diseño lógico, el diseñador traduce el esquema conceptual de alto nivel al modelo de datos de la implementación del sistema de base de datos que se usará. El diseñador usa el esquema resultante específico a la base de datos en la siguiente fase de diseño físico, en la que se especifican las características físicas de la base de datos.

En este apartado se tratan sólo los conceptos del modelo E-R usados en la fase de diseño del esquema conceptual. Se ha presentado una breve visión del proceso de diseño de bases de datos para proporcionar un contexto para la discusión del modelo de datos E-R.

5.6.2. Ejemplo de diseño de base de datos mediante el modelo e-r

5.6.2.1. Diseño de base de datos para un banco

Nos centramos ahora en los requisitos de diseño de la base de datos para el banco en más detalle y desarrollamos un diseño más realista. Sin embargo, no se intentará modelar cada aspecto del diseño de la base de datos para un banco; se considerarán sólo unos cuantos aspectos para ilustrar el proceso de diseño de bases de datos.

La especificación inicial de los requisitos de usuario se puede basar en entrevistas con los usuarios de la base de datos y en el análisis propio del diseñador del desarrollo. La descripción que surge de esta fase de diseño sirve como base para especificar la estructura conceptual de la base de datos. La siguiente lista describe los principales requisitos del banco.

- El banco está organizado en sucursales. Cada sucursal está ubicada en una ciudad particular y se identifica por un nombre único. El banco supervisa los activos de cada sucursal.

- Los clientes del banco se identifican mediante sus valores de id-cliente. El banco almacena cada nombre de cliente, y la calle y ciudad donde viven los clientes. Los clientes pueden tener cuentas y pueden pedir préstamos. Un cliente puede estar asociado con un banquero particular, que puede actuar como responsable de préstamos o banquero personal para un cliente.

- Los empleados del banco se identifican mediante sus valores de id-empleado. La administración del banco almacena el nombre y número de teléfono de cada empleado, los nombres de los subordinados del empleado, y el número id-empleado del jefe del empleado. El banco también mantiene registro de la fecha de comienzo del contrato del empleado, así como su antigüedad.

- El banco ofrece dos tipos de cuentas: cuentas de ahorro y cuentas corrientes. Las cuentas pueden asociarse a más de un cliente y un cliente puede tener más de una cuenta. Cada cuenta está asignada a un único número de cuenta. El banco mantiene un registro del saldo de cada cuenta y la fecha más reciente en que la cuenta fue accedida por cada cliente que mantiene la cuenta. Además, cada cuenta de ahorro tiene un tipo de interés y para cada cuenta corriente se almacena el descubierto.

- Un préstamo tiene lugar en una sucursal particular y puede estar asociado a uno o más clientes. Un préstamo se identifica mediante un único número de préstamo. Para cada préstamo el banco mantiene

registro del importe del préstamo y de los pagos del préstamo. Aunque un número de pago del préstamo no identifica de forma única un pago entre todos los préstamos del banco, un número de pago identifica un pago particular para un préstamo específico. Para cada pago se almacenan la fecha y el importe.

En un desarrollo de un banco real, el banco mantendría información de los abonos y cargos en las cuentas de ahorros y en las cuentas corrientes, igual que se mantiene registro de los pagos para los préstamos. Debido a que los requisitos del modelo para este seguimiento son similares, y para mantener nuestro ejemplo reducido, en este modelo no se mantiene un seguimiento de tales abonos y cargos.

5.6.2.2. Designación de los conjuntos de entidades

La especificación de los requisitos de datos sirve como punto de partida para la construcción de un esquema conceptual para la base de datos.

- El conjunto de entidades sucursal, con los atributos nombre-sucursal, ciudad-sucursal y activo.

- El conjunto de entidades cliente, con los atributos id-cliente, nombre-cliente, calle-cliente y ciudad-cliente. Un posible atributo adicional es nombre-banquero.

- El conjunto de entidades empleado, con los atributos id-empleado, nombre-empleado, número-teléfono, sueldo y jefe. Algunas características descriptivas adicionales son el atributo multivalorado nombre-subordinado, el atributo base fecha-comienzo y el atributo derivado antigüedad.

- Dos conjuntos de entidades cuenta -cuenta-ahorro y cuenta-corriente- con los atributos comunes número-cuenta y saldo; además, cuenta-ahorro tiene el atributo tipo-interés y cuenta-corriente tiene el atributo descubierto.

- El conjunto de entidades préstamo, con los atributos número-préstamo, importe y sucursal-origen.

- El conjunto de entidades débiles pago-préstamo, con los atributos número-pago, fecha-pago e importe-pago.

5.6.2.3. Designación de los conjuntos de relaciones

Se especifican los siguientes conjuntos de relaciones y correspondencia de cardinalidades:

- prestatario, un conjunto de relaciones varios a varios entre cliente y préstamo.

- préstamo-sucursal, un conjunto de relaciones varios a uno que indica la

sucursal en que se ha originado un préstamo. Nótese que este conjunto de relaciones reemplaza al atributo sucursal-origen del conjunto de entidades préstamo.

- pago-préstamo, un conjunto de relaciones uno a varios de préstamo a pago, que documenta que se ha realizado un pago de un préstamo.

- impositor, con el atributo de relación fecha-acceso, un conjunto de relaciones varios a varios entre cliente y cuenta, indicando que un cliente posee una cuenta.

- banquero-consejero, con el atributo de relación tipo, un conjunto de relaciones varios a uno que expresa que un cliente puede ser aconsejado por un empleado del banco, y que un empleado del banco puede aconsejar a uno o más clientes. Nótese que este conjunto de relaciones ha reemplazado al atributo nombre-banquero del conjunto de entidades cliente.

- trabaja-para, un conjunto de relaciones entre entidades empleado con papeles que indican jefe y trabajador; la correspondencia de cardinalidades expresa que un empleado trabaja para un único jefe, y que un jefe supervisa uno o más empleados. Nótese que este conjunto de relaciones reemplaza el atributo jefe de empleado.

5.6.3. Diagrama E-R

Conforme a lo discutido se presenta ahora el diagrama E-R completo para el ejemplo del banco. En la figura 5.2 se muestra la representación completa de un modelo conceptual de un banco, expresada en términos de los conceptos E-R. El diagrama incluye los conjuntos de entidades, atributos, conjuntos de relaciones y correspondencia de cardinalidades.

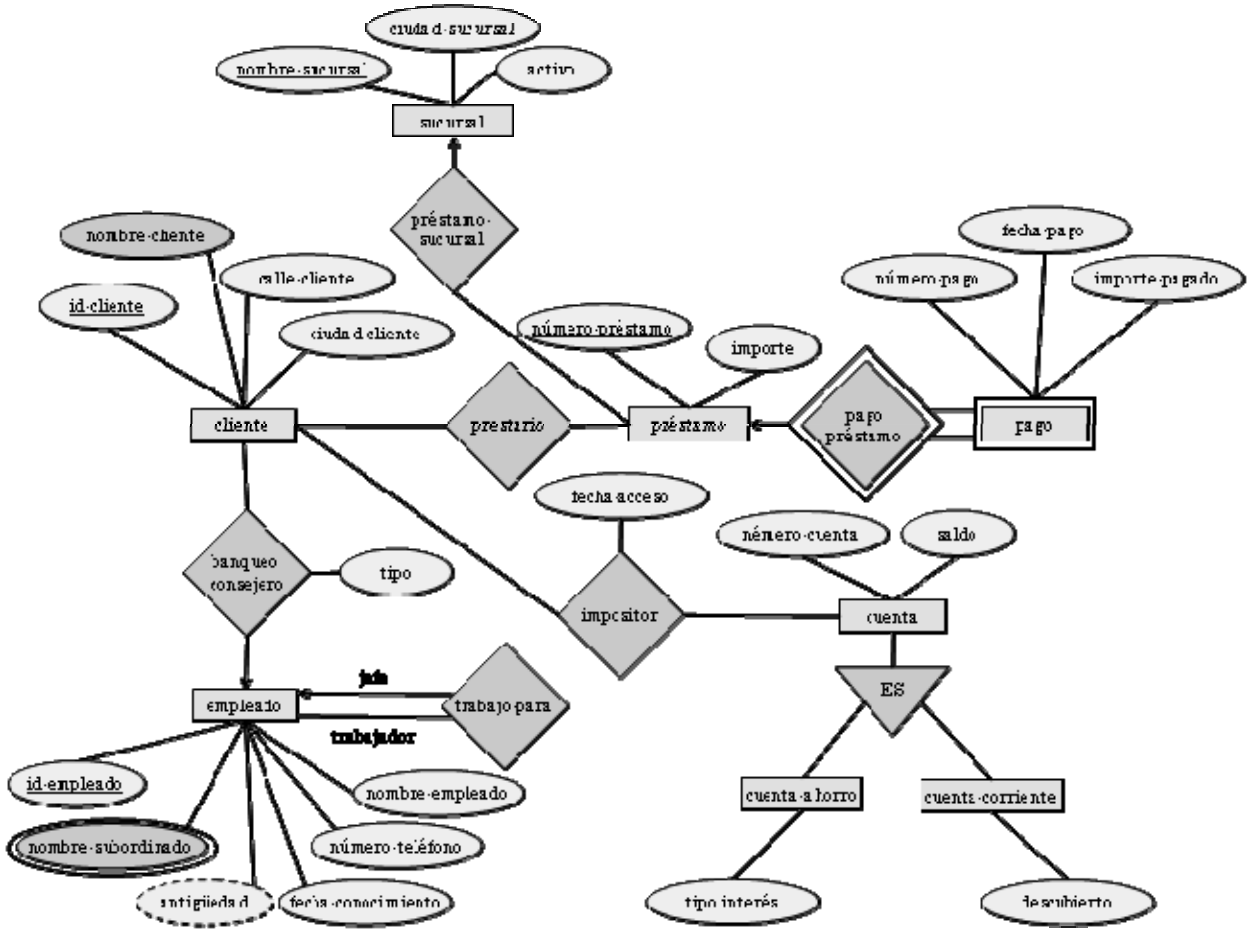


Figura 5.2. *Diagrama E-R para un banco.*

5.6.3.1. Claves

Es necesario tener una forma de especificar como las entidades dentro de un conjunto de entidades dado y las relaciones dentro de un conjunto de relaciones dado son distinguibles. Conceptualmente las entidades y las relaciones individuales son distintas; desde una perspectiva de base de datos, sin embargo, la diferencia entre ellas se debe expresar en término de sus atributos.

Por lo tanto, los valores de sus atributos de una entidad deben ser tales que permitan identificar unívocamente a la entidad. En otras palabras, no se permite que ningún par de entidades tengan exactamente los mismos valores de sus atributos.

Una clave permite identificar un conjunto de atributos suficiente para distinguir las entidades entre sí. Las claves también ayudan a identificar unívocamente a las relaciones y así a distinguir las relaciones entre sí.

Una superclave es un conjunto de uno o más atributos que, tomados colectivamente, permiten identificar de forma única una entidad en el conjunto de entidades. Por ejemplo, el atributo id-cliente del conjunto de entidades cliente es suficiente para distinguir una entidad cliente de las otras. Así, id-cliente es una superclave. El atributo nombre-cliente de cliente no es una superclave, porque varias personas podrían tener el mismo nombre.

El concepto de una superclave no es suficiente para lo que aquí se propone, ya que, como se ha visto, una superclave puede contener atributos innecesarios. Si *K* es una superclave, entonces también lo es cualquier superconjunto de *K.* A menudo interesan las superclaves tales que los subconjuntos propios de ellas no son superclave. Tales superclaves mínimas se llaman claves candidatas.

- **Clave candidata:** Este registro de cada entrada en la tabla es imprescindible. Indicará de forma unívoca la identidad de la entrada a la que representa. Es habitual usar un número que se incrementa con cada inserción, o autonumérico. Puede haber más de una clave candidata en una tabla. Sólo una de ellas actuará como clave primaria.

Se usará el término clave primaria para denotar una clave candidata que es elegida por el diseñador de la base de datos como elemento principal para identificar las entidades dentro de un conjunto de entidades. Una clave (primaria, candidata y superclave) es una propiedad del conjunto de entidades, más que de las entidades individuales. Cualesquiera dos entidades individuales en el conjunto no pueden tener el mismo valor en sus atributos clave al mismo tiempo. La designación de una clave representa una restricción en el desarrollo del mundo real que se modela.

Otros tipos de claves usualmente utilizados son:

- **Clave alternativa:** Son aquellas claves candidatas que no han sido seleccionadas como clave primaria.

- **Clave simple:** Es una clave que está compuesta de un solo atributo.

- **Clave compuesta:** Es una clave que esta compuesta por más de un atributo.

5.6.3.2. Conversión del modelo entidad-relación a un esquema de base de datos (Conversión a tablas)

5.6.3.2.1. Introducción

El modelo es una representación visual que gráficamente nos da una perspectiva de como se encuentran los datos involucrados en un proyecto u organización.

Pero el modelo no nos presenta propiamente una instancia de los datos, un ejemplo que muestre con claridad algunos datos de muestra y como se relacionan en realidad. Por eso es conveniente crear un "esquema", el cual consiste en tablas las cuales en sus tuplas contienen instancias de los datos.

5.6.3.2.2. Transformación del modelo entidad-relación al modelo relacional

Para transformar un modelo entidad-relación a modelo relacional seguiremos las siguientes reglas:

- Toda entidad del modelo entidad-relación se transforma en una tabla.
- Cualquier atributo de una entidad se transforma en un campo dentro la tabla, manteniendo las claves primarias.
- Las relaciones N:M (muchos a muchos) se transforman en una nueva tabla que tendrá como clave primaria la concatenación de los atributos clave de las entidades que relaciona.
- En las relaciones 1:N (uno a muchos) se pueden tener dos casos:
    - Si la entidad que participa con cardinalidad máxima uno lo hace también con cardinalidad mínima uno, entonces se propaga el atributo de la entidad que tiene cardinalidad máxima 1 a la que tiene cardinalidad máxima N, desapareciendo el nombre de la relación. Si existen atributos en la relación éstos también se propagarán.
    - Si la entidad que participa con cardinalidad máxima uno lo hace también cardinalidad mínima cero, entonces se crea una nueva tabla formada por las claves de cada entidad y los atributos de la relación. La clave primaria de la nueva tabla será el identificador de la entidad que participa con cardinalidad máxima N.
- En el caso de las relaciones 1:1 (uno a uno) también pueden darse dos casos:
    - Si las entidades poseen cardinalidades (0,1), la relación se convierte en una tabla.
    - Si una de las entidades posee cardinalidad (0,1) y la otra (1,1), conviene propagar la clave de la entidad con cardinalidad (1,1) a la tabla resultante de la entidad con cardinalidad (0,1). Si ambas entidades poseen cardinalidades (1,1) se puede propagar la clave de cualquiera de ellas a la tabla resultante de la otra.
- En el caso de las relaciones N-arias se aplica la misma regla que para las relaciones N:M
- En el caso de las relaciones reflexivas supondremos que se trata de una relación binaria con la particularidad que las dos entidades son iguales y aplicaremos las reglas vistas en los puntos anteriores.

5.6.3.2.2.1. Relaciones N:M (muchos a muchos)

Supongamos el siguiente modelo entidad-relación.

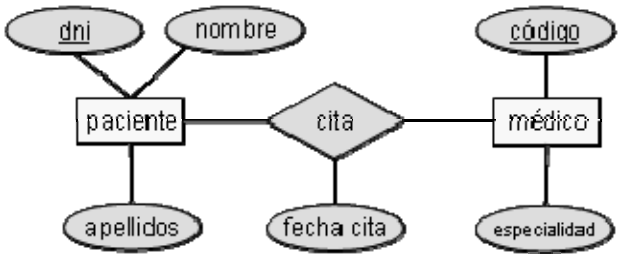


Figura 5.3. *Diagrama E-R para citas médicas (relación muchos a muchos).*

En este caso la relación "cita" se transforma en una nueva tabla cuya clave primaria estará formada por los atributos dni, que es la clave primaria de paciente, y código, que es la clave primaria de médico. Además tendrá como campo fecha cita, ya que este atributo forma parte de la relación.

El modelo relacional quedaría de la siguiente forma (en negrita las claves primarias):

- PACIENTE (**dni**, nombre, apellidos)
- MÉDICO (**código**, especialidad)
- CITA (**dni_paciente, código_médico**, fecha_cita)

PACIENTE

dni	nombre	apellidos

MÉDICO

código	especialidad

CITA

dni_paciente	**código_médico**	fecha_cita

Figura 5.4. *Transformación a tablas de un modelo e-r para citas médicas (relación muchos a muchos).*

5.6.3.2.2.2. Relaciones 1:N (uno a muchos)

Veamos ahora el caso de una relación 1:N. En el siguiente modelo entidad-relación un médico pertenece a una única especialidad (debe pertenecer a una obligatoriamente), y una especialidad tiene 1 o más médicos.

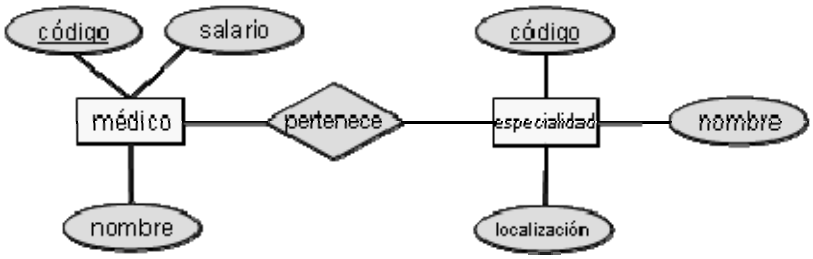


Figura 5.5. *Diagrama E-R para la especialidad a la que pertenece un médico (relación uno a muchos).*

En este caso se propaga el atributo código de especialidad a la tabla MÉDICO. El modelo relacional quedaría de la siguiente manera:

- MÉDICO (**dni** ,nombre, salario, código_especialidad)
- ESPECIALIDAD (**código**, nombre, localización)

MÉDICO

dni	nombre	salario	código_especialidad

ESPECIALIDAD

código	nombre	localización

Figura 5.6. *Transformación a tablas de un modelo e-r para especialidad a la que pertenece un médico (relación uno a muchos).*

5.6.3.2.2.3. Relaciones reflexivas

Veamos ahora como quedaría en el modelo relacional la siguiente relación reflexiva. En el siguiente modelo entidad-relación un MÉDICO es delegado de varios MÉDICOS y un MEDICO tiene obligatoriamente un delegado y sólo a uno.

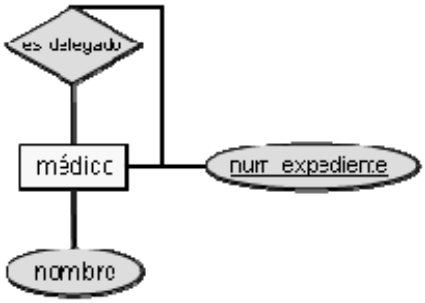


Figura 5.7. *Diagrama E-R para la delegación de un médico (relación reflexiva)*

Como podemos observar en las reglas de transformación, en este caso la relación reflexiva se trata como si fuera una relación binaria con la particularidad de que las dos entidades son iguales. Al tratarse de una relación 1:N se propagará la clave de la entidad MÉDICO a la entidad MÉDICO, quedando el modelo relacional de la siguiente forma:

- MÉDICO(**num_expediente**, nombre, num_expediente_delegado)

MÉDICO

num_expediente	nombre	num_expediente_delegado

Figura 5.8. *Transformación a tablas de un modelo e-r para la delegación de un médico (relación reflexiva).*

5.7. NORMALIZACIÓN

El proceso de normalización de una base de datos consiste en aplicar una serie de reglas a las relaciones obtenidas tras el paso del modelo E-R (entidad-relación) al modelo relacional.

Las bases de datos relacionales se normalizan para:

- Evitar la redundancia de los datos.
- Evitar problemas de actualización de los datos en las tablas.
- Proteger la integridad de los datos.

En el modelo relacional es frecuente llamar tabla a una relación, aunque para que una tabla bidimensional sea considerada como una relación tiene que cumplir con algunas restricciones:

- Cada columna debe tener su nombre único.

- No puede haber dos filas iguales. No se permiten los duplicados.
- Todos los datos en una columna deben ser del mismo tipo.

En general, las primeras tres formas normales son suficientes para cubrir las necesidades de la mayoría de las bases de datos. El creador de estas 3 primeras formas normales (o reglas) fue Edgar F. Codd, éste introdujo la normalización en un artículo llamado A Relational Model of Data for Large Shared Data Banks Communications of the ACM, Vol. 13, No. 6, June 1970, pp. 377-387.

5.7.1. 1° forma normal (1FN)

La primera de las formas normales que se van a estudiar, la primera forma normal, impone un requisito muy elemental a las relaciones; a diferencia de las demás formas normales, no exige información adicional como las dependencias funcionales.

Un dominio es atómico si se considera que los elementos del dominio son unidades indivisibles. Se dice que el esquema de una relación R está en la primera forma normal (1FN) si los dominios de todos los atributos de R son atómicos.

Un conjunto de nombres es un ejemplo de valor no atómico. Por ejemplo, si el esquema de la relación empleado incluyera el atributo hijos, los elementos de cuyo dominio son conjuntos de nombres, el esquema no se hallaría en la primera forma normal.
Los atributos compuestos, como el atributo dirección con sus atributos componentes calle y ciudad, tienen también dominios no atómicos.

Se da por supuesto que los enteros son atómicos, por lo que el conjunto de enteros es un dominio atómico; el conjunto de todos los conjuntos de enteros es un dominio no atómico. La diferencia estriba en que normalmente no se considera que los enteros tengan subpartes, pero sí se considera que los tienen los conjuntos de enteros, es decir, los enteros que componen el conjunto. Pero lo importante no es lo que sea el propio dominio, sino el modo en que se utilizan los elementos del dominio en la base de datos.

El dominio de todos los enteros no sería atómico si se considerara que cada entero es una lista ordenada de cifras.

Como ilustración práctica del punto anterior, considérese una organización que asigna a los empleados números de identificación de la manera siguiente: las dos primeras letras especifican el departamento y las cuatro cifras restantes son un número único para el empleado dentro de ese departamento. Ejemplos de estos números pueden ser IN0012 y EE1127. Estos números de identificación pueden dividirse en unidades menores y, por tanto, no son atómicos. Si el esquema de una relación tuviera un atributo cuyo dominio consistiera en números de identificación codificados como se ha indicado, el esquema no se hallaría en la primera forma normal.

Cuando se utilizan estos números de identificación, se puede averiguar el departamento de cada empleando escribiendo código que analice la estructura de los números de identificación. Ello exige programación adicional y la información queda

codificada en el programa de aplicación en vez de en la base de datos. Surgen nuevos problemas si se utilizan estos números de identificación como claves principales: Cada vez que un empleado cambia de departamento hay que cambiar su número de identificación, lo que puede constituir una tarea difícil, o en su defecto el código que interpreta ese número dará un resultado erróneo.

El empleo de atributos con el valor dado por el conjunto puede llevar a diseños con almacenamiento de datos redundantes, lo que, a su vez, puede dar lugar a inconsistencias. Por ejemplo, en lugar de representar la relación entre las cuentas y los clientes como una relación independiente impositor, puede que un diseñador de bases de datos esté tentado a almacenar un conjunto de titulares con cada cuenta y un conjunto de cuentas con cada cliente. Siempre que se cree una cuenta, o se actualice el conjunto de titulares de una cuenta, hay que llevar a cabo la actualización en dos lugares; no llevar a cabo las dos actualizaciones puede dejar la base de datos en un estado inconsciente. La conservación de sólo uno de estos conjuntos evitaría la información repetida, pero complicaría algunas consultas.

5.7.2. 2° forma normal (2NF)

Una relación está en segunda forma normal (2FN) solamente si todos los atributos son dependientes en forma completa de la clave.

Su nombre ya nos indica el hecho de que la segunda forma normal es por lo general el próximo paso de normalización y descomposición. Para ser accesible a la normalización, y poder ser puesta en segunda forma normal, la relación debe poseer las siguientes propiedades:

- Debe estar en primera forma normal
- Debe tener una clave compuesta.

La consecuencia inmediata de los requerimientos expresados más arriba es que cualquier relación en primera forma normal que tiene una clave simple, está automáticamente en segunda forma normal. Comencemos con un ejemplo en forma de tabla de una relación consistente en 17 atributos, que se presenta en la figura. La misma se encuentra en primera forma normal y tiene una clave compuesta que consiste en dos atributos P y Q. Estos están subrayados en la figura para mostrar que sirven como clave. La tupla de relación puede también escribirse linealmente en forma simbólicamente:

R = (A,B,C,D,E,F,G,H,I,L,M,N,O,P,Q)

El próximo paso es crear un grafo de dependencia, presentando aquí como figura. Debe notarse que este grafo se crea examinado con conocimientos y atributos para determinar como participan y relacionan entre ellos.

No resulta suficiente analizar la matríz de relación, la cual puede hacernos creer que existe una dependencia debido a que la muestra de la cual se ha extraído dicha relación es pequeña. Si somos inducidos a error por los datos existentes y construimos una dependencia donde esta no existe, se planteará un problema. Cuando lleguen nuevos datos que contradigan la dependencia, deberá dejarse de lado el esquema completo.

Supongamos en consecuencia que el grafo que se puede observar en la figura ha sido derivado en forma funcional y que expresa correctamente las dependencias. Resulta claro a partir de este grafo que los atributos que parten de P son dependientes solamente de este. De un modo similar los que parten de Q dependen solamente de este último. Solamente aquellos que parten de la línea de trazos que conecta a P y Q tienen dependencia completa de ambos.

5.7.3. 3° forma normal (3FN)

Un esquema de relación R está en tercera forma normal (3FN) respecto a un conjunto F de dependencias funcionales si, para todas las dependencias funcionales de F^+ de la forma $\alpha \rightarrow \beta$, donde $\alpha \subseteq R$ y $\beta \subseteq R$, se cumple al menos una de las siguientes condiciones:

- $\alpha \rightarrow \beta$ es una dependencia funcional trivial.
- α es una superclave de R.
- Cada atributo A de $\alpha \rightarrow \beta$ está contenido en alguna clave candidata de R.

Obsérvese que la tercera condición no dice que una sola clave candidata deba contener todos los atributos de $\alpha \rightarrow \beta$, cada atributo A de $\alpha \rightarrow \beta$ puede estar contenido en una clave candidata diferente.

Las dos primeras alternativas son iguales que las dos alternativas de la definición de FNBC (Forma Normal de Boyce-Codd). La tercera alternativa de la definición de 3FN parece bastante intuitiva, y no resulta evidente el motivo de su utilidad. Representa, en cierto sentido, una relajación mínima de las condiciones de FNBC que ayudan a asegurar que cada esquema tenga una descomposición que conserve las dependencias en 3FN. Su finalidad se aclarará más adelante, cuando se estudie la descomposición en 3FN.

Obsérvese que cualquier esquema que satisfaga FNBC satisface también 3FN, ya que cada una de sus dependencias funcionales satisfará una de las dos primeras alternativas. Por tanto, FNBC es una restricción más estricta que 3FN.

La definición de 3FN permite ciertas dependencias funcionales que no se permitían en FNBC. Una dependencia $\alpha \rightarrow \beta$ que sólo satisfaga la tercera alternativa de la definición de 3FN no se permitiría en FNBC, pero sí se permite en 3FN.

Volvamos al ejemplo Esquema-asesor. Se demostró que este esquema de relación no tiene una descomposición FNBC de reunión sin pérdida que conserve las dependencias. Resulta, sin embargo, que este esquema está en 3FN. Para comprobarlo, obsérvese que {nombre-cliente, nombre-sucursal} es una clave candidata de Esquema-asesor, por lo que el único atributo no contenido en una clave candidata de Esquema-asesor es nombre-asesor. Las únicas dependencias funcionales no triviales de la forma

$$\alpha \rightarrow \text{nombre-asesor}$$

incluyen {nombre-cliente, nombre-sucursal} como parte de a. Dado que {nombre-cliente, nombre-sucursal} es una clave candidata, estas dependencias no violan la definición de 3FN.

Como optimización, al buscar 3FN, se pueden considerar sólo las dependencias funcionales del conjunto dado F, en lugar de las de F+. Además, se pueden descomponer las dependencias de F de manera que su lado derecho consista sólo en atributos sencillos y utilizar el conjunto resultante en lugar de F.

Dada la dependencia $\alpha \rightarrow \beta$, se puede utilizar la misma técnica basada en el cierre de los atributos que se empleó para FNBC para comprobar si a es una superclave. Si a no es una superclave, hay que comprobar si cada atributo de f3 está contenido en alguna clave candidata de R; esta comprobación resulta bastante más costosa, ya que implica buscar las claves candidatas. De hecho, se ha demostrado que la comprobación de la 3FN resulta NP-duro; por tanto, resulta bastante improbable que haya algún polinomio con complejidad polinómica en el tiempo para esta tarea.

5.8. SISTEMA DE ARCHIVOS

El camino hacia las bases de datos ha sido largo y en el trayecto se han desarrollado un gran número de técnicas que forman los cimientos de las bd y de otras tecnologías.

Dentro de estas técnicas tenemos:

Archivos, Sistemas de Archivos, Acceso y manipulación de archivos, Indices. Pero por qué no es suficiente utilizar las herramientas anteriores y es necesario emplear un DBMS ?, no es lo mismo ?, cuál es la diferencia ?

No es lo mismo, un sistemas de archivos aún cuando pensemos que contiene lógicamente archivos y que se cuenta con índices para accesar los registros en ellos, carece de mucha funcionalidad que se emplea en la mayoría de las aplicaciones, aunque como se mencionó anteriormente, un DBMS emplea sistemas de archivos e índices para la manipulación de datos.

La funcionalidad adicional que provee un DBMS surge en base de algunos inconvenientes al emplear sistemas de archivos únicamente:

- Redundancia de datos e inconsistencias (Redundancy and Inconsistency): formatos, duplicidad de información (alto costo de almacenamiento y acceso) e incongruencia entre datos o copias de datos a lo largo del sistema.

- Dificultad de acceso (Access): en un sistema de archivos no se pueden obtener aquellos datos que no estén implantados en un programa, se carece de niveles de abstracción.

- Aislamiento de datos (Isolation): debido al factor tiempo y los requerimientos que van surgiendo se puede llegar a tener un problema al intentar separar un conjunto de datos porque ya se tiene un enredo en los archivos y se podría dar el caso en que dos usuarios estén manipulando la misma información pero de distinta manera.

- Integridad (Integrity): si queremos asociar dos datos, por ejemplo un alumno con una materia que esté cursando, debemos asegurarnos que ambas entidades existan, de lo contrario el alumno parecerá cursando un curso fantasma y viceversa. Para ello se emplean "restricciones de consistencia" (consistency constraints)

- Atomicidad (Atomicity): el problema clásico de transacciones bancarias, u ocurre toda la transacción o no ocurre nada pero no puede quedarse a medias.

- Acceso concurrente (Concurrent-access): garantizar un buen tiempo de respuesta, que todos los usuarios puedan accesar y/o modificar la información; esto no es fácil porque también hay que considerar que aunque los datos son los mismos, las aplicaciones no necesariamente lo son.

- Seguridad (Security): no toda la información debe estar disponible a todos los usuarios, algunos usuarios solo tendrán permisos de lectura, esto es relativamente sencillo de resolver aplicando "roles" pero el problema aumenta cuando en luegar de pensar en terminos de usuarios pensamos en terminos de aplicaciones ya que el número de roles y sus combinaciones aumenta y mantener las restricciones de seguridad se torna complicado.

Podemos entonces extender la definición de DBMS como un sistema robusto que es capaz de emplear algoritmos de almacenamiento y recuperación de información para poder implementar un modelo de datos de manera física garantizando que todas las transacciones que se realizan con respecto a dichos datos sean "ácidas" (Atomicity, Consistency, Isolation, Durability).

5.9. RESUMEN

El modelo de datos entidad-relación (E-R) se basa en una percepción del mundo real consistente en un conjunto de objetos básicos llamados entidades y en relaciones entre esos objetos.

El modelo está pensado principalmente para el proceso de diseño de la base de datos. Fue desarrollado para facilitar el diseño permitiendo la especificación de un esquema de la empresa. Tal esquema representa la estructura lógica general de la base de datos. Esta estructura general se puede expresar gráficamente mediante un diagrama E-R.

Una entidad es un objeto que existe y es distinguible de otros objetos. Se expresa la distinción asociando con cada entidad un conjunto de atributos que describen el objeto.

Una relación es una asociación entre diferentes entidades. Un conjunto de relaciones es una colección de relaciones del mismo tipo y un conjunto de entidades es una colección de entidades del mismo tipo.

La correspondencia de cardinalidades expresa el número de entidades a las que otra entidad se puede asociar a través de un conjunto de relaciones.

Una superclave de un conjunto de entidades es un conjunto de uno o más atributos que, tomados colectivamente, permiten identificar unívocamente una entidad en un conjunto de entidades. Se elige una superclave mínima para cada conjunto de entidades de entre sus superclaves; la superclave mínima se denomina la clave primaria del conjunto de entidades. Análogamente, un conjunto de relaciones es un conjunto de uno o más atributos que, tomados colectivamente, permiten identificar unívocamente una relación en un conjunto de relaciones. De igual forma se elige una superclave mínima para cada conjunto de relaciones de entre todas sus superclaves; ésta es la clave primaria del conjunto de relaciones.

Un conjunto de entidades que no tiene suficientes atributos para formar una clave primaria se denomina conjunto de entidades débiles. Un conjunto de entidades que tiene una clave primaria se denomina conjunto de entidades fuertes.

La especialización y la generalización definen una relación de contenido entre un conjunto de entidades de nivel más alto y uno o más conjuntos de entidades de nivel más bajo. La especialización es el resultado de tomar un subconjunto de un conjunto de entidades de nivel más alto para formar un conjunto de entidades de nivel más bajo. La generalización es el resultado de tomar la unión de dos o más conjuntos disjuntos de entidades (de nivel más bajo) para producir un conjunto de entidades de nivel más alto. Los atributos de los conjuntos de entidades de nivel más alto los heredan los conjuntos de entidades de nivel más bajo.

La agregación es una abstracción en la que los conjuntos de relaciones (junto con sus conjuntos de entidades asociados) se tratan como conjuntos de entidades de nivel más alto, y pueden participar en las relaciones.

Las diferentes características del modelo E-R ofrecen al diseñador de bases de datos numerosas decisiones de cómo representar mejor la empresa que se modela. Los conceptos y objetos pueden, en ciertos casos, representarse mediante entidades, relaciones o atributos. Ciertos aspectos de la estructura global de la empresa se pueden describir mejor usando conjuntos de entidades débiles, generalización,

especialización o agregación. A menudo el diseñador debe sopesar las ventajas de un modelo simple y compacto frente a otros más precisos pero más completos.

Una base de datos que se representa en un diagrama E-R se puede representar mediante una colección de tablas. Para cada conjunto de entidades y para cada conjunto de relaciones de la base de datos hay una única tabla a la que se le asigna el nombre del conjunto de entidades o del conjunto de relaciones correspondiente. Cada tabla tiene un número de columnas, cada una de las cuales tiene un nombre único. La conversión de una representación de base de datos en un diagrama E-R a un formato de tabla se basa en la derivación de un diseño de bases de datos relacional desde un diagrama E-R.

El proceso de normalización de una base de datos consiste en aplicar una serie de reglas a las relaciones obtenidas tras el paso del modelo E-R (entidad-relación) al modelo relacional.

La primera forma normal, impone un requisito muy elemental a las relaciones; a diferencia de las demás formas normales, no exige información adicional como las dependencias funcionales. Una relación está en segunda forma normal (2FN) solamente si todos los atributos son dependientes en forma completa de la clave.

PARTE SEGUNDA:

CONSTRUCCIóN DE BASE DE DATOS

Objetivo: Al finalizar el alumno será capaz de construir Bases de Datos apegado a los criterios inicialmente estudiados.

CAPÍTULO 6: ACCESS 2003

6.1. INTRODUCCIÓN

6.1.1. ¿Qué es una base de datos?

Una biblioteca ha de mantener listas de los libros que posee, de los usuarios que tiene, una clínica, de sus pacientes y médicos, una empresa, de sus productos, ventas y empleados. A este tipo de información se le llama datos.

Un gestor de base de datos es un programa que permite introducir y almacenar datos, ordenarlos y manipularlos. Organizarlos de manera significativa para que se pueda obtener información no visible como totales, tendencias o relaciones de otro tipo. Debe permitir en principio:

- Introducir datos
- Almacenar datos
- Recuperar datos y trabajar con ellos

Todo esto se puede hacer con una caja de zapatos, lápiz y papel; pero a medida que la cantidad de datos aumenta, han de cambiar las herramientas. Se pueden usar carpetas, archivadores…, pero en un momento dado es conveniente acudir a los ordenadores, aunque las operaciones siguen siendo las mismas.

6.1.1.1. Tabla o fichero, registro y campo

Un programa de base de datos almacena la información que introducimos en forma de tablas como las que podemos ver, por ejemplo, en un directorio telefónico:

En este directorio nos interesa tener almacenados de modo ordenado los datos de una serie de personas. Para que aparezcan de modo claro los hemos desglosado en tres apartados: Nombre, Dirección y Teléfono, haciendo que aparezca cada uno en una columna diferente. Así es mucho más sencillo encontrar la dirección de una persona buscando a partir de su nombre.

Aquí podemos ver cómo la información referida a una persona, "un dato", aparece en una fila de la tabla: a esto es a lo que se denomina Registro. A cada una de las partes en las que hemos desglosado la información se le denomina Campo, y al conjunto formado por todos los registros, Tabla.

Registro: es el concepto básico en el almacenamiento de datos. El registro agrupa la información asociada a un elemento de un conjunto y está compuesto por campos.

Tabla: conjunto de registros homogéneos con la misma estructura.

Tenemos entonces lo siguiente:

6.1.1.2. Tipos de campos

En el directorio telefónico podemos ver que hay unos campos más importantes que otros: así el Nombre es fundamental para que el registro tenga sentido. Sería absurdo que apareciera una dirección en el directorio sin ir acompañado de un nombre. Por este motivo se suelen denominar campos fundamentales a aquellos que definen al registro, y campos secundarios a los que lo complementan.

6.1.2. Tipos de base de datos: planas y relacionales

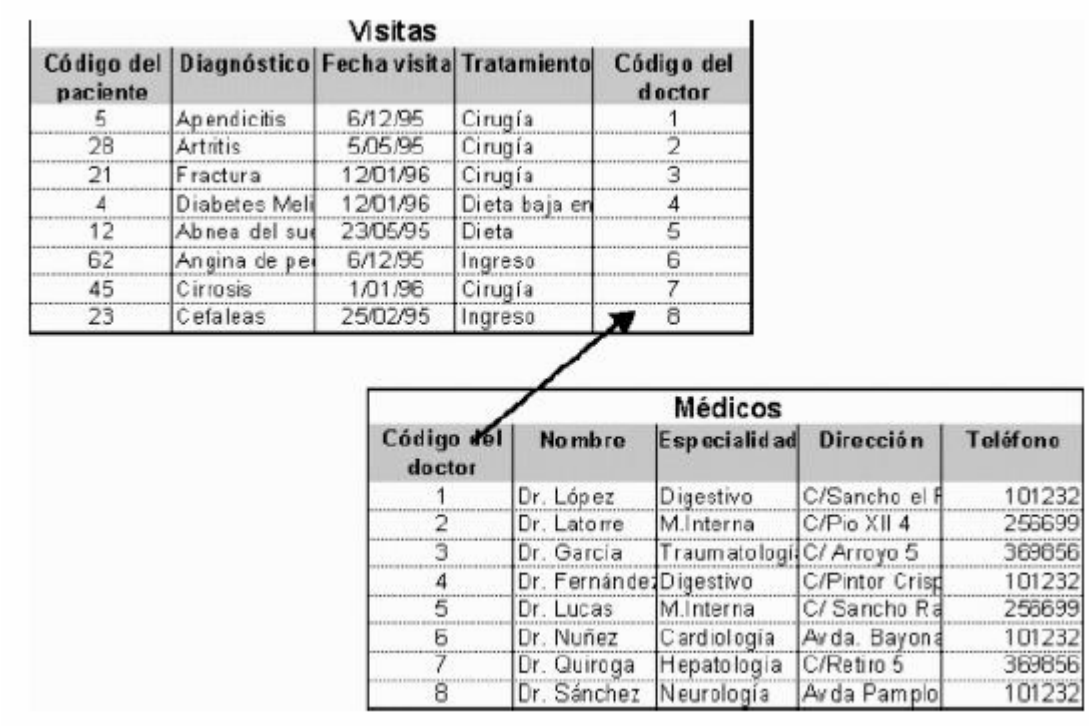
Para hacer una base de datos que cumpla las funciones de directorio telefónico necesitamos una sola tabla, pero puede haber casos en los que necesitemos más de una.

Un hospital, por ejemplo, necesitará almacenar más datos además del nombre y dirección de sus pacientes. Tendrá que llevar, a parte de otras muchas cosas, constancia de las visitas que un paciente haga al hospital. ¿Es posible almacenar esta información en una sola tabla?:

Hospital						
Fecha	Nombre	Dirección	Tfno.	Diagnóstico	Tratamiento	Médico
6-12-95	Cabrera Ortiz, Pedro	C/Mayor 12 4D	101232	Apendicitis	Cirugía	Dra. Sanz
5-5-95	García García, Ana	Avda. Arroyos, 54	256899	Gripe	Frenadol	Dr. Pérez
12-1-96	Santos Gemio, Luis	c/ Berruguete, 74	369856	Sarampión	Augmentine	Dr. Pérez
12-1-96	Cabrera Ortiz, Pedro	C/Mayor 12 4D	101232	Sinusitis	Sinus	Dr. Alonso
23-5-95	García García, Ana	Avda. Arroyos, 54	256899	Sarampión	Clamoxil	Dra. Sanz
6-12-95	Cabrera Ortiz, Pedro	C/Mayor 12 4D	101232	Sinusitis	Sinus	Dr. Pérez
1-1-96	Santos Gemio, Luis	c/ Berruguete, 74	369856	Amigdaltis	Clamoxil	Dr. Alonso
25-2-95	Cabrera Ortiz, Pedro	C/Mayor 12 4D	101232	Amigdaltis	Clamoxil	Dra. Sanz

Esta tabla contiene de modo abreviado los campos que interesan en una base de datos de este tipo. Pero se plantea un problema: si cada vez que viene un paciente al médico se le tiene que abrir una ficha, en poco tiempo los datos personales del paciente (dirección y teléfono) estarán repetidos muchas veces. Si esto se multiplica por todos los pacientes la base de datos contendrá gran cantidad de datos redundantes innecesarios, lo que aumentará su tamaño y la hará menos eficaz.

Para hacer una base de datos como ésta se usarán necesariamente varias tablas que luego se relacionarán por un campo común en ambas:



Visitas

Código del paciente	Diagnóstico	Fecha visita	Tratamiento	Código del doctor
5	Apendicitis	6/12/95	Cirugía	1
28	Artritis	5/05/95	Cirugía	2
21	Fractura	12/01/96	Cirugía	3
4	Diabetes Meli	12/01/96	Dieta baja en	4
12	Abnea del sue	23/05/95	Dieta	5
62	Angina de pe	6/12/95	Ingreso	6
45	Cirrosis	1/01/96	Cirugía	7
23	Cefaleas	25/02/95	Ingreso	8

Médicos

Código del doctor	Nombre	Especialidad	Dirección	Teléfono
1	Dr. López	Digestivo	C/Sancho el F	101232
2	Dr. Latorre	M.Interna	C/Pio XII 4	256699
3	Dr. García	Traumatologi	C/ Arroyo 5	369856
4	Dr. Fernández	Digestivo	C/Pintor Crisp	101232
5	Dr. Lucas	M.Interna	C/ Sancho Ra	256699
6	Dr. Nuñez	Cardiologia	Avda. Bayona	101232
7	Dr. Quiroga	Hepatologia	C/Retiro 5	369856
8	Dr. Sánchez	Neurología	Avda Pamplo	101232

De esta manera se consigue que no haya datos repetidos. Con esta estructura cada vez que venga un paciente, no es necesario volver a introducir sus datos personales. Basta con introducir su código para que el Gestor de base de datos sepa

de que paciente se trata. Para que se pueda establecer esta relación es necesario que las dos tablas contengan un campo en común (en este caso el código del médico).

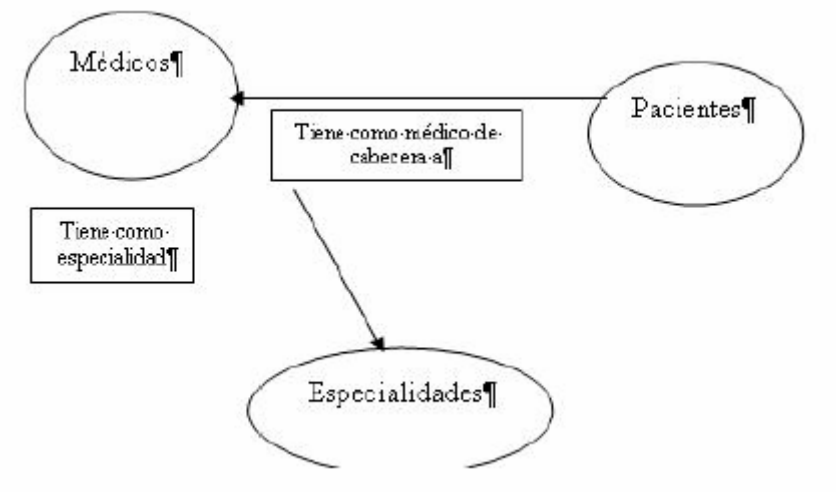
A esta forma de organizar la base de datos mediante distintas tablas relacionadas por campos comunes se le llama base de datos relacional. Cuando se utiliza solamente una tabla hablamos de una base de datos plana.

No todos los programas de gestión de base de datos tienen esta capacidad de manejar bases de datos relacionales, por eso, antes de elegir uno deberemos considerar si necesitamos o no esta capacidad.

Generalmente todos los programas de base de datos de propósito múltiple de última generación tienen capacidad relacional. Algunos de los más conocidos son Oracle, Fox, Access, FileMaker, 4D, Butler…

6.1.2.1. Ejemplo de base de datos relacional

Otra base de datos posible para un hospital sería ésta: guardar sólo información sobre los pacientes, los doctores y las especialidades. Asumiendo que cada médico tiene sólo una especialidad, y que cada paciente tiene asignado un médico de cabecera la estructura de la base de datos sería ésta:



Se observa que existen relaciones entre distintos objetos de la realidad. Estas relaciones deben respetarse para luego poder recuperar información de manera precisa y rápida.

Esta base de datos relacional estará formada por tablas. Con la característica de que las mismas se relacionan entre sí mediante uno o más campos. Se puede decir que cada objeto de la realidad será una tabla en nuestra base de datos y que hay que buscar la manera de reflejar las relaciones antes mencionadas. Así, para este ejemplo, se pueden emplear las siguientes tablas:

Cada tabla está compuesta por registros, y en este caso, cada registro contendrá la información pertinente de: un paciente, un doctor o una especialidad.

A su vez, cada registro está formado por diferentes campos, por ejemplo, para la tabla pacientes tendremos los siguientes campos: Nombre, Apellidos, Dirección, Teléfono, Identificador. A cada campo se le asociará un tipo de dato de acuerdo con lo que se quiera guardar en él, como se verá más adelante.

6.1.3. Algunas consideraciones sobre diseño de bases de datos

Antes de ver lo que es el programa en sí es importante que se tenga claro qué pasos hay que seguir al diseñar una base datos.

1 Es importante conocer exactamente para qué se quiere usar la base de datos, qué datos son los que interesan de los que existen en la realidad y qué información se necesitará extraer.

2 Una vez que esto esté claro, se definen las Tablas que compondrán la base de datos. Dentro de cada tabla, se piensa qué campos serán necesarios. Conviene detenerse y definir correctamente la base de datos, ya que un mal diseño hará que el sistema sea lento y los resultados no sean los esperados.

6.1.4. Bases de datos de red

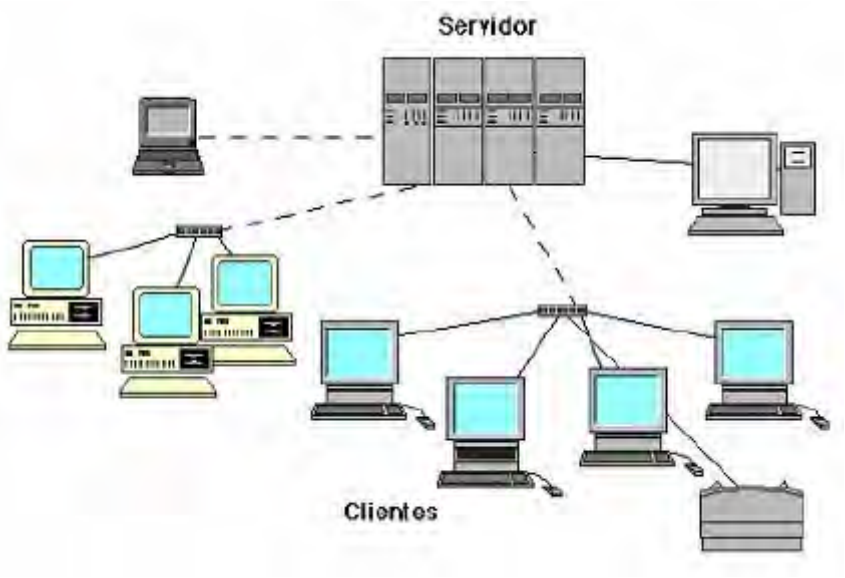
El uso de una base de datos de un directorio telefónico personal es muy distinto del uso de una base datos de un hospital, una empresa o un banco.

El directorio telefónico sólo lo utilizará una persona cada vez, mientras que las otras bases de datos necesitarán ser consultadas al mismo tiempo por muchas personas desde distintos sitios.

En la base de datos de un hospital muchas personas pueden necesitar acceder a los datos de un paciente al mismo tiempo: una enfermera en una planta para conocer

la dosis a suministrar de los medicamentos; el médico para estudiar el caso de ese paciente; y desde administración necesitarán los datos sobre ese paciente para elaborar el coste de su hospitalización. Todos ellos necesitarán por tanto hacer consultas o introducir nuevos datos.

Esto sería imposible si la base de datos estuviera situada en un ordenador al que no se puede acceder más que sentándose delante. Si se pusieran en varios sitios ordenadores con bases de datos iguales, al final del día y tras las operaciones que se hayan realizado, una base de datos ya no tendría nada que ver con otra y cualquier consulta posterior a cualquiera de ellas sería del todo inviable.



Para este tipo de bases de datos con múltiples usuarios aparecieron las llamadas bases de datos de red. Estas están situadas en un único ordenador -llamado servidor (generalmente ordenadores de gran potencia)- y se puede acceder a ellas desde terminales u ordenadores con un programa que permita el acceso a ella -los llamados clientes-. Los Gestores de bases de datos de este tipo permiten que varios usuarios hagan operaciones sobre ella al mismo tiempo: uno puede hacer una consulta al mismo tiempo que otro, situado en un lugar diferente, está introduciendo datos en la base.

Gestores de este tipo son: Oracle, PL4, DB2 o SQL Server, que está pensados únicamente para este uso y no se emplean para bases de datos personales.

FileMaker y Access, originariamente pensados para uso personal, tienen capacidades de red que hacen de ellos programas muy aptos para su empleo en bases de datos de pequeñas empresas, que no necesitan un número de accesos simultáneos muy alto.

6.1.5. Utilidad de una base de datos

Las tres cosas básicas que debe permitir un gestor de base de datos son: introducir datos, almacenarlos y recuperarlos.

Al mismo tiempo permiten otra serie de funciones que hacen de ellos herramientas incomparablemente superiores a los métodos tradicionales de almacenamiento de datos: archivadores, carpetas, etc.

Cualquier gestor debe permitir: ordenar los datos, realizar búsquedas, mostrar distintas vistas de los datos, realizar cálculos sobre ellos, resumirlos, generar informes a partir de ellos, importarlos y exportarlos.

6.1.5.1. Ordenar datos

Un directorio telefónico, para que sea útil debe estar ordenado por el orden alfabético de los nombres. Del mismo modo cualquier programa de base de datos debe permitir hacer lo mismo.

El orden en una base de datos puede ser alfabético, numérico, de fecha o de hora; por cualquier campo, y de modo ascendente o descendente. Así, se puede ordenar indistintamente la tabla de Visitas por la fecha de la visita, por los nombres de los pacientes o por el número de código. También se puede especificar varios criterios al mismo tiempo: por ejemplo, por orden alfabético de nombre y, para los casos en que un nombre coincida, por fecha de visita.

6.1.5.2. Búsquedas

En los anteriores sistemas de archivo de datos si se quería buscar un conjunto determinado de registros era necesario tener los datos ordenados previamente por un criterio determinado (por ejemplo, en los ficheros de biblioteca, por materia o autor). Luego visualmente y a mano, a menudo con gran trabajo y pérdida de tiempo, ir extrayendo los registros de uno en uno. Al terminar de usarlos se tenía que seguir el proceso contrario. En el caso de que se quisiera hacer una búsqueda por un criterio diferente al del orden del archivo (por editoriales en el ejemplo anterior) resultaba del todo imposible.

Cualquier programa de base de datos realiza búsquedas muy rápidas por cualquiera de los campos de la base, indistintamente del modo en que estén ordenados. Permiten hacer búsquedas con varios criterios distintos (de este autor y con fecha de publicación posterior a esta), búsquedas combinadas (de tal autor o de tal otro), contrarias (que no sean de este autor), etc. Deshacer la búsqueda es igual de rápido.

Una vez hecha la extracción, el programa nos permite realizar los mismos procesos que con el total de la base de datos (ordenaciones, informes…) pero ejecutados únicamente sobre los registros extraídos.

6.1.5.3. Formularios e informes

En las bases de datos, los datos se almacenan en forma de tablas. Esto no quiere decir que deban tener esta forma de presentación en la pantalla a la hora de introducir datos o extraerlos, ni que haya que imprimirlos así.

Aquí es donde entran los conceptos de formulario e informe. Ambos son similares, pero tienen una función diferente. Básicamente los formularios son presentaciones hechas para mostrar los datos en pantalla, mientras que los informes están pensados para imprimirlos.

Formularios

Cualquier programa de base de datos permite mostrar los datos en pantalla de modos muy diferentes. Esto tiene motivos funcionales y estéticos.

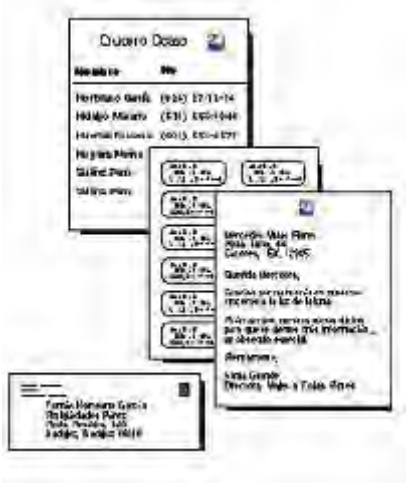
Los formularios permiten mostrar al mismo tiempo en la pantalla campos procedentes de distintas tablas relacionadas de forma que resulte mucho más sencillo trabajar con ellas. Al mismo tiempo se puede hacer que no aparezcan determinados campos.

Esto es fundamental tanto en las bases multiusuario como en la base de datos del ejemplo del hospital. En este tipo de bases de datos no interesará que todos los usuarios vean todos los datos. A una enfermera le interesa acceder a los datos médicos del paciente pero en modo alguno a los datos económicos que estarán almacenados en la misma base de datos. En el caso de una persona que trabaje en administración del hospital ocurre lo contrario: debe tener acceso a los datos económicos pero no a los médicos.

Los formularios, unidos a un acceso a la base de datos mediante usuarios y contraseñas, permiten solucionar este problema. Se puede hacer que cada usuario vea los datos que le interesan manteniendo ocultos los restantes.

Por otro lado los formularios permiten dar una apariencia más agradable a la presentación de los datos que hace que el trabajo con ellos sea más cómodo, permitiendo insertar datos, modificarlos, o eliminar algún registro.

Los informes son presentaciones de los datos preparadas para imprimir. Los gestores de base de datos tienen la capacidad de ir intercalando los datos de la base sobre textos con cualquier formato de tal forma que generan de modo automático cartas, etiquetas postales, listados.



6.1.5.4. Cálculos y sumarios

Los programas de bases de datos tienen la capacidad de realizar operaciones matemáticas sobre los registros. Así, por ejemplo, si se tiene almacenado en un campo de una tabla el salario de los empleados el programa puede calcular el salario menos impuestos de cada empleado. En la mayor parte de las bases de datos, los datos procedentes de los cálculos no quedan almacenados aumentando el tamaño de la base de datos, si no que sólo queda guardada la operación o fórmula.

Médico	Salario	Salario - IRPF
Dr. Sanz	250000	210000
Dr. Alonso	300000	252000
Dr. Sánchez	195000	163800
TOTAL: 3	**745000**	**625800**

Cálculo

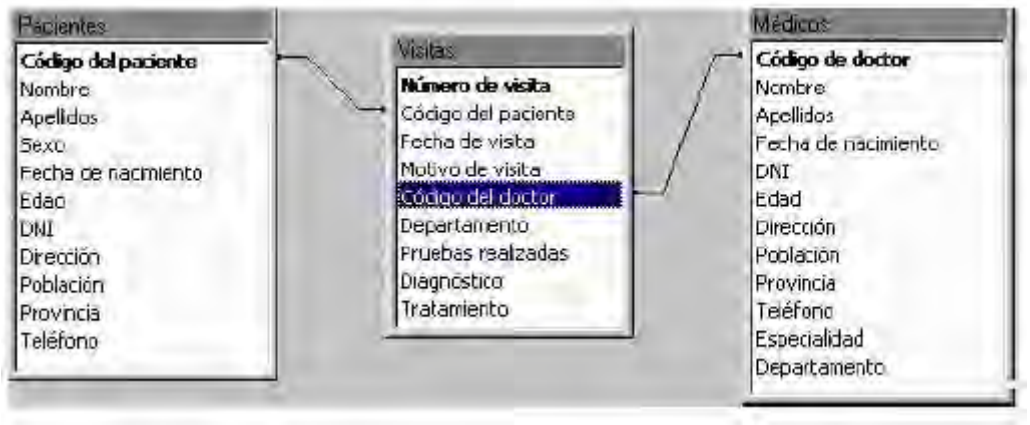
Sumario

También es posible realizar operaciones y cálculos estadísticos sobre el total de los registros: el programa puede calcular el número de médicos que trabajan en el hospital o en cada departamento, lo que han cobrado entre todos o desglosado por departamentos, etc. A este tipo de operaciones se les suele denominar sumarios porque son resultado de cálculos sobre grupos de registros.

6.1.6. Los ejemplos del manual

En el manual se utilizan varios ejemplos para explicar el funcionamiento de Access. Dos de ellos son de un hospital. Para mayor comprensión de los ejemplos a continuación se muestra la estructura de cada ejemplo de forma clara.

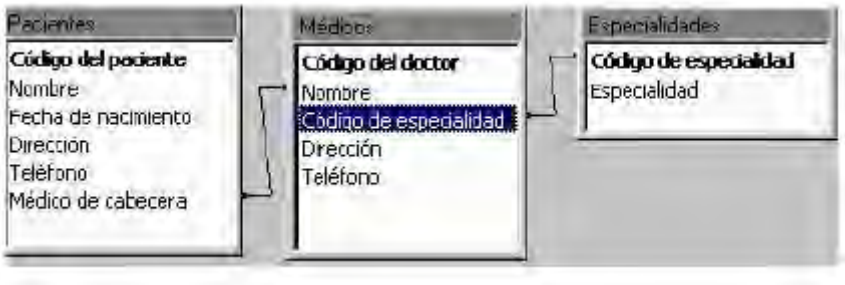
Ejemplo 1:



Un paciente puede acudir al médico muchas veces en la vida. En cada visita que realiza el paciente le puede atender un médico distinto por motivos distintos. Un médico a su vez atiende a muchos pacientes. En esta base de datos la relación entre los pacientes y los médicos se ha realizado creando una tabla: visitas.

Los campos en común son los códigos de los pacientes y de los médicos. Estos campos compartidos tienen el origen en la tabla que los creó (tabla médicos o tabla pacientes) pero esos mismos datos se podrán ver en la tabla visitas gracias a la relación. De esa forma los datos de una visita en parte procederán de las tablas médicos y pacientes, y en parte serán datos propios de visitas.

Ejemplo 2:



En esta base de datos la relación entre la tabla de los médicos y la de pacientes es más directa.

Hay una tercera tabla para tener un listado de las especialidades y no escribir el nombre mal.

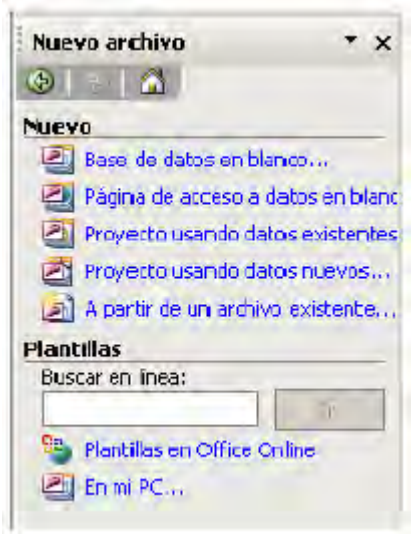
En este ejemplo cada médico tiene adjudicada una lista de pacientes. A cada paciente le corresponde un médico de cabecera. Por tanto el campo que permite la relación es el nombre del médico (Médicos) y médico de cabecera (Pacientes).

6.2. CREACIÓN DE UNA BASE DE DATOS CON ACCESS

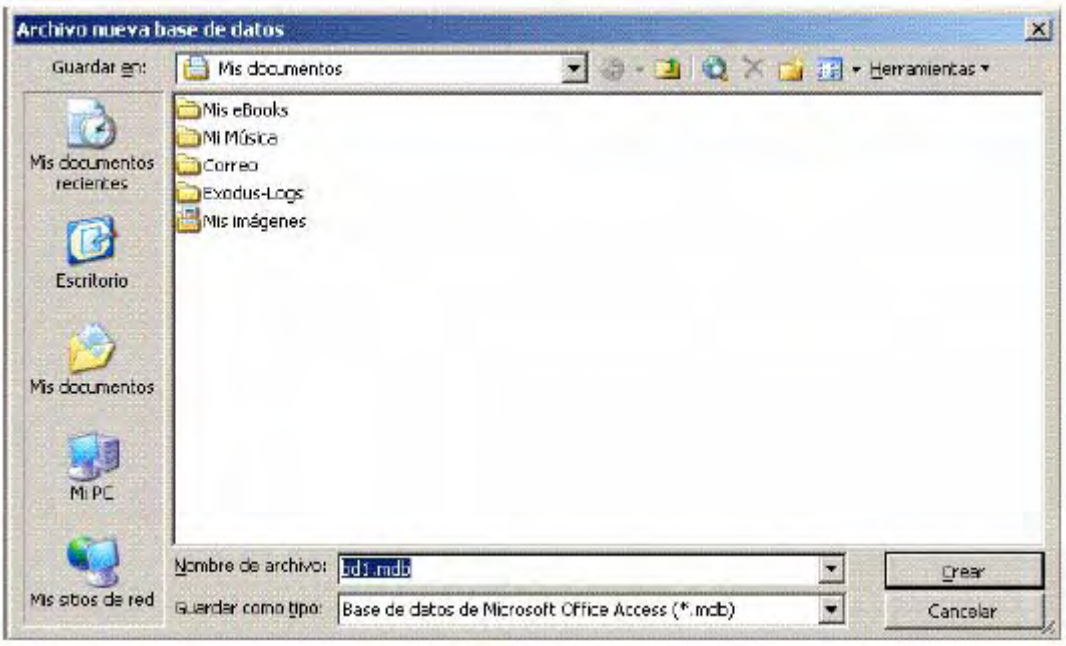
¿Cómo crear un fichero de base de datos?

Al iniciar el programa se presentará la siguiente ventana:

Se selecciona «Crear un archivo nuevo» y se hace doble clic. Aparecerá otro cuadro de diálogo en el que se podrá elegir entre crear una «Base de datos en blanco», una «página de acceso a datos en blanco», «abrir un archivo ya existente»…



Para crear un archivo nuevo pulsaremos sobre «Base de datos en blanco…», aparecerá otro cuadro de diálogo en el que se dará nombre a la base de datos que se va a crear. Se de utilizar un nombre apropiado y relacionado con el contenido de la base de datos, para poder recuperarla con facilidad posteriormente.



Se escribe el nombre en Nombre de archivo y se pulsa el botón «Crear». Aparecerá la ventana de Access.

Desde esta ventana se trabajan las bases de datos de Access. Seleccionando las pestañas se accede a los distintos elementos que componen una base de datos; tablas, consultas, formularios, informes, macros y módulos.

Para volver a esta ventana desde cualquier otra se pulsa el botón «  ».

En el apartado de cada elemento de la base de datos se explicará como crearlo desde la ventana de cada uno. Pero desde la ventana de base de datos hay dos formas de crear cualquier elemento sin necesidad de seleccionar la pestaña: «Tabla», «Consulta»....

«  » Este botón cambia dependiendo del último objeto creado. Al pulsarlo se creará otro objeto similar al último creado. Si se quiere elegir otro objeto sólo hay que hacer clic sobre la flecha de la derecha. Se desplegará el menú de todos los elementos de Access:

En este menú se encuentran todos los elementos que componen Access. Para crear uno de ellos, basta con situar el ratón por encima de él y hacer clic.

Otra forma de crear un elemento de Access sin seleccionar la pestaña es a través del menú Insertar.

6.2.1. Crear un nuevo fichero de base de datos

Si se está utilizando Access, con una base de datos abierta y se desea crear un nuevo fichero, se puede pulsar la tecla «  » o seleccionar «Archivo|Nuevo».

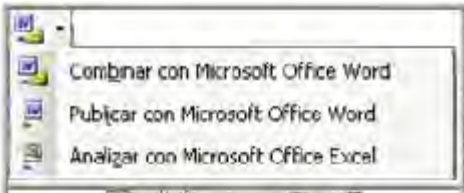
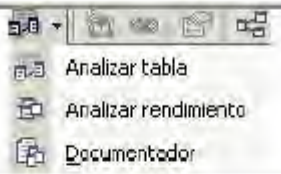
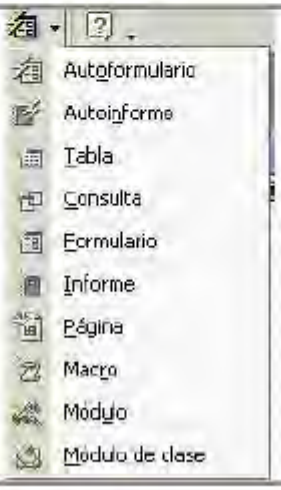
6.2.2. Los menús de Access

En la ventana de Access, como en todos los programas de Microsoft, hay un Menú y una Barra de herramientas. La diferencia de Access es que tiene 22 barras de herramientas distintas. Las barras de herramientas y el menú varían según el elemento que esté seleccionado: tablas, consultas, formularios, informes...

Nada más crear una base de datos, no todos los botones están activos. Irán variando según se vayan añadiendo elementos a la base de datos y según lo que se seleccione; tabla, consulta, formulario, informe, macros, módulos…

La barra de herramientas tiene los botones agrupados por la función que realizan, muchos de estos grupos se repiten en las diferentes barras. Estos son todos los botones que componen la barra de herramienta de la ventana general de Access. Según se vayan viendo los diferentes elementos de Access se verá que en sus barras aparecen muchos de estos elementos y otros nuevos:

	1°. Muestra las vistas disponibles 2°. Abre una base de datos ya existente 3°. Guarda la base de datos actual
	1°. Permite buscar archivos en el disco duro o en una unidad de red 2°. Imprime el elemento seleccionado 3°. Muestra la presentación preliminar de lo que se imprimirá
	1° Corrige la ortografía 2° Cortar 3°. Copiar 4° Pegar
	Deshacer: deshace la última acción realizada.

	Estos son los vínculos que tiene Access con el Office: Con el MS Word para combinar y publicar (correspondencia) Con Excel para analizar los datos
	Estas opciones analizan los datos
	Editor de secuencias de comandos en Microsoft
	Código
	Muestra las propiedades del elemento que esté seleccionado, la fecha de creación, y la última modificación
	1°. Crea nuevos elementos de la base de datos
	1°. Muestra las relaciones existentes entre las tablas o ayuda a crearlas 2° Llama al ayudante de Office

6.3. LAS TABLAS

Para empezar a trabajar con una base de datos primero es necesario crear las tablas. Dentro de cada una hay que definir los campos que contendrán la información.

6.3.1. ¿Cómo crear una tabla?

Para crear una tabla se selecciona la pestaña « 

» aparecerá la siguiente ventana:

Estas son las diferentes opciones que presenta Access para crear una tabla:

1 Vista Hoja de datos: crea una nueva tabla con formato de tabla. En la primera fila de la tabla aparecen los campos: Campo 1, Campo 2… sobre los cuales se escriben los nombres de los campos.

2 Vista Diseño: permite crear los campos manualmente y configurar el diseño de la tabla.

3 Asistente para tablas: el asistente pide las características de los campos y de la tabla y la genera automáticamente.

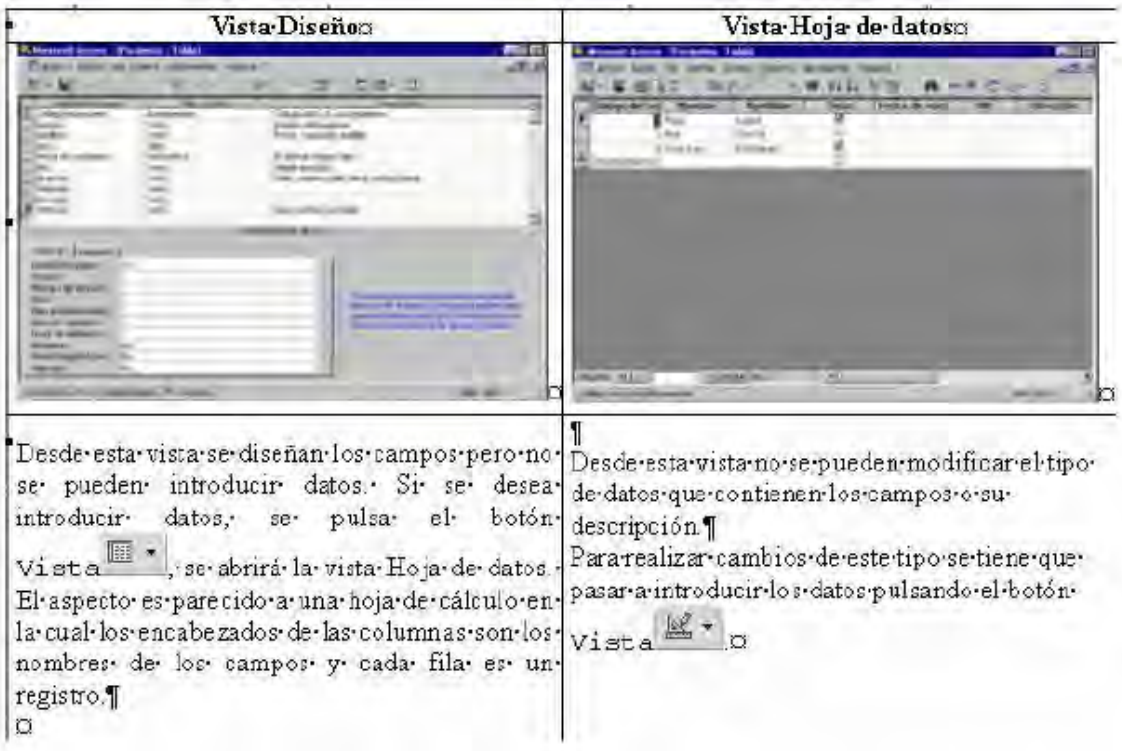
4 Importar tabla: esta opción permite importar datos de otra base de datos, que no necesariamente tiene que estar creada por Access.

5 Vincular tabla: crea vínculos entre las tablas importadas y las originales, las modificaciones que se efectúen en los datos se transmiten a aquéllas.

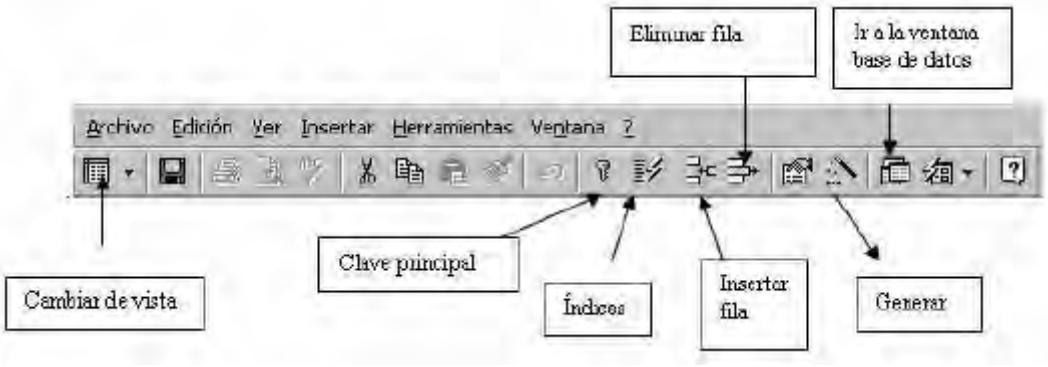
Cualquiera de estas opciones es válida para crear una base de datos. Pero las opciones a través de las cuales se crea personalmente una tabla son «Vista Hoja de datos» y «Vista Diseño». Con la opción Vista Diseño se crea la estructura para luego rellenar los datos en Vista Hoja de datos. Pero también se puede empezar directamente en Vista Hoja de datos introduciendo la información y Access crea la estructura automáticamente.

6.3.2. Las Vistas de la tabla

Las tablas se pueden ver desde dos vistas distintas, en cada una de ellas no sólo cambia el aspecto de la tabla, sino que además varían el menú y la barra de herramientas:



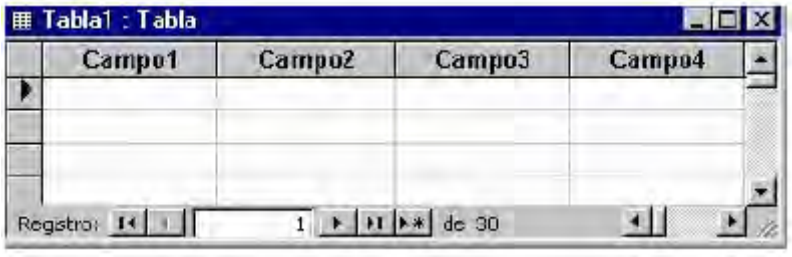
Esta es la barra de herramientas y el menú de Vista Diseño:



En la Vista Hoja de datos el menú añade todo lo relacionado con "Registros" ya que en esta vista se pueden introducir datos. En la barra de herramientas se añaden elementos de orden de los datos:

6.3.2.1. Vista Hoja de datos

Con esta opción es más fácil entender cómo se almacena la información en una base de datos. A continuación se va a crear la tabla Pacientes. Se selecciona «Vista Hoja de datos» y se pulsa «Aceptar».



En esta tabla ya se puede empezar a introducir datos, aunque antes conviene dar nombre a los campos. Para esto se hace doble clic sobre «Campo1» y se escribe el nombre que se le quiera dar al campo. En el caso de la tabla Pacientes el primer campo va a ser "Nombre".

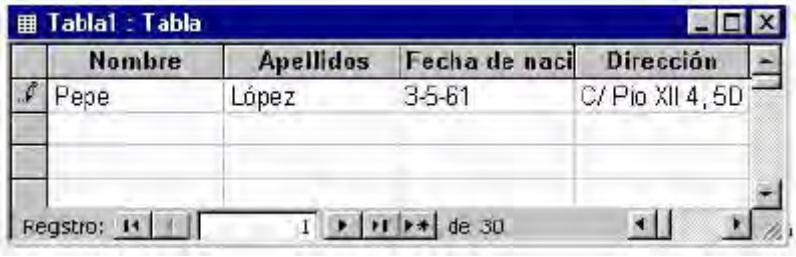
Los nombres de los campos tienen que cumplir unas normas. No pueden tener más de 64 caracteres, no pueden tener puntos, exclamaciones o corchetes.

Por otro lado no puede haber dos campos que se llamen de la misma manera.

Para que Access cree la estructura de la tabla basta con introducir los nombres de los campos y un registro.

Para escribir el primer registro se sitúa el ratón sobre el campo y se escribe el primer dato. Para pasar a la siguiente columna se puede usar el ratón o la tecla tabulador.

Siguiendo el mismo procedimiento, doble clic sobre la primera fila, se añaden los campos: apellidos, fecha de nacimiento y dirección.



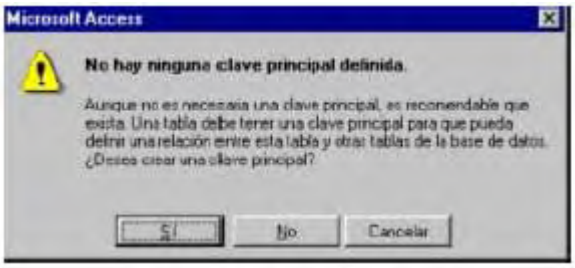
Como aún no se han definido las características de los campos, es conveniente hacerlo antes de introducir más datos, esto se hace en el modo de vista diseño.

6.3.2.2. Vista Diseño

Se elige «Vista Diseño» y se pulsa «Aceptar». Access pedirá que se le dé un nombre a la tabla.

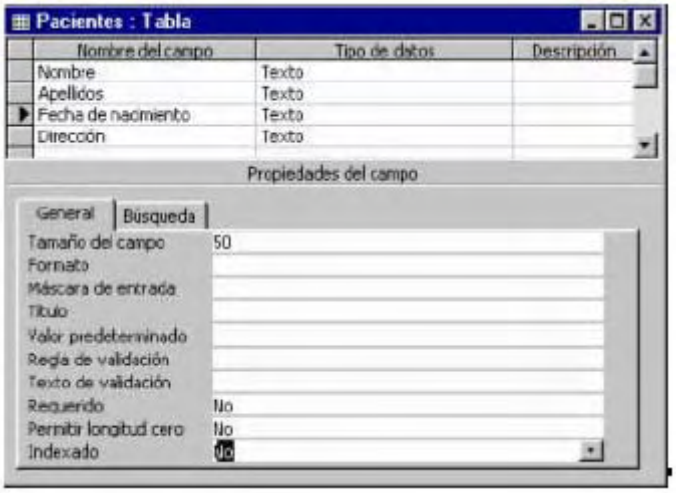
Se escribe el nombre: Pacientes en el recuadro y se pulsa «Aceptar».

A continuación aparecerá otro mensaje comunicando que no se ha creado una clave principal.



Por el momento se pulsa «No», ya que no se va a crear ahora, se verá más adelante en este manual.

Después de pulsar «No», se abrirá la ventana de diseño de la tabla ya creada con el tipo de dato definido por Access de forma automática.



En la primera columna se ven los nombres de los campos, y en la segunda columna el tipo de datos que ha elegido automáticamente Access con los datos introducidos. Como se ve en cada columna se introducen diferentes características del campo:

- Nombre del campo: los nombres de los campos.
- Tipo de datos: texto, numérico, fecha/hora, contador, Si/No, memo, moneda, objeto OLE.
- Descripción: en esta columna se introduce la descripción del contenido del campo o su finalidad.
- Propiedades de los campos: estableciendo las propiedades de los campos se controla la apariencia de los datos, y se puede evitar que se introduzcan de modo incorrecto.

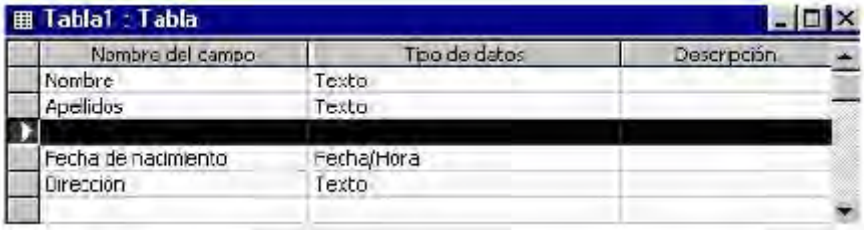
En este manual se va a crear una base de datos de un hospital. Se ha empezado con la tabla de los pacientes pero no se han introducido todos los campos. Faltan los campos: sexo, población, provincia, teléfono, DNI.

El tipo de dato se verá mas adelante en el apartado campos, junto con la descripción y las propiedades del campo. Por ahora se introduce el nombre del campo y se selecciona el tipo de dato tal y como aparecen en este gráfico:

6.3.2.3. Insertar un campo

El campo Sexo se encuentra entre Apellidos y Fecha de Nacimiento. Para introducir un campo nuevo entre dos ya existentes pulsa la tecla «  » .El campo insertado aparecerá sobre el campo que estaba seleccionado cuando se pulsó la tecla insertar campo en este caso, estaba seleccionado Fecha de nacimiento, por lo tanto el nuevo campo se creará encima de él y se le pondrá el nombre Sexo.

Para introducir el nombre del campo Sexo se tiene que seleccionar el campo Fecha de nacimiento, y pulsar insertar campo:



6.3.2.4. Introducir el nombre del campo

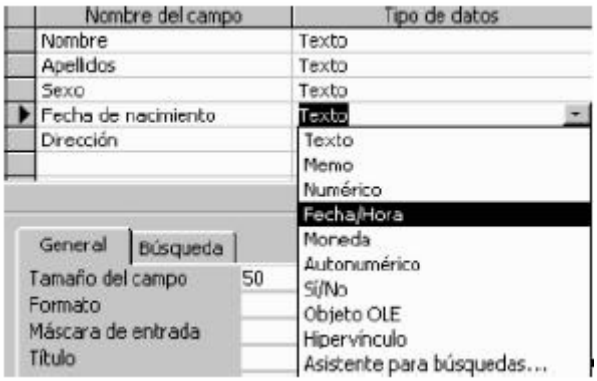
Para introducir el nombre se hace clic sobre la celda correspondiente y se teclea el nombre que se le vaya a dar al campo.

6.3.2.5. Elegir el tipo de dato

Mientras en el Vista Hoja de datos no se introduzcan datos Access asigna a todos los campos el tipo de dato: texto. Si se hubiera introducido un dato numérico el tipo de dato sería numérico.

Para cambiar el tipo de dato, basta con situarse sobre el campo. En la celda de tipo de datos aparecerá una flecha. Al pulsar sobre ella se despliega un menú en el que se puede seleccionar el «tipo de dato».

El «tipo de dato» que se le va a dar al campo Fecha de Nacimiento es «Fecha/hora». Para cambiar el tipo de dato se pulsa sobre esa celda, en la parte derecha de la celda aparece una flecha, que al ser pulsada despliega un menú. Para seleccionar el tipo de dato que interese hay que situarse sobre él con el ratón.



Para añadir los campos: población, provincia, teléfono, DNI y edad basta con situar el ratón sobre la fila en blanco que sigue a Dirección.

6.3.2.6. Mover un campo

A continuación se va a mover el campo DNI y se va a situar debajo de Fecha de nacimiento. Para mover el campo, primero se debe seleccionar haciendo clic sobre él. El campo cambia de color y a su izquierda aparece una flecha. Situando el cursor sobre esta flecha el cursor mismo se convierte en otra flecha. Si se pulsa en ese momento el botón del ratón se puede arrastrar el campo a la posición que se quiera. De modo que la tabla de pacientes quedará finalmente así:

Nombre del campo	Tipo de datos	Descripción
Nombre	Texto	Nombre del paciente
Apellidos	Texto	Primer y segundo apellido
Sexo	Texto	Hombre/mujer
Fecha de nacimiento	Fecha/Hora	No olvidar ningun dato
DNI	Texto	Añadir letra NIF
Dirección	Texto	Calle, numero, piso, letra, codigo postal
Población	Texto	
Provincia	Texto	
Teléfono	Texto	Incluir prefijo provincial

6.3.3. La clave principal

La clave principal suele ser uno o varios de los campos de la tabla. El contenido de este campo identifica cada registro del campo de manera única. De modo que no se podrán introducir dos registros iguales o almacenar valores nulos en los campos de la clave principal.

Para la tabla Pacientes se tiene que pensar que campo no se repite. Podría ser el campo nombre, pero el nombre no es algo único. Los campos nombre y apellidos juntos también se podrían repetir en algún caso. De modo que lo más indicado es crear un código único para cada paciente. Se selecciona el campo nombre y se inserta un campo. Se llama Código del paciente y se elige el tipo de dato Autonumérico. Este tipo de dato hace que Access genere un número único a cada registro de la tabla. De esta forma es totalmente seguro que el campo no tendrá ningún registro repetido.

Es decir los datos de un paciente no aparecerán repartidos en tres veces, sino en una sola vez, de forma que cuando se quiera consultar el estado físico de un paciente se tendrá la seguridad de que ahí están todos sus datos médicos.

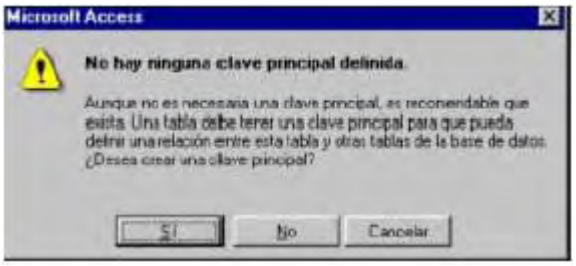
Nombre del campo	Tipo de datos	Descripción
Código del paciente	Autonumérico	Código único de cada paciente
Nombre	Texto	Nombre del paciente
Apellidos	Texto	Primer y segundo apellido
Sexo	Texto	Hombre/mujer
Fecha de nacimiento	Fecha/Hora	No olvidar ningun dato
DNI	Texto	Añadir letra NIF
Dirección	Texto	Calle, numero, piso, letra, codigo postal
Población	Texto	
Provincia	Texto	
Telefono	Texto	Incluir prefijo provincial

Para establecer este campo como clave principal se hace clic sobre él y en la barra de herramientas se pulsa el botón «Establecer Clave Principal» «  ». También se puede realizar esta operación desde el «Menú Edición | Clave Principal» o pulsando con el botón derecho sobre el campo que queremos que sea la clave y aparecerá un menú desplegable y seleccionamos clave principal.

Al establecer un campo como clave principal aparecerá una llave a la izquierda del campo, tal y como muestra la imagen:

Nombre del campo	Tipo de datos	Descripción
Código del paciente	Autonumérico	Código unico de cada paciente
Nombre	Texto	Nombre del paciente
Apellidos	Texto	Primer y segundo apellido

No se tiene que definir obligatoriamente una clave principal, pero normalmente es conveniente hacerlo. Si no se establece la clave principal, al cerrar la tabla aparece un cuadro de diálogo pidiendo que se establezca:



Si se elige la opción Si, Access creará automáticamente un campo Autonumérico que será la clave principal.

6.3.3.1. Tipos de clave principal

En Microsoft Access existen tres tipos de clave principal: Autonumérico, Campo simple y Campos múltiples.

6.3.3.1.1. Claves principales de Autonumérico

Un campo Autonumérico puede establecerse para que el programa introduzca automáticamente un número secuencial cuando se agrega un registro a la tabla.

Designar un campo de este tipo como clave principal de una tabla es la forma más sencilla de crear una clave principal.

Cuando no se establece una clave principal antes de guardar una tabla recién creada, Microsoft Access pregunta si se desea que cree una clave principal automáticamente. Si se contesta afirmativamente, Microsoft Access creará una clave principal de Autonumérico.

6.3.3.1.2. Claves principales de Campo simple

Si se tiene un campo que contiene valores exclusivos, como números de identificación o números de pieza, se puede designar ese campo como la clave principal.

Si el campo seleccionado como clave principal tiene valores duplicados o Nulos, Microsoft Access no establece la clave principal.

Se puede ejecutar una Consulta de buscar duplicados con el fin de determinar qué registros contienen datos duplicados. Si no se puede eliminar fácilmente las entradas duplicadas mediante la edición de los datos, se puede agregar un campo Autonumérico y establecerlo como clave principal o bien definir una clave principal de campos múltiples.

6.3.3.1.3. Claves principales de Campos múltiples

En situaciones en las que no se puede garantizar la exclusividad de un solo campo, se pueden designar dos o más campos como clave principal.

La situación más común en la que surge este problema es en la tabla utilizada para relacionar otras dos tablas en una relación varios a varios.

Si no se está seguro de poder seleccionar una combinación de campos apropiada para una clave principal de campos múltiples, probablemente resultará más conveniente agregar un campo Autonumérico y designarlo como la clave principal en su lugar.

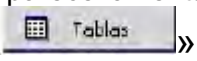
6.3.4. Guardar la tabla

Una vez haya definido la estructura de la tabla se pulsa el botón de «Guardar» «  » o se elige «Archivo|Guardar». El nombre que se le va a asignar a esta tabla es Pacientes. Conviene guardar la tabla siempre que se realiza algún cambio.

6.3.5. Abrir y trabajar una tabla

Para abrir una tabla de una base de datos ya existente, primero se tendrá que direccionar a esa base de datos.

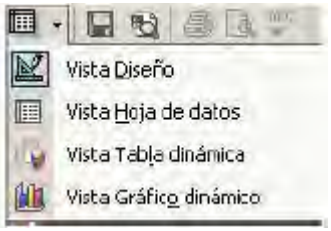
Para abrir una base de datos, se selecciona «Archivo|Abrir», o se pulsa el botón «  ». Aparecerá una ventana de diálogo con las diferentes bases de datos, se selecciona la que interese y se pulsa «Abrir».

Una vez abierta la base de datos, siempre aparece la ventana de la base de datos, con múltiples pestañas. Se elige la pestaña « 

» y se hace clic sobre la tabla que se desea abrir.

Se pulsa el botón «Abrir», si se desea introducir datos, o Diseño si se desea añadir un campo o variar las propiedades de los campos.

Una vez se encuentra dentro de una de estas dos opciones siempre se tiene la oportunidad de cambiar pulsando el botón cambiar de vistas:



6.3.5.1. Introducir datos en la tabla

En Vista Hoja de Datos se pueden introducir datos. En el primer registro sólo aparecerá una fila. Se hace clic sobre ella y se escriben los datos en los campos. En el momento en que se empiece a escribir se añadirá una fila más. En la primera columna de la fila sobre la que esté escribiendo aparecerá un lápiz, y en la siguiente un asterisco. Para pasar de un campo a otro pulse «Intro» o «Tabulador».

	Código del pa	Nombre	Apellidos
	1	Pepe	López
	2	Ana	Garcia
	3	Jose Luis	Rodríguez
*	(Autonumérico)		

Cuando se quiere añadir otro registro sólo se tiene que pinchar sobre la fila con el asterisco.

6.4. LOS CAMPOS

Para crear los campos de una manera más completa es necesario profundizar en cada una de las características de un campo.

6.4.1. Nombre del campo

En esta columna se introduce el nombre de los campos. La columna tiene un máximo de 64 caracteres. Se pueden utilizar espacios, pero no se puede empezar con un espacio. No se pueden utilizar: los puntos, los signos de admiración, los acentos graves ni los corchetes.

Dentro de una misma tabla no puede haber dos campos con el mismo nombre.

El nombre del campo debe ser descriptivo de la información que el campo va a contener para no crear confusión a la hora de trabajar con los datos.

6.4.2. Tipo de datos

1. Texto: almacena cadenas de caracteres, ya sean números (con los que no se vaya a realizar operaciones), letras o cualquier símbolo.

2. Numérico: Almacena números destinados a realizar operaciones. Hay cinco tamaños:

   - Byte: para almacenar el rango de 0 a 255
   - Entero: para el rango entre -32768 y 32767
   - Entero Largo: para el rango entre-2.147.483.648 y 2.147.483.647
   - Simple: para números decimales entre el -3,4x 1038 y el 3,4x 1038 con 7 decimales
   - Doble: Doble para números entre el -1,797x 1038 con 15 lugares decimales.

3. Fecha/hora: fecha y hora general, fecha y hora larga, fecha y hora corta.

4. Autonumérico: Es un valor numérico que Access incrementa de modo automático cada vez que se añade un registro. No se puede modificar manualmente.

5. Si/No: Para almacenar datos que sólo tengan dos posibilidades: si-no, 0-1, verdadero-falso, blanco-negro...

6. Memo: Para almacenar texto largo, hasta de 64000 bytes.

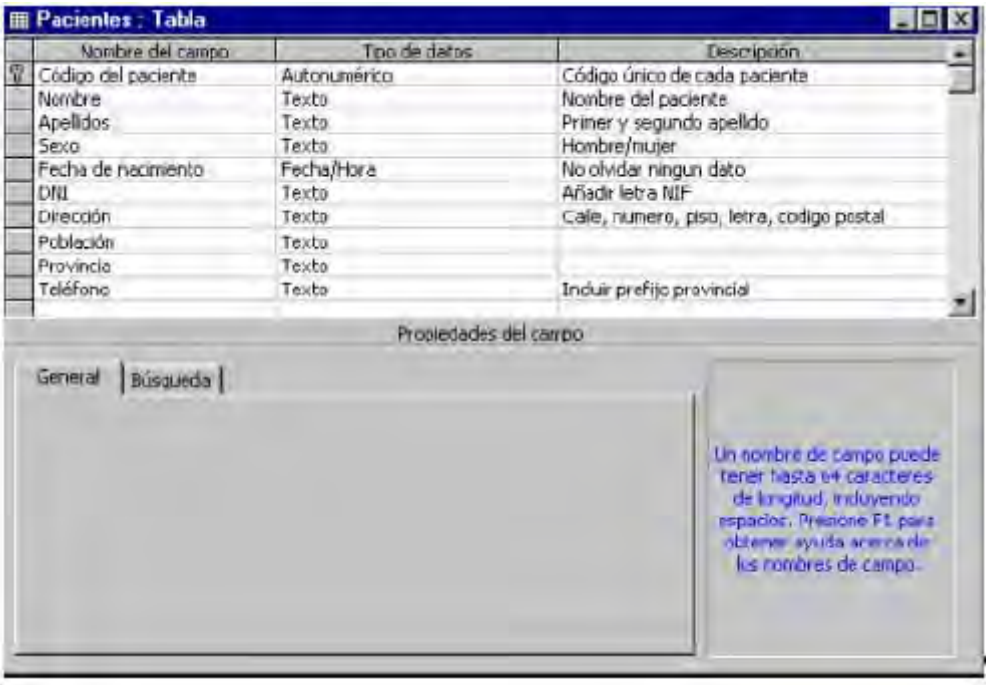
7. Moneda: Para almacenar valores de moneda.

8. Objeto OLE: Son objetos tales como gráficos, texto, imágenes, creados en otras aplicaciones, que se han incrustado o vinculado.

6.4.3. Descripción

En esta columna se introduce la descripción del contenido del campo o su finalidad. El texto que se introduzca aparecerá en la barra de estado cuando se seleccione el campo en un formulario, de este modo se facilita el introducir el contenido adecuado en cada campo.

En la tabla pacientes, que se ha creado en el apartado anterior, la descripción podría ser la siguiente:



Las propiedades de los campos ayudan a detallar el tipo de dato que va a contener un campo, y por lo tanto la utilidad del mismo dentro de la base de datos. Las propiedades se explican en el capítulo Propiedades de los campos.

A continuación se crean estas dos tablas para poder continuar con el ejemplo de la base de datos de un hospital:

La clave principal de la tabla Médicos es el código del médico y de la tabla visita, el número de visita.

6.5. PROPIEDADES DE LOS CAMPOS

Estableciendo las propiedades de los campos en Vista Diseño se controla la apariencia de los datos, y se puede evitar que se introduzcan de modo incorrecto.

6.5.1. Tamaño del campo

Esta propiedad la pueden tener tanto los campos de tipo numérico como los de texto.

En los campos de texto el tamaño del campo indica el número de caracteres que puede contener. El valor máximo es de 255 caracteres y por defecto Access aplica 50. Se puede introducir un valor inferior, pero si ya se han introducido datos hay que tener mucho cuidado, ya que si se da un tamaño inferior al de algún campo los caracteres restantes se perderán.

En los de tipo numérico limita el rango de valores permitido y si el campo puede contener valores fraccionarios. El campo numérico puede tener estos tamaños: Byte, Entero, Entero Largo, Simple y Doble.

Configuración	Rango	Lugares Decimales	Lugar que ocupa
Byte	0 a 255	Ninguno	1 byte
Entero	-32.768 a +32.768	Ninguno	2 bytes
Entero Largo	-2.147.483.648 a + 2.147.483.647	Ninguno	4 bytes
Simple	$-3,4x10^{38}$ a $3,4x10^{38}$	7	4 bytes
Doble	$-1.797x10^{38a}$ $1,797 x10^{38}$	15	8 bytes

El Byte es el que menos tamaño tiene y por tanto el que menos ocupa. El Doble es el que más ocupa. No conviene que el tamaño sea mayor de lo necesario, ya que cuanto más ocupe un campo más lento se procesaran los datos cuando se esté trabajando.

6.5.2. Formato

Esta propiedad la pueden tener todos los campos menos los Objetos OLE.

El formato sólo afecta a la presentación de los datos, nunca al valor almacenado de una tabla. Los números y las fechas se pueden presentar con diferentes formatos.

Los números pueden aparecer con separador de miles, con un símbolo de moneda o con un determinado número de decimales.

Si no se determina nada en esta propiedad Access utiliza el formato General y los datos aparecen tal y como se hayan introducido.

El formato que se especifique para un campo de una tabla será el que Access utilice para los formularios e informes basados en dicha tabla. Si se realiza un cambio

de formato después de haber creado un formulario o un informe, este formato nuevo no le afectará.

Para garantizar la coherencia entre las distintas aplicaciones, Access utiliza los valores establecidos para los formatos de número y de fecha/hora en la sección internacional del Panel de Control de Windows.

6.5.2.1. Formatos de campos de tipo Numérico y Moneda.

Si no especifica ningún formato, o si especifica el formato número general, Access presentará los números sin separador de miles ni ningún otro tipo de formato.

Si se desea que los números de un campo aparezcan con algún formato determinado se presiona la flecha de la derecha en Formato, aparecen dos columnas: la izquierda indica el nombre del tipo de formato y la derecha el resultado de aplicar ese formato a un número.

6.5.2.2. Lugares decimales

Cuando se selecciona un formato de número que no sea número general en esta propiedad se puede especificar un número de lugares decimales exacto. Si se ha escogido el formato número general sólo se presentará el número de lugares decimales necesarios para cada valor.

6.5.2.3. Formatos personalizados de campos numéricos

Aunque en la propiedad formato se puede elegir un formato ya creado de la lista que se despliega, Access también permite establecer un formato propio. Este formato se crean a través de códigos que Access convierte en formatos.

Un tipo de formato se crea con cuatro secciones:

1° Para los números positivos

2° Para los números negativos

3° Para los que tengan valor cero

4° Para los que no tengan ningún valor (que el campo esté vacío)

Código	Función
, (coma)	Separador decimal
. (punto)	Separador de miles
0	Muestra un dígito o el 0
#	Muestra un dígito o un espacio en blanco
%	Multiplica el valor por cien y lo muestra seguido del símbolo porcentaje
E- o e-	Notación científica con un signo menos junto a los exponentes negativos
E+ o e+	Notación científica con un signo mas junto a los exponentes positivos

Ejemplo de las cuatro secciones: #.##0,00 ; -#.##0,00; 0,00; "Vacío"

6.5.2.4. Códigos de formato para campos de cualquier tipo

Para crear formatos personalizados para cualquier tipo de campos se utilizan estos códigos:

Código	Función
Espacio	Muestra un espacio
"texto"	Muestra lo que esté entre comillas
!	Fuerza la alineación a la izquierda, en lugar de la alineación a la derecha
*	Rellena el espacio disponible con el carácter que venga a continuación.
\	Muestra el carácter que haya a continuación. Evita que ese carácter sea tomado por un código de control si coincide con alguno
[color]	Muestra la sección en el color indicado entre los corchetes. Los colores disponibles son: negro, azul, verde, cyan, rojo, magenta, amarillo, blanco

6.5.2.5. Formatos de campos de tipo Texto y Memo

En estos dos campos no hay ningún formato predefinido. Si se desea se puede crear uno propio. Al igual que en los campos numéricos un formato se crea en varias secciones con unos códigos.

En los campos de texto y memo sólo hay dos secciones, la primera se usa si el campo tiene texto y la segunda si el campo está vacío.

Los códigos son los siguientes:

Código	Función
@	Si los caracteres que se introducen en un campo no lo completan cada uno de los códigos que se inserten en formato se convertirá en un espacio o carácter para rellenar el campo
&	Funciona igual que el anterior, pero si no hay suficientes caracteres para sustituir todos los símbolos Acces no insertará nada
<	Presentará todos los caracteres en minúsculas
>	Presentará todos los caracteres en mayúsculas

6.5.2.6. Formatos de campos de tipo Fecha/Hora

Los formatos predefinidos de este campo dependen de la configuración de la sección internacional del panel de control de Windows.

En este tipo de campo también se pueden crear formatos personalizados. Con la diferencia de que sólo hay una sección:

Código	Significado
:	Separador de hora
/	Separador de fecha
D	Día del mes en uno o dos digitos numéricos (1-31)
dd	Día del mes en dos dígitos numéricos(01-31)
ddd	Las tres primeras letras del día de la semana (Dom-Sáb)
Dddd	Nombre completo del día de la semana
E	Día de la semana en números (1-7)
m	Mes del año en uno o dos dígitos(1-12)
mm	Mes del año en dos dígitos (01-12)
mmm	Las tres primeras letras del mes (Ene-Dic)
Mmmm	Nombre completo del mes (Enero-Diciembre)
t	Número del trimestre del año (1-4)
aa	Los últimos dígitos del año (01-99)
aaa	Año completo(0100-9999)
h	La hora en 1 o 2 dígitos (0-23)
hh	La hora en 2 dígitos (00-23)
n	El minuto en 1 o 2 digitos (0-59)
nn	El minuto en 2 dígitos (00-59)
s	El segundo en 1 o 2 dígitos (0-59)
ss	El segundo en 2 dígitos (00-59)
AM/PM	Reloj de 12 horas con las letras que correspondan
AMPM	Reloj de 12 horas con el indicador de mañana/tarde definido en el panel de control de Windows

6.5.2.7. Formato de campos tipo Si/No

Si no se ha seleccionado un formato para este campo, Access mostrará un -1 para Sí y un 0 para No.

En este tipo de campos hay formatos predefinidos y también se pueden crear formatos personalizados. Hay tres secciones:

1° Escribir punto y coma

2° Representar los valores que no sean cero

3° ";"Representar los valores cero

En el campo Sexo se ha seleccionado un tipo de dato Si/No. En este tipo de dato no hay nada predefinido para seleccionar un sexo.

; "Hombre" ; "Mujer"

Cuando se introducen los datos dependiendo desde que vista aparecerá un botón al que hay que activar o desactivar. Si el botón está en blanco el valor es cero, por tanto el sexo de ese registro será mujer. Si se activa o selecciona el valor de ese campo ya no será cero por tanto el sexo será hombre.

6.5.3. Lugares decimales

Esta propiedad sólo la tienen los campos de tipo numérico y de moneda. Determina el número de cifras decimales en la presentación de los campos.

Esta propiedad tiene dos configuraciones:

1 De 0 a 15. Aparecerán tantas cifras decimales como se indiquen sin tener en cuenta las que se especifiquen en el formato.

2 Auto: aparecerá el número de cifras decimales predeterminadas para cada formato o la que este definida en la propiedad formato.

6.5.4. Máscara de entrada

Esta propiedad la tienen los campos de texto, numérico, fecha/hora y de moneda. Obliga a que los datos introducidos en un campo se ajusten a una plantilla determinada.

Para crear una máscara hay un esquema de 3 partes separadas por ";":

1° Presenta la máscara de entrada

2° Indica si los caracteres literales empleados en la máscara se almacenan o no en el campo junto con los datos. Se escribe 0 para que se guarden y 1 para que no se guarden.

3° Especifica el carácter que debe aparecer en los espacios en blanco de la máscara. Access por defecto utiliza el subrayado.

Los códigos son los siguientes:

Código	Función
0	Dígito (0-9). Introducción obligatoria. No permite signos
#	Dígito o espacio. Introducción opcional. Las posiciones en blanco se convierten en espacios y se permiten los signos
9	Dígito o espacio. Introducción opcional. No permite signos
L	Letra (A-Z). Introducción obligatoria
?	Letra (A-Z). Introducción opcional.
A	Letra o dígito. Introducción obligatoria
a	Letra o dígito. Introducción opcional
&	Cualquier carácter o espacio. Introducción obligatoria
C	Cualquier carácter o espacio. Introducción opcional
.,:;-/	Marcador de posición decimal y separador de miles, fecha y hora.
<	Convierte los caracteres en minúsculas
>	Convierte los caracteres en mayúsculas
!	Hace que la máscara de entrada se rellene de derecha a izquierda
\	Hace que el carácter que venga a continuación se presente como un carácter literal

Ejemplo:
(900)009-00 00; 0; "_"

6.5.5. Título

Esta propiedad la tienen todos los tipos de campos.

Especifica la etiqueta que se utilizará en la presentación del campo cuando se crean tablas, formularios e informes.

6.5.6. Valor predeterminado

Esta propiedad la tienen todos los campos menos los de tipo contador y Objeto OLE.

Introduce un valor por defecto en todos los campos. Este valor lo introduce el usuario cuando un campo va tener casi siempre el mismo valor. Cuando este valor varíe se puede modificar.

6.5.7. Regla de validación

Esta propiedad se puede establecer en todos los campos menos en los de tipo Contador y Objeto OLE.

Se especifican las condiciones que deben cumplir los datos que se introduzcan, si los datos no cumplen las condiciones Access no admitirá ese dato.

Para introducir las condiciones que debe cumplir un campo se selecciona Regla de validación en las propiedades del campo. A la derecha del espacio en blanco hay unos puntos suspensivos. Se hace clic sobre ellos. Aparecerá una ventana para generar expresiones, en esta ventana habrá que especificar esas condiciones.

6.5.8. Texto de validación

Cuando Access no admite un dato porque no cumple la regla de validación no aparece ningún mensaje que explique por qué no admite el dato a no ser que se utilice el texto de validación.

En esta propiedad se debe introducir cuál es la condición que debe cumplir el dato para que el usuario lo sepa.

6.5.9. Requerido

Se aplica a todos los campos menos a los de tipo Contador.

Si se encuentra activado Si no dejará que el usuario abandone un registro sin haberlo rellenado.

6.5.10. Permitir longitud cero

Se aplica a los campos de tipo texto y memo.

Esta propiedad es útil para las consultas y expresiones, ya que los valores nulos se comportan de distinta forma.

Para introducir una cadena de longitud cero se teclea dos comillas dobles sin espacio entre ellas ("").

6.5.11. Indexado

Se puede aplicar a todos los tipos de campo menos a Memo, Si/No, y Objeto OLE.

Esta propiedad crea un índice de ese campo. De modo que acelera las búsquedas de un registro por el contenido de ese campo. No conviene aplicarlo mas que al campo por el que se vayan a realizar las búsquedas porque si no la actualización de los datos será muy lenta.

Sin duplicados: es una de las opciones de esta propiedad y significa que no puede haber dos campos con la misma clave. Con duplicados, hace que Access cree un índice normal con cada uno de los registros.

6.6. LOS FORMULARIOS

La introducción de los datos directamente sobre las tablas es bastante incómoda. No sólo no se pueden ver todos los campos sin desplazarse con la barra de herramientas, sino que además los registros están uno encima de otro. Si se pierde la referencia del registro se pueden introducir datos que no correspondan a ese registro.

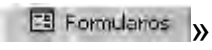
Los formularios permiten la introducción de datos en las tablas de una forma más sencilla y más limpia. En vez de introducir los datos directamente sobre la tabla, los datos se introducen en la tabla a través de los formularios.

Hay diferentes formatos de formularios, en alguno de ellos los registros se pueden ver de forma aislada, en otros, todos los registros juntos, o también se puede diseñar un formulario con un formato totalmente personalizado.

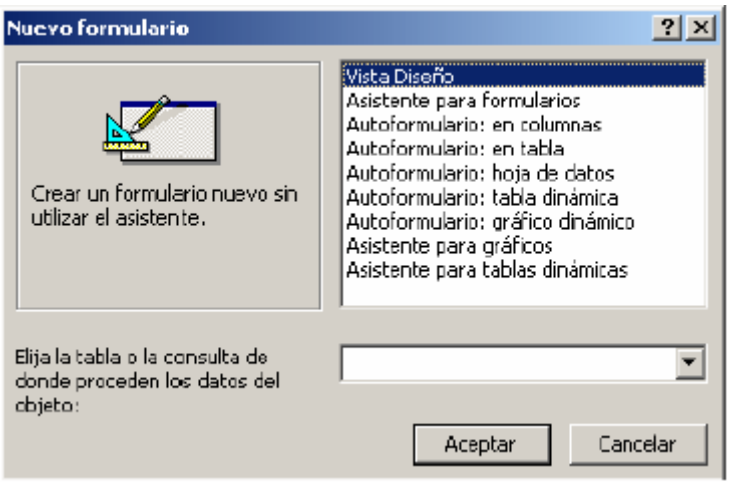
En una base de datos se puede crear más de un formulario basado en una misma tabla. Un formulario puede tomar varios campos de una tabla o todos, incluso puede tomar campos de diferentes tablas.

Las tablas almacenan la información, los formularios se encargan de recogerla. Para que las tablas puedan incluir los datos de los formularios deben estar cerradas. Al cerrar o guardar los formularios, las tablas se actualizan.

6.6.1. Crear formularios

Para crear un nuevo formulario se selecciona la pestaña « 

» de la ventana de la base de datos. Se pulsa «Nuevo», aparecerá la siguiente ventana:



En esta ventana se dan siete posibilidades distintas para crear un formulario. Crear el formulario manualmente en Vista diseño, con asistentes o con los autoformularios.

- Vista Diseño: seleccionando esta opción se puede crear un formulario totalmente personalizado.

- Asistente para formularios: Access crea automáticamente un formulario con los campos que el usuario seleccione. Con este asistente se pueden crear formularios de formatos muy distintos.

- Autoformulario: columnas: Access crea un formulario en columnas con todos los campos de la tabla. Cada registro aparece de forma independiente con los campos ordenados en una columna.

- Autoformulario: tabula: crea automáticamente un formulario con todos los campos en formato tabular: Los registros aparecen en filas y columnas. En este tipo de formulario se presentan todos los registros que se hayan introducido.

- Autoformulario: hoja de datos: esta opción crea automáticamente un formulario con el formato de hoja de datos. Este es el mismo formato que el que tienen las tablas para introducir datos.

- Asistente para gráficos: crea un formulario con un gráfico, muestra los datos en formato gráfico.

- Asistente para tablas dinámicas: crea un formulario de Microsoft Access con una tabla dinámica de Microsoft Excel. Una tabla dinámica es una tabla interactiva que puede resumir grandes cantidades de datos utilizando el formato y los métodos de cálculo que se elijan.

Para continuar con el ejemplo del hospital se va a crear un formulario de la tabla pacientes. Se selecciona Autoformulario: columnas y la tabla Pacientes. Para seleccionar la tabla se pulsa sobre la flecha de la derecha, se despliegan las diferentes tablas que existen en la base de datos. Para seleccionar una se hace clic sobre ella. Si se quisiera extraer datos de varias tablas se tendría que seleccionar Vista Diseño o Asistente para formularios, o bien crear una consulta de varias tablas y hacer un formulario con ella.

6.6.2. Autoformulario: columnas

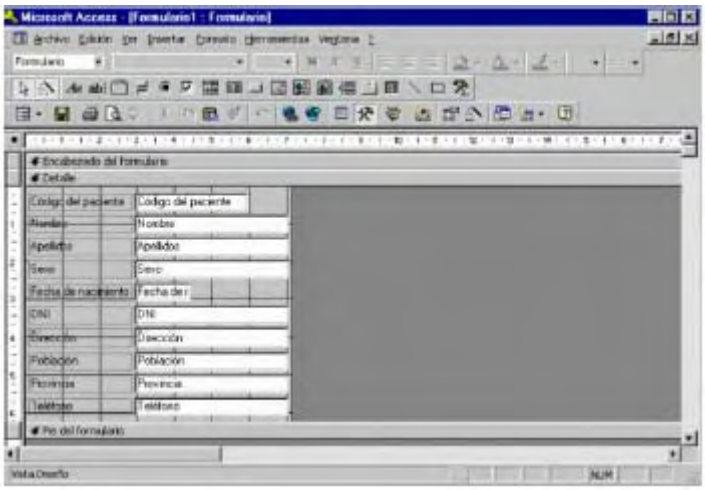
Al seleccionar la opción Autoformulario: columnas Access automáticamente generará un formulario en columnas y lo abrirá en Vista Formulario:

6.6.2.1. Vistas de Formulario

En Formulario hay tres vistas distintas, además de la presentación preliminar.

6.6.2.1.1. Vista diseño

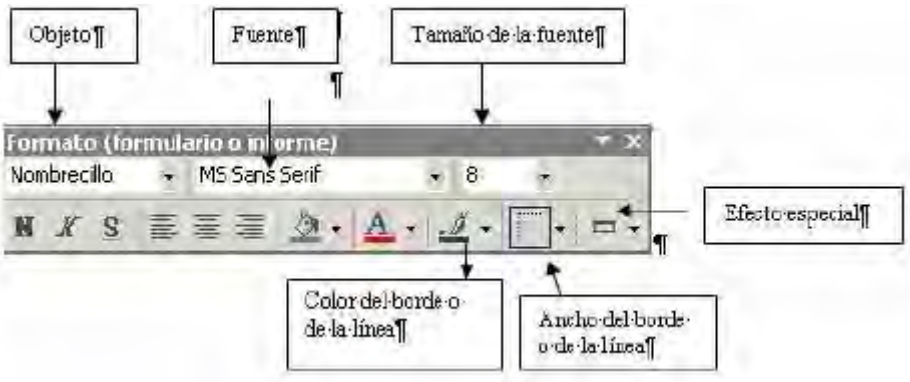
Como su nombre indica en esta vista fundamentalmente se varía el diseño del formulario. La ventana se compone de un menú y tres barras de herramientas:



La primera barra de herramientas se refiere al diseño de formularios. Puede aparecer integrada debajo del menú o de forma independiente tal como aparece en el gráfico de debajo:

El botón «caja de herramientas» sirve para abrir o cerrar el cuadro de herramientas. Y el botón «autoformulario» para crear un autoformulario cuando se desee. El resto de los elementos que componen la barra ya se han visto.

La siguiente barra de herramientas se refiere al formato del formulario, es muy parecido a la barra de herramientas de un editor de textos.



Objeto: dentro del marco aparecerá el objeto seleccionado. Si se pulsa la flecha de la derecha aparecerán todos los objetos del formulario. Haciendo clic sobre cualquiera de ellos el objeto se seleccionará y se le podrán aplicar todas las características de formato que se quiera.

Seleccionando a través de este menú los objetos sólo se pueden seleccionar de uno en uno.

- Fuente: al pulsar la flecha de la derecha se despliegan los diferentes tipos de fuentes que existen, y que se pueden aplicar a todos los objetos de un formulario.

- Tamaño: sirve para dar mayor número de puntos a una fuente, y de esta forma aumentar el tamaño.

- Color del borde o de la línea: pulsando sobre este botón se despliega una paleta de colores predeterminados. Al elegir cualquiera de ellos este color se aplicará al objeto que esté seleccionado.

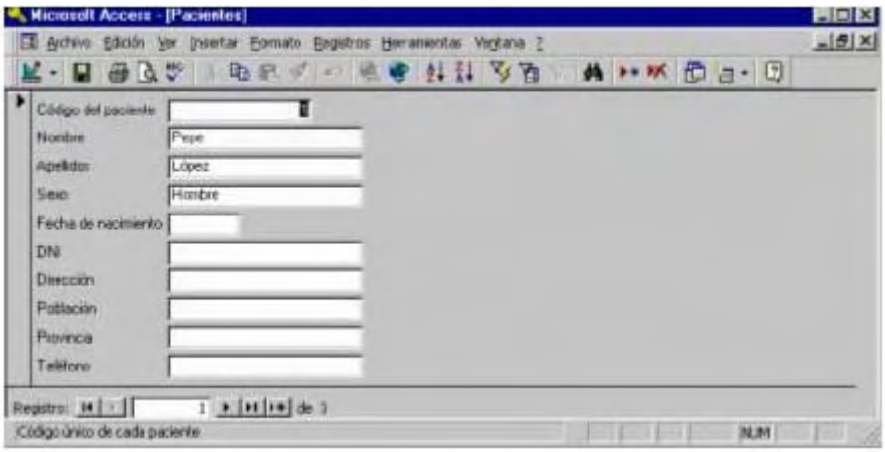
- Ancho del borde de la línea: hay seis grosores diferentes además del que viene por defecto.

- Efecto especial: hay seis efectos especiales para aplicar al campo: sin relieve, con relieve, bajo relieve, grabado, sombreado y cincelado.

En el cuadro de herramientas se encuentran todos los elementos que componen el formulario, se explicarán en el apartado 6.3. los controles:

6.6.2.1.2. Vista formulario

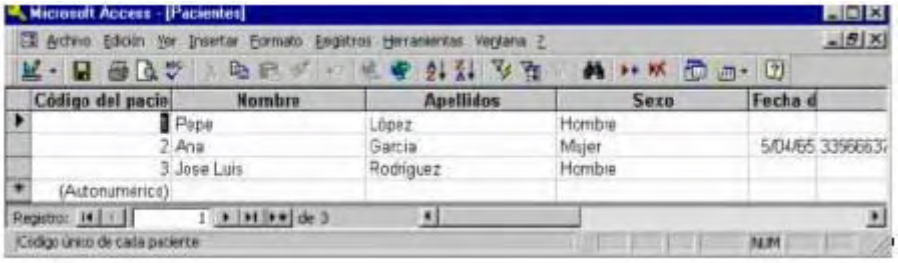
Esta vista es para introducir los datos en el formulario. La barra de herramientas es la misma que la de Vista Hoja de datos, ya que en ambas vistas se pueden introducir registros:



Los elementos de la barra de herramientas son todos conocidos.

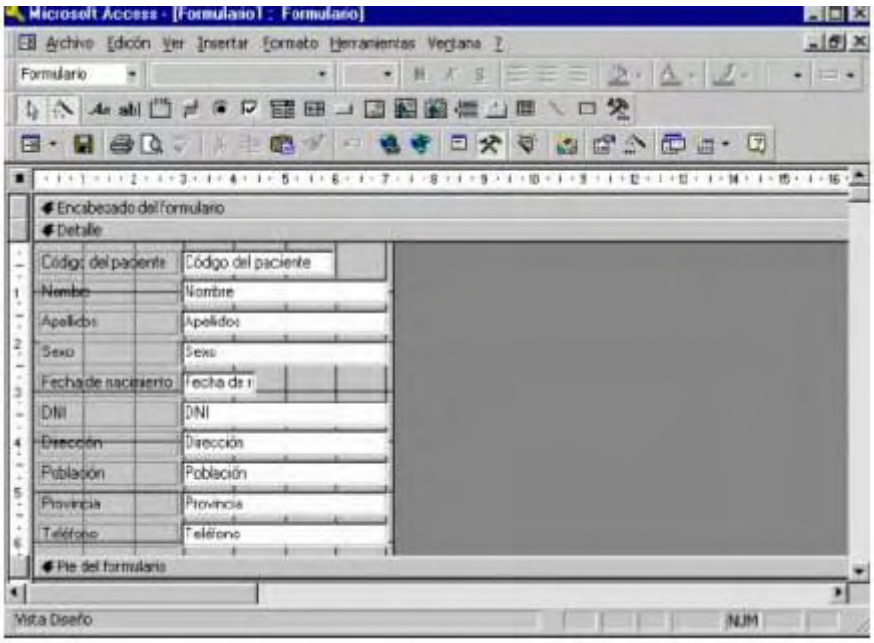
6.6.2.1.3. Vista Hoja de datos

En esta vista se introducen datos como si se tratara de una hoja de cálculo. Esta vista es común con la Vista Hoja de datos de las tablas, la única diferencia con respecto a las tablas es que en el formulario puede haber campos de diferentes tablas.



El formulario creado automáticamente no es muy claro. Las etiquetas de algunos campos no se ven completas, y los datos de los diferentes campos se alinean en distintas posiciones.

Para mejorar la presentación se pulsa el botón «Vista» de la barra de herramientas, y se cambia a «Vista Diseño», el aspecto del formulario será este:



Para modificar la posición, el tamaño y el aspecto en general de cada elemento primero es necesario saber qué es cada elemento, como se mueven, modifican o añaden nuevos elementos.

Los elementos que componen un formulario se llaman controles.

6.6.3. Los controles

Toda la información de un formulario está contenida en los controles. Los controles son objetos de un formulario que muestran datos, realizan acciones o decoran el formulario. Los controles también son elementos del informe.

Los controles pueden ser dependientes, independientes o calculados.

- Control dependiente: está unido a un campo de una tabla o consulta. Los controles dependientes se utilizan para mostrar, introducir y actualizar valores de los campos de la base de datos.

- Control independiente: no tiene un origen en una tabla o consulta. Los controles independientes se pueden utilizar para mostrar información, líneas, rectángulos e imágenes, independientemente de que estos existan en la tabla.

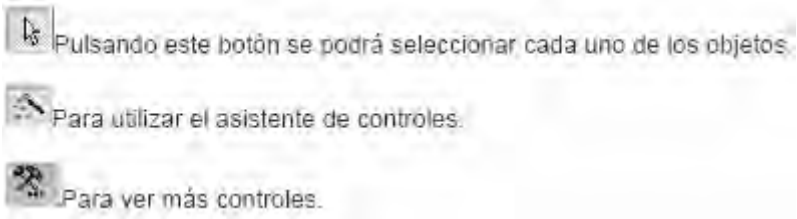
- Control calculado: el origen de los datos es una expresión, no un campo. Una expresión es una combinación de operadores ("=";"+", "-","*" y "/"), nombres de controles, nombres de campos, funciones que devuelven un solo valor y valores constantes.

La expresión puede incluir datos de un campo de la tabla o consulta del formulario o datos de otro control del formulario.

A los controles se tiene acceso a través del cuadro de herramientas de la Vista diseño. Estos son los diferentes tipos de controles:

Cuadro de texto	Sirven para mostrar o introducir datos
Etiqueta	Crea una etiqueta
Grupo de opciones	Formado por un grupo de casillas de verificación o botones de opción. Sólo permite que se active una opción.
Botón de opción	Para valores Si/No. Se puede utilizar dentro de un grupo de opciones.
Casilla de verificación	Para valores Si/No. Se puede utilizar dentro de un grupo de opciones.
Botón de alternar	Para valores Si/No. No puede utilizarse dentro de un grupo de opciones.
Cuadro combinado	Permite seleccionar un elemento de una lista o escribir el dato directamente.
Cuadro de lista	Permite seleccionar un elemento de una lista.
Botón de comando	Inserta un botón que al ser pulsado ejecuta instrucciones.
Imagen	Inserta un marco para incluir una imagen. No es un objeto OLE. No se edita.
Marco de objeto dependiente	Inserta un marco para incluir un objeto OLE que depende del valor de un campo.
Marco de objeto independiente	Inserta un marco para incluir un objeto OLE que no depende del contenido de un campo.
Subformulario/subinforme	Permite introducir un formulario dentro de otro.
Salto de página	Cuando el formulario tiene mas de una página, así se indica dónde empieza cada una.
Línea	Inserta una línea en el formulario.
Rectángulo	Inserta un rectángulo.

En el cuadro de herramientas hay otros botones que no son controles:



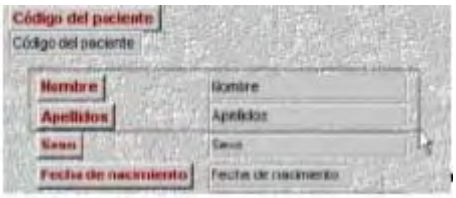
6.6.3.1. Manejo de los controles

Hay controles que muestran información que sólo está en el formulario (controles independientes) estos son las etiquetas, las líneas y los rectángulos. Los cuadros de texto, los marcos de objeto muestran información contenida en la tabla o consulta adyacente.

Para realizar cualquier modificación en un control se hace clic sobre él.

Para modificar el tamaño de la etiquetas y desplazar los campos se selecciona la etiqueta:

- Seleccionar: para seleccionar varios campos se debe pulsar la tecla «Mayúsculas» o «Control» y sin soltarla hacer clic sobre todos los elementos que se desee seleccionar. Si por equivocación se seleccionara uno que no se quisiera, sin soltar la tecla «Mayúsculas» se debe volver a hacer clic para deseleccionar. También se puede seleccionar uno o varios elementos haciendo un cuadrado con el ratón. Para realizar esta operación se hace clic sobre una de las esquinas de lo que va a ser el cuadrado, se arrastra el ratón en diagonal sin soltar el botón, cuando ese cuadrado abarque todo lo que se quiere seleccionar se suelta el botón del ratón.



- Modificar el tamaño: haciendo clic sobre los cuadraditos pequeños de las esquinas se modifica el tamaño del objeto seleccionado. El ratón se convertirá en una flecha que apunta en dos direcciones y al estirar hacia una de ellas variará el tamaño.

- Modificar la posición: si se pulsa sobre la esquina superior izquierda el ratón se convertirá en una mano con el dedo índice apuntando hacia arriba.

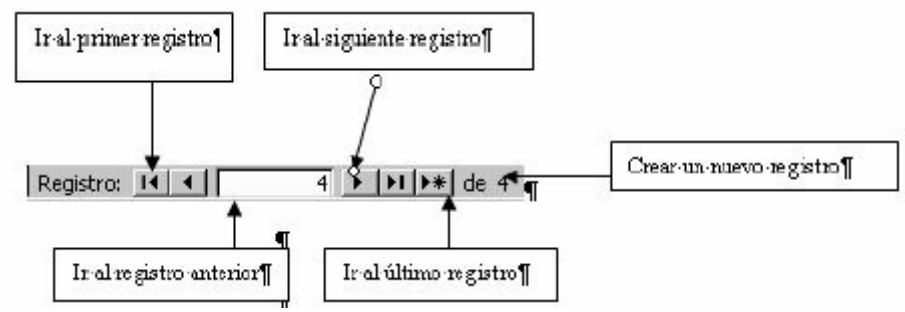
Haciendo clic se podrá mover la etiqueta o el campo de forma independiente. Sin embargo si pasa el ratón sobre cualquiera de los bordes del campo o de la etiqueta el cursor será una mano abierta. Si se hace clic y se mueve se moverá tanto el campo como la etiqueta.

- Alinear: para alinear varios elementos primero hay que seleccionarlos y luego abrir el menú «Formato | Alinear» y se desplegará un menú para seleccionar respecto a qué lado se deben alinear esos campos.

En vista diseño también se pueden añadir más elementos a la presentación de formularios que se verán más adelante.

Modificando el formulario con respecto a la creación automática del programa, el aspecto del formulario será más claro:

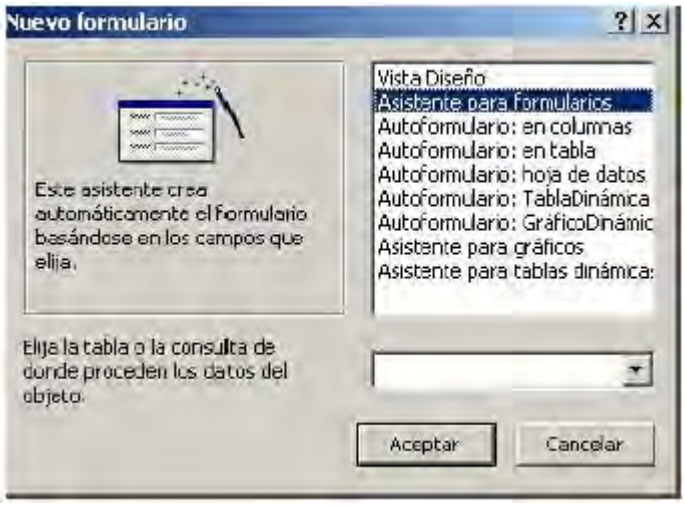
Una vez creado este formulario, se puede introducir todos los datos que se quiera. Para moverse entre los diferentes registros se utilizan los botones que hay debajo del formulario:



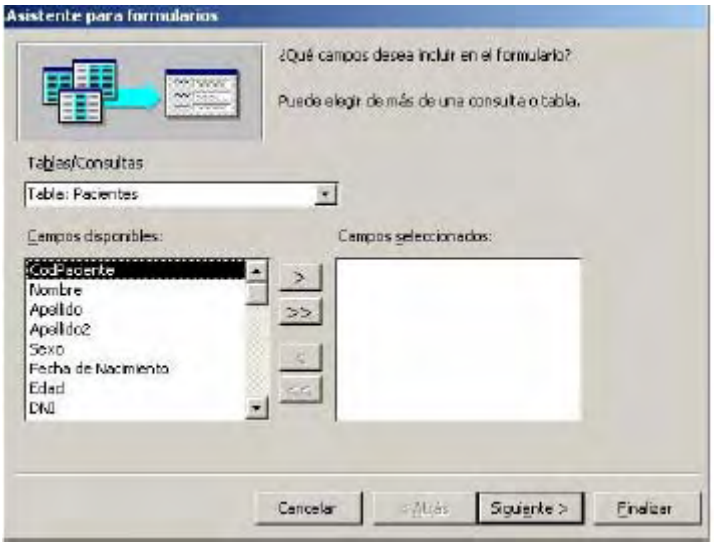
Para crear un nuevo registro se puede pulsar el botón «Crear un nuevo registro». Para pasar de un campo a otro dentro del formulario se puede utilizar el ratón, la tecla tabulador o la tecla «Intro». Una vez se han introducido todos los datos de un registro si se vuelve a pulsar «Intro» se crea un nuevo registro en blanco.

6.6.4. Asistente para formularios

Otra forma de generar un formulario es utilizando el asistente para formularios. Tras pulsar «Nuevo» aparecerá esta ventana:



En este caso se selecciona Asistente para formularios. Después se selecciona la tabla de la que se van a extraer los campos y se pulsa «Aceptar». Aparecerá esta ventana:



En esta ventana se eligen los campos que se desea que aparezcan en el formulario. Aunque en la ventana anterior se ha seleccionado la tabla de la cual se quieren extraer los campos para el formulario, aún se puede cambiar de tabla pulsando sobre la flecha que se encuentra bajo «Tablas/Consultas». Una vez seleccionada la tabla se escogen los campos que se quiere que aparezcan en el formulario. Se pueden seleccionar todos los campos o sólo algunos. Incluso se pueden seleccionar campos de diferentes tablas para un mismo formulario.

Para seleccionar los campos del formulario se utilizan los botones que hay entre «Campos disponibles» y «Campos seleccionados»:

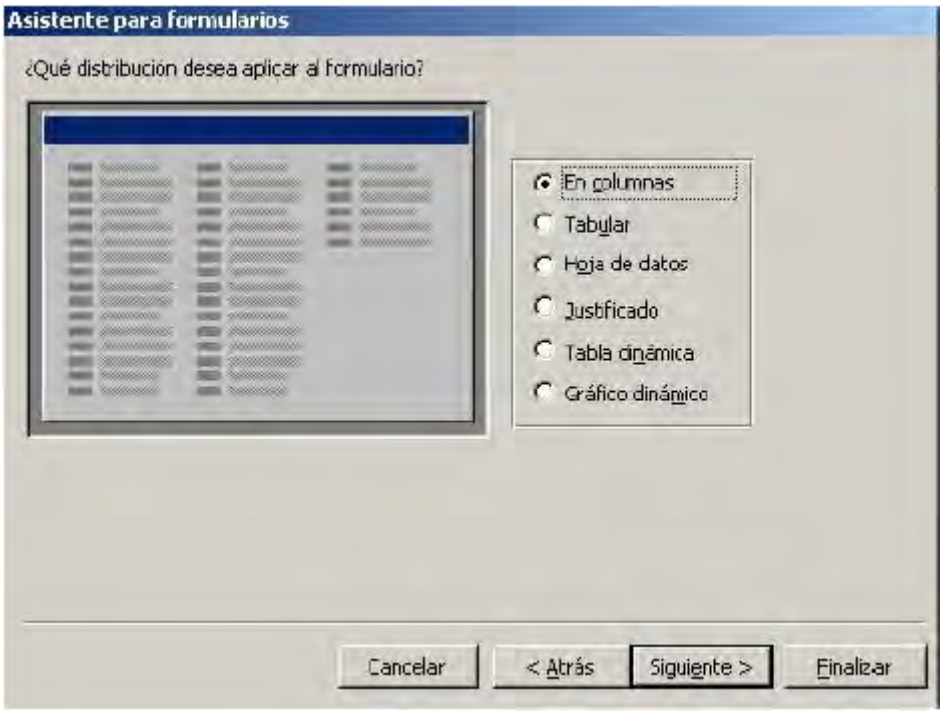
>	Pasar un campo
>>	Pasar todos los campos
<	Eliminar un campo
<<	Eliminar todos los campos ya seleccionados

Para añadir un campo a la lista de Campos seleccionados: primero se selecciona el campo con el ratón y después se pulsa el botón «>». El campo aparecerá en la zona de la derecha: «Campos seleccionados». A su vez el campo desaparecerá de la lista de «Campos disponibles».

Tras seleccionar todos los campos que se quiere que aparezcan en el formulario se pulsa el botón «Siguiente» para continuar con la creación del formulario.

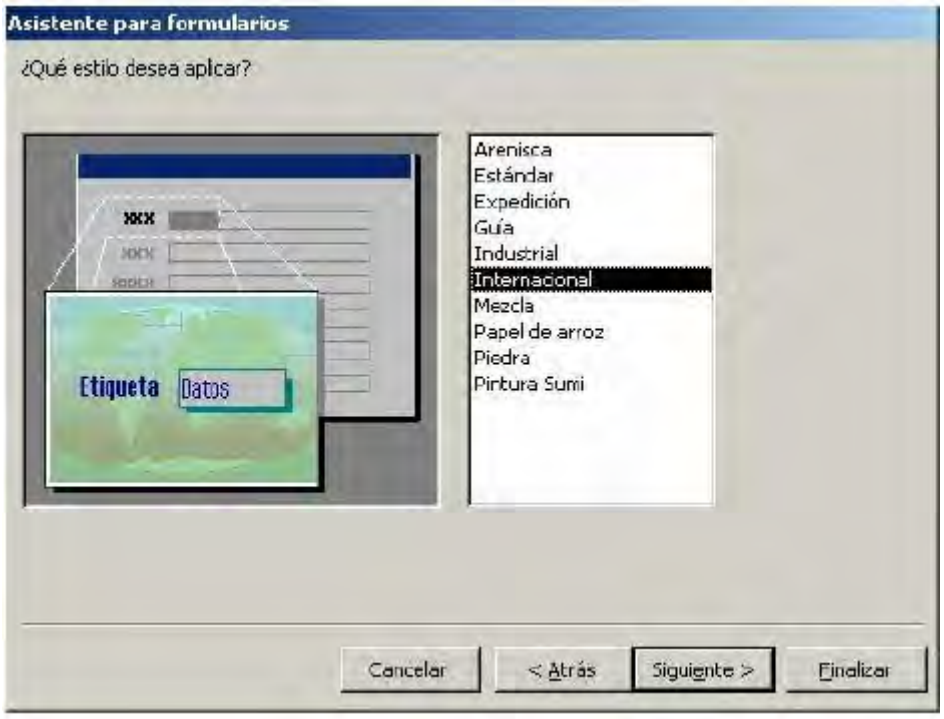
Siempre se puede volver al paso anterior pulsando el botón «Atrás» para volver y modificar alguna de las elecciones hechas. Si se pulsa el botón «Cancelar» se cancela la creación de un formulario sin guardar lo que se ha hecho. Si se pulsa el botón «Finalizar» el formulario quedará guardado hasta el paso en el que se esté en ese momento.

Tras pulsar «Siguiente» aparece esta pantalla, en la que se debe seleccionar el tipo de distribución de los campos.



Al seleccionar cada opción aparece una muestra de cada distribución en la ventana izquierda:

Tras elegir el tipo de distribución se pulsa «Siguiente».



En esta ventana se selecciona el aspecto gráfico del formulario. El color o la imagen de fondo. El color de las etiquetas y los botones. Se selecciona uno de los modelos de la lista y se pulsa «Siguiente».

En esta ventana se le da nombre al formulario. Este es el último paso de creación del formulario, a partir de este momento se pueden introducir datos a través del formulario.

También cabe la opción de seguir modificando el diseño del formulario desde la Vista diseño.

El formulario ya está creado, dependiendo de la modificaciones personales tendrá un aspecto distinto, pero la finalidad es la misma: introducir datos de una forma más cómoda. Este podría ser el aspecto de un formulario retocado desde la Vista diseño:

6.6.5. Formulario con subformulario

La utilidad de un formulario con un subformulario es poder observar los datos de dos tablas que tienen algo en común.

Para crear un subformulario se va a utilizar el ejemplo2. En este ejemplo cada paciente tiene asignado un médico de cabecera. De esta forma cada médico tiene una lista de los pacientes que le corresponden. Con el subformulario será posible ver dentro del formulario de los médicos la lista de los pacientes que le corresponde a cada médico.

De forma que tras crear el formulario médicos, con todos los datos del médico, se va a crear dentro del formulario, el subformulario.

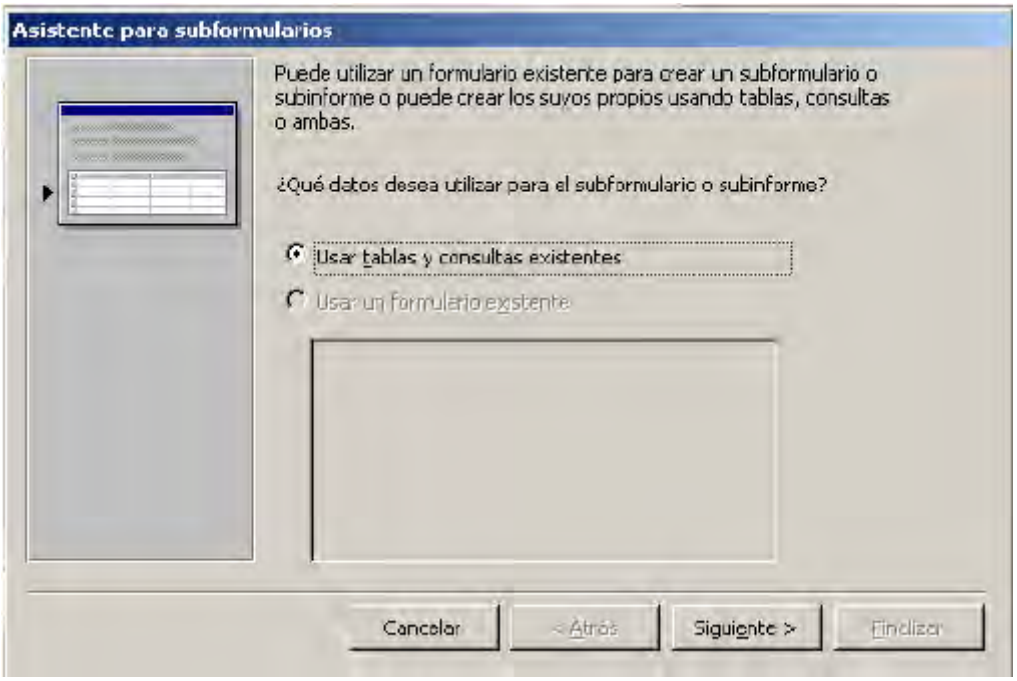
Para generar un subformulario de manera manual dentro de un formulario primero se debe abrir el formulario principal en Vista diseño. Dentro del cuadro de herramientas se encuentra el botón «Subformulario/Subinforme» «  ».

Siguiendo con el ejemplo2, se abre el formulario de Médicos con el que se va a trabajar, en modo diseño. Y se hace clic sobre el botón «subformulario»: «  »

Se crea un rectángulo con el ratón en la zona donde se quiera situar el subformulario. Para crear el rectángulo se hace clic en lo que va a ser la esquina superior izquierda y se desplaza el ratón en diagonal hacia la esquina inferior derecha. Cuando el rectángulo tenga la forma deseada se suelta el ratón. Dentro del rectángulo creado estará el subformulario.

Tras crear el rectángulo aparece la siguiente ventana de diálogo, donde se elige si el formulario se hace a partir de una tabla o de un formulario ya existente.

El diseño será mejor si se escoge un formulario, pero para eso el formulario de pacientes tiene que estar ya creado. En este caso se selecciona «Tabla o consulta».



En la ventana siguiente se selecciona la tabla de Pacientes y los campos que se quiere que formen parte del subformulario.

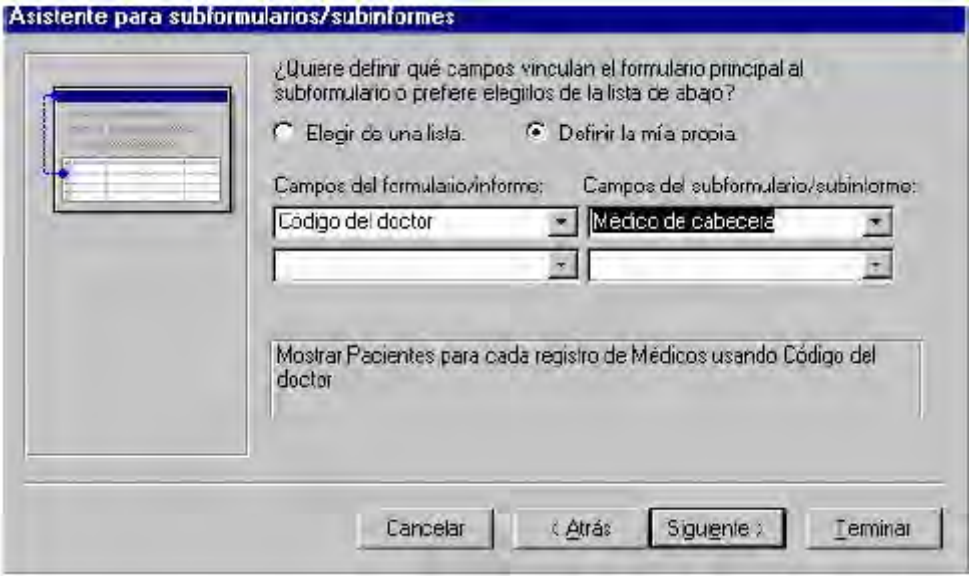
Se pulsa «Siguiente». En esta ventana hay que determinar el tipo de conexión que existe entre el formulario principal y el subformulario. El programa sugiere vínculos, si

ninguno de ellos coincide con la relación real que hay entre las dos tablas se selecciona Definir la mía propia, si alguno de los vínculos coincide con la relación real. Se selecciona y se pulsa «Siguiente».

En este caso, la relación que sugiere el programa no coincide con la realidad.

Los campos de las dos tablas que coinciden en su contenido son Código del médico de la tabla médicos con el campo médico de cabecera de la tabla pacientes.

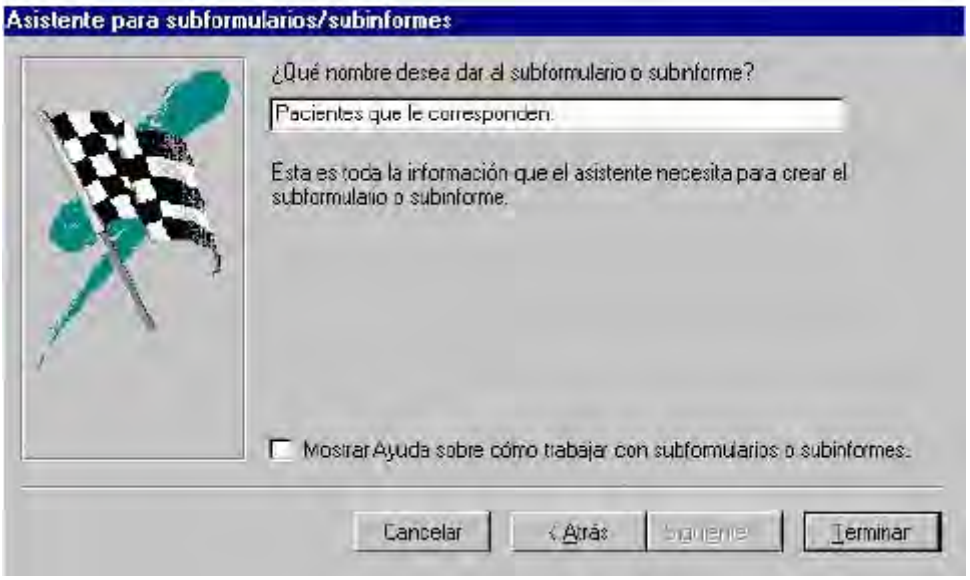
Para poder definir bien la relación se selecciona Definir la mía propia. El aspecto de la ventana variará:



En Campos del formulario/informe se debe seleccionar el campo del formulario Médicos que permite la relación con la otra tabla; Visitas. y en Campos del subformulario/subinforme se debe seleccionar el campo que permite la relación de la

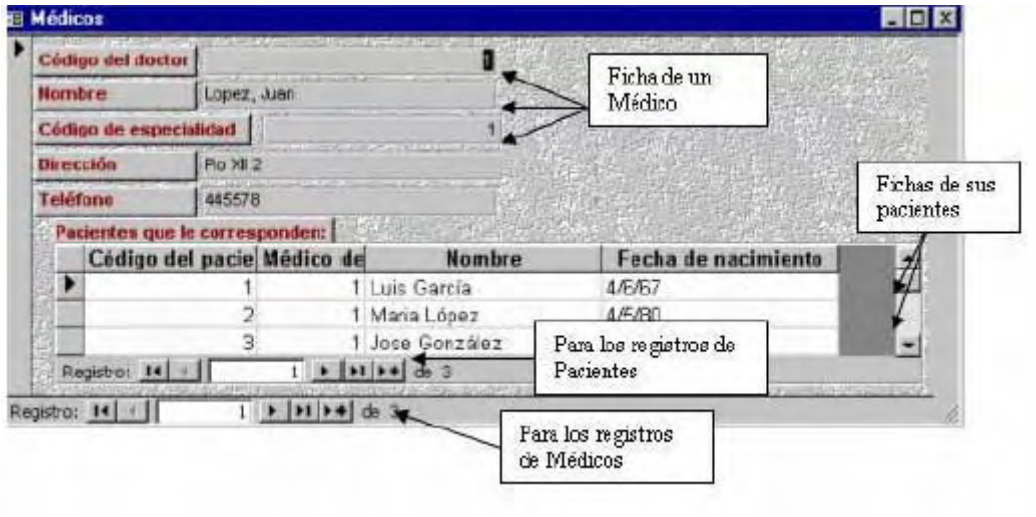
tabla Visitas con el formulario Médicos. Los campos que permiten la relación deben contener el mismo dato, no importa su nombre si no los datos que contengan.

Para seleccionar los campos del formulario y del subformulario se pulsa la flecha que hay en el primer recuadro bajo Campos del... En la siguiente ventana se le da nombre al subformulario, ya que se almacenará junto con los demás formularios.



Tras realizar estos pasos el formulario de médicos quedará así en el Vista Diseño:

En Vista Formulario se ve la utilidad del subformulario:



6.7. ORDENAR, BUSCAR, REEMPLAZAR Y FILTRAR

Los comandos de buscar, reemplazar y ordenar resultan prácticos para la búsqueda de datos en una tabla. Es una forma muy sencilla de buscar datos y realizar modificaciones dentro de una tabla. «Edición/Buscar» o bien «Edición/Reemplazar».

6.7.1. Ordenar Registros

Los registros de las tablas aparecen generalmente en el orden en el que han sido introducidos aunque también pueden aparecer ordenados por el campo que es la clave principal.

Este orden no tiene por qué quedar así, los registros se pueden ordenar de formas muy distintas según el contenido de los campos.

Para ordenar los registros de la tabla de pacientes se abre la tabla de pacientes desde la vista hoja de datos:

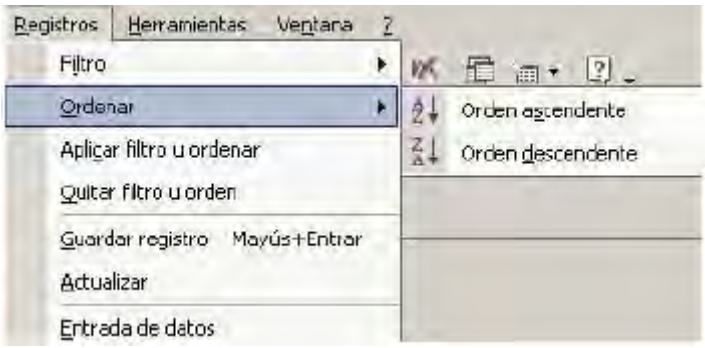
6.7.1.1. Ordenar los registros con un campo

El orden se puede aplicar a un solo campo o a varios. En este caso se van a ordenar los pacientes por el apellido alfabéticamente, lo primero es situarse en la columna de Apellidos. El orden alfabético puede ser ascendente o descendente. Si no se ordenan los datos de los pacientes aparecen en el orden en el que se introdujeron.

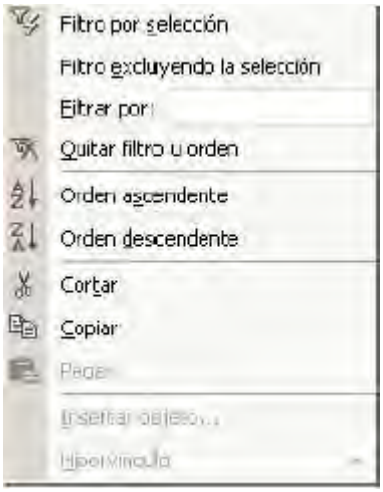
Para ordenar los pacientes según el apellido de forma ascendente se pueden seguir tres procedimientos:

1 Pulsar sobre el botón «  ». Si se prefiere el orden descendente: «  »

2 Se selecciona el menú «Registros|Ordenar|Orden Ascendente»



3 Utilizar el menú que aparece al pulsar el botón derecho del ratón situándose sobre la columna que se desea ordenar:



Siguiendo cualquiera de estos tres procedimientos se pueden ordenar los registros de un campo de forma ascendente o descendente. El resultado del orden será éste:

Los campos de tipo Memo u Objeto OLE no se pueden ordenar. Esto mismo se puede hacer desde la Vista Diseño seleccionando el campo.

6.7.1.2. Ordenar los registros con varios campos

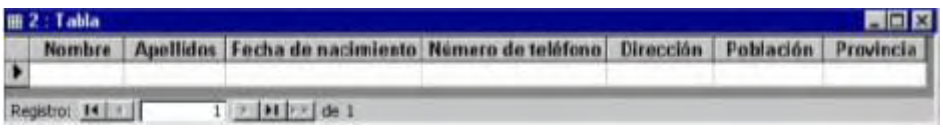
En el ejemplo anterior se ordenaban los registros de los pacientes según un único dato: su apellido. Si se quisiera ordenar esta misma tabla pero también en función del apellido, la edad y la provincia de procedencia el procedimiento sería distinto.

Desde la Vista Formulario sólo es posible ordenar los registros con un campo. Sin embargo la Vista Hoja de datos permite seleccionar varios campos para realizar este orden, siempre y cuando estos campos estén juntos en la vista Hoja de Datos.

Para que los campos por los que queremos realizar el orden estén juntos es necesario mover las columnas que contienen los datos. Para mover una columna primero se tiene que seleccionar. Al pasar el ratón por encima el cursor del ratón se convierte en una flecha negra «  », se pulsa el botón del ratón. La columna ya está seleccionada si se colorea de negro.

Para mover la columna seleccionada se debe pulsar el botón izquierdo del ratón de nuevo, y sin soltarlo arrastrar la columna a la nueva posición. Se debe soltar el botón cuando la unión de dos columnas esté más oscura.

El campo trasladado aparecerá entre esos dos campos.



Una vez los campos apellidos, edad y provincia están situados de forma contigua, se seleccionan los campos. Para seleccionarlos los tres con todos sus datos, se utiliza sólo la primera fila, donde están los nombres de los campos. Se hace clic sobre la primera cabecera y sin soltar el botón, se arrastra el ratón hasta la última. Al seleccionar la cabecera se selecciona toda la columna, de forma que sólo hay que seleccionar la cabecera para seleccionar todos los registros que contiene.

Una vez seleccionadas las tres columnas se pulsa el botón de «orden ascendente» «  ». Los datos se ordenarán según la primera columna empezando de izquierda a derecha.

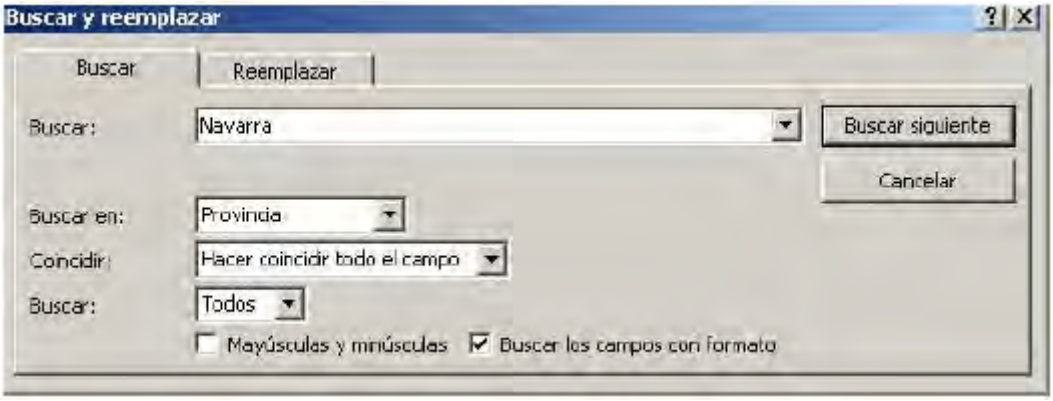
Si hay dos registros iguales en la primera columna, el criterio de orden entre esos registros será la segunda columna. Si volvieran a coincidir en la segunda entonces se ordenarán según el tercer campo, y así sucesivamente.

6.7.2. Buscar datos

En las tablas se puede buscar un dato determinado. Esta forma de búsqueda localiza un registro conociendo uno de sus datos.

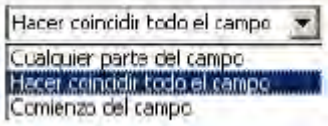
La diferencia de esta búsqueda con respecto a los filtros es que sólo localiza los registros con ese dato de uno en uno. Los filtros localizan todos los registros que tienen un dato en común de una sola vez.

En la tabla pacientes vamos a localizar a un paciente procedente de Navarra. Para realizar esta operación primero es muy importante abrir la tabla de pacientes y situar el cursor sobre el campo Provincia. Se selecciona el menú «Edición|Buscar» o se pulsa el botón «» .A continuación se abrirá esta ventana:



- Buscar: en este campo se determina el dato a buscar en los registros.
- Buscar en: aquí se determina la dirección de la búsqueda según la situación del cursor en la tabla.

- Arriba: si se selecciona y el cursor está en mitad de la tabla, buscará sólo en la mitad superior de la tabla. Cuando llegue a la parte superior de la tabla no seguirá buscando.
- Abajo: si se selecciona pasará lo mismo pero hacia el final de la tabla.
- Todos: si se selecciona, el programa seguirá buscando hasta que haya encontrado todos los registros que contengan ese dato independientemente de la zona de la tabla en la que estén.
- Coincidir: aquí se debe señalar si el texto que se ha escrito en buscar debe coincidir con:



- Hacer coincidir todo el campo: los datos deben ser exactamente esos en el registro completo.

- Cualquier parte del campo: si sólo debe encontrarse ese texto en alguna parte del registro.

- Comienzo del campo: el dato debe empezar por ese texto.

De esta forma se puede buscar un dato sin recordar la palabra entera, por ejemplo el apellido de una persona si no se recuerda cómo se escribía:"Jiménez"; "Gimenez", se puede poner sólo la parte de la que uno esté seguro: "imenez". O buscar todos los apellidos que terminen en "ez". Todos los nombres que empiecen por:

- Mayúsculas y minúsculas: si en el campo Buscar se escribe alguna mayúscula y se selecciona Coincidir Mayúsculas y minúsculas, el registro que se busca deberá tener las mismas mayúsculas y minúsculas, aunque el texto sea el mismo.

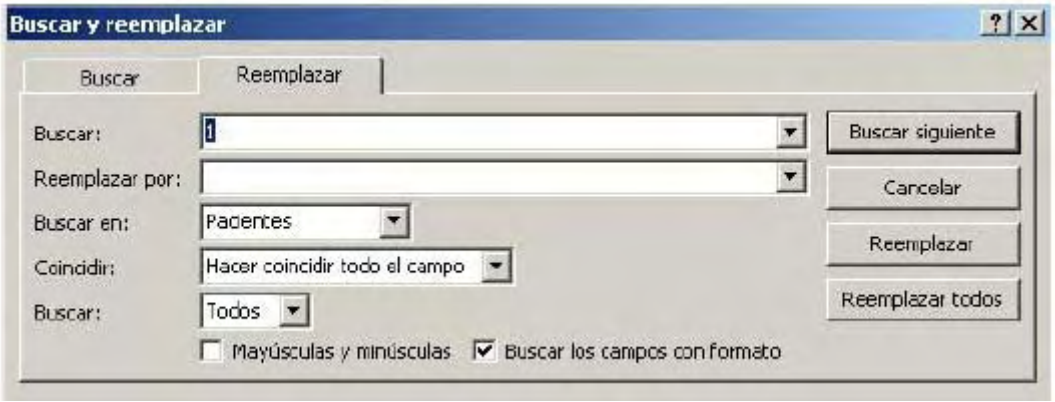
- Buscar los campos con formato: buscará sólo aquellos campos que coincidan exactamente con el formato utilizado en Buscar.

Una vez especificadas las características de la búsqueda se procede a realizarla pulsando: «Buscar siguiente» busca de uno en uno todos los registros, se podrá pulsar hasta que no haya más registros que coincidan con la petición de búsqueda.

6.7.3 Reemplazar datos

Esta opción es muy similar, sólo que no sólo busca los datos de registro en registro sino que además los sustituye por otro dato, o el mismo dato con modificaciones. El funcionamiento de esta función es muy similar al de Buscar.

Para reemplazar es necesario tener la tabla abierta y situarse sobre la columna donde se va a buscar el dato a sustituir. Se selecciona el menú «Edición|Reemplazar», a continuación se abrirá esta ventana:



En esta ventana primero se realiza la búsqueda y una vez encontrado el dato se sustituye. Casi todas las funciones de la ventana son las mismas, las únicas distintas son los botones de la derecha:

« Reemplazar » Reemplaza un registro encontrado por lo que se haya indicado en Reemplazar por. Reemplaza de uno en uno.

« Reemplazar todos » Reemplaza todos los registros que coincidan con las características de la búsqueda por lo indicado en Reemplazar por.

Cuando Access no encuentra más elementos que coincidan con el de búsqueda muestra un mensaje de aviso. Para terminar se pulsa « Aceptar ».

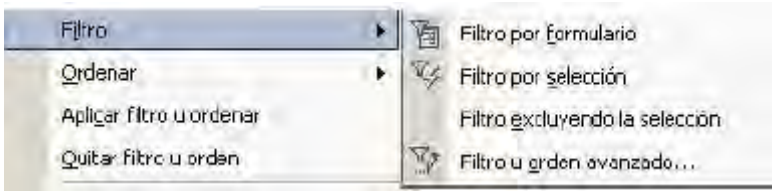
6.7.4. Filtros

Un filtro es una selección de algunos elementos de una tabla o de un formulario. Aplicando un filtro es posible ver dentro de una tabla sólo aquellos registros que cumplen unas condiciones. El resto de los registros no desaparecen, sólo están ocultos.

Access tiene tres tipos de filtros:

- Filtro por selección
- Filtro por formulario
- Filtro u orden avanzado

Para aplicar un filtro se selecciona el menú «Registros|Filtro»:



O bien se utilizan los botones de la barra de herramientas: « »

Para aplicar o desactivar un filtro se puede utilizar el botón « ». Aplicará el último filtro utilizado y desactivará el filtro que en ese momento esté activado.

6.7.4.1. Filtro por selección

Este filtro se aplica seleccionando dentro de la tabla el elemento que va a ser la condición de filtrado.

Cuando la tabla o el formulario no es muy grande es sencillo de utilizar.

En el ejemplo de los pacientes si se quiere a aplicar un filtro para mostrar sólo aquellos pacientes que se apelliden García, lo primero es situarse sobre cualquier registro que tenga el apellido García.

Después se pulsa el botón de «Filtro por selección» «  ». De la tabla aparecerán sólo aquellos registros en los cuales el campo apellidos sea García, el resto desaparecerán:

Para volver a ver todos los registros de la tabla bastará con pulsar sobre el botón de «quitar filtro» «  ».

6.7.4.2. Filtro por formulario

Es un filtro de mucha utilidad para tablas grandes y con un gran número de registros.

Tras pulsar sobre el botón de «filtro por formulario» «  », el menú y la barra de herramientas variarán:

Pulsando en la primera fila de cada columna aparecerá una flecha en la zona derecha de la celda. Pulsando sobre esta flecha se despliega la lista de todos los valores archivados en ese campo. Si desea dejar en blanco esta casilla se pulsa el botón «  ».

Si se prefiere se puede escribir directamente el valor exacto que se está buscando o la expresión cuyo resultado se desea utilizar como criterio.

 Una vez establecidos los criterios en todos los campos, este filtro se puede guardar, dentro de consultas.

 Se pueden volver a aplicar filtros almacenados en consultas.

 Borra lo contenido en la cuadrícula del campo, cuando no se quiera aplicar ninguna criterio en ese campo.

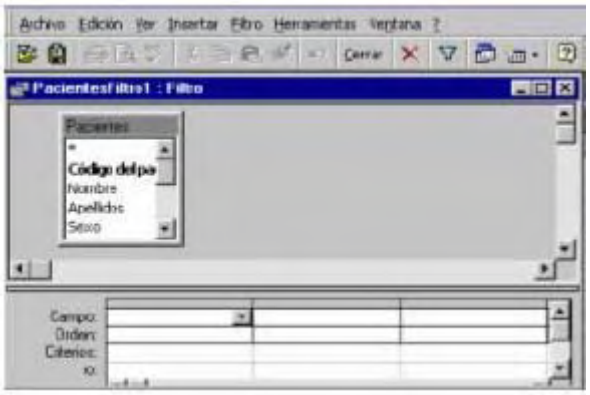
 Cierra esta ventana y vuelve a la tabla normal.

6.7.4.3. Filtro u orden avanzado

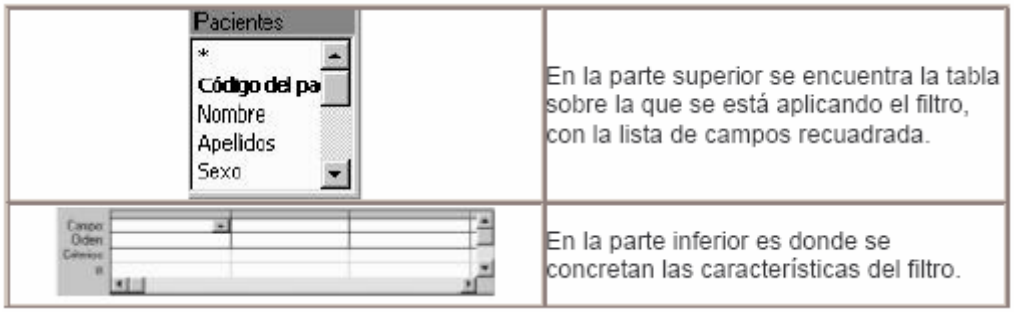
Este filtro permite determinar con mayor minucia las características del filtro. Es muy similar a una consulta, y se almacena en Consultas.

Para aplicar este filtro es necesario seleccionar el menú «Registros|Filtro» u orden avanzado ya que no hay ningún botón en la barra de herramientas con esta función.

Tras seleccionar este filtro se abrirá esta ventana:



Al igual que en el filtro por formulario, el menú y la barra de herramientas son propios de los filtros. El resto de la pantalla se divide en dos.



Para crear el filtro hay que detallar el campo, el orden a seguir y los criterios:

- Campo: para incluir el campo o los campos con los que se va a crear el filtro, se pueden emplear tres procedimientos distintos:

  1. Al hacer clic sobre la casilla campo aparece una flechita dentro, que despliega la lista de todos los campos de la tabla.

  2. Seleccionando la tabla en la zona superior y haciendo doble clic sobre el campo.

  3. Seleccionando la tabla en la zona superior y arrastrando el campo hacia la zona inferior.

  Se pueden utilizar tantos campos como se consideren necesario para crear el filtro. Para seleccionar todos los campos en una celda se selecciona el *, el filtrado se realizará en todos los campos de la tabla.

- Orden: establece el orden que se seguirá cuando se encuentre más de un registro que cumpla las condiciones del filtro. El orden puede ser ascendente, descendente o simplemente no seguir un orden.

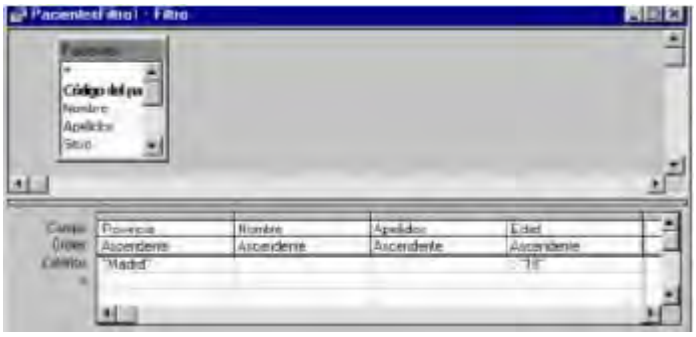
- Criterios: en esta casilla se escribe el criterio que debe seguir el filtro. Se puede escribir:

  - Un texto: se teclea y el programa lo pondrá automáticamente entre comillas.

  - Una expresión: utilizando los operadores "Entre...Y" o los operadores de comparación (<, >,<>, <=,>=).

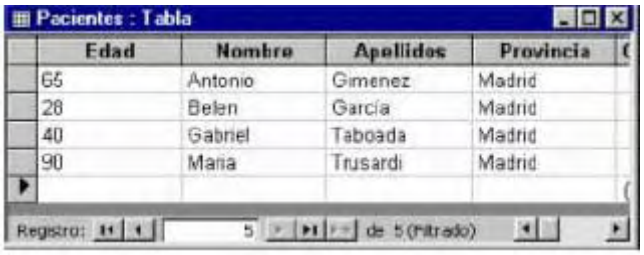
Siguiendo con la tabla Pacientes, se va a aplicar un filtro para ver únicamente los nombres de aquellos pacientes procedentes de Madrid y con menos de 18 años.

Lo primero que se debe hacer es seleccionar el menú «Registros|Filtro» u orden avanzado. En esta ventana se determinarán las condiciones del filtro tal y como aparecen en esta imagen:



Al escribir >18, el número 18 se ha escrito sin comillas y el programa las añadido automáticamente. Lo mismo sucede con el criterio de provincia, se escribe sin las comillas, y el programa las añade de forma automática.

Una vez se han establecido todos los criterios de filtrado se pulsa la tecla para aplicar el filtro «  ». Automáticamente se abrirá la tabla de Pacientes, pero sólo con los registros que cumplan las condiciones de filtrado.



Para volver a ver todos los registros de la tabla se vuelve a pulsar el botón «  ». Si se piensa que en un futuro puede ser practico volver a ver sólo estos datos, se puede almacenar el filtro para aplicarlo. Pero en la tabla, se estarán guardando todos los registros por mucho que se guarde el filtro. Porque aunque el resto de los registros no se vean, siguen estando ahí.

6.8. CONSULTAS A LA BASE DE DATOS

La consulta es una solicitud de información a la base de datos. Los datos mostrados pueden proceder de una sola tabla o de un conjunto de tablas. El resultado de la consulta es la "Hoja de respuestas dinámica"; en esta hoja se encuentra la información extraída de las tablas. Pero la información no se guarda en la hoja de respuestas, sino que sigue estando almacenada en las tablas.

En determinados tipos de consulta se puede modificar la información de las tablas, pero la consulta sigue siendo una forma de acceder a la tabla, no un objeto que almacene información. La consulta muestra lo que la tabla almacena según los criterios solicitados.

La consulta es un filtro avanzado, y funciona prácticamente de la misma forma. Lo único que los diferencia es que los filtros sólo se pueden activar desde una tabla o desde un formulario.

6.8.1. ¿Qué puede hacer una consulta?

- Elegir tablas: las consultas se pueden realizar sobre una sola tabla o sobre todas las tablas creadas en esa base de datos. De esta forma las combinaciones posibles para obtener información son muchas.

- Modificar los datos de las tablas: aunque las consultas no son tablas, dan acceso a ellas, y permite modificar, eliminar o añadir registros nuevos. También se puede utilizar una consulta para crear una nueva tabla que contenga registros de otra tabla o de varias tablas.

- Elegir uno o varios campos: al crear una consulta es posible especificar qué campo se desea ver.

- Seleccionar registros: una consulta se puede concretar hasta el punto de ver sólo un registro.

- Realizar cálculos: se pueden realizar cálculos con los campos mostrados en la consulta. Por ejemplo contar el número de registros seleccionados o acumular totales. Se crearán campos nuevos: campos calculados que contendrán el resultado del cálculo.

- Para crear nuevos formularios, informes o consultas: partiendo de los datos obtenidos en una consulta se pueden crear nuevos elementos.

6.8.2. Tipos de consultas

6.8.2.1. Consulta de selección

Es la más sencilla, se indican unos criterios para ver sólo lo que interesa de una tabla. Los datos aparecen en la Hoja de respuestas dinámicas, esta parece una tabla pero no lo es, sólo muestra los datos de una tabla o de varias tablas según los criterios de la consulta.

Aunque la hoja de respuestas dinámica no es una tabla se pueden introducir datos en las tablas a través de ella.

6.8.2.2. Consulta de tablas de referencias cruzadas

Presenta los datos con títulos en las filas y en las columnas; la apariencia es la de una hoja de cálculo. De esta forma se resume en muy poco espacio mucha información de una forma muy clara.

6.8.2.3. Consulta de parámetros

Cuando se ejecuta, muestra un cuadro de diálogo que solicita la información para recuperar registros o un valor que desea insertar en un campo. Se puede diseñar la consulta para que solicite más de un dato, por ejemplo, entre dos fechas.

6.8.2.4. Consulta de acciones

Es una forma de modificar registros de una o varias tablas a través de una sola operación. A través de este tipo de consulta también se puede crear una nueva tabla, eliminar o añadir registros, modificarlos...

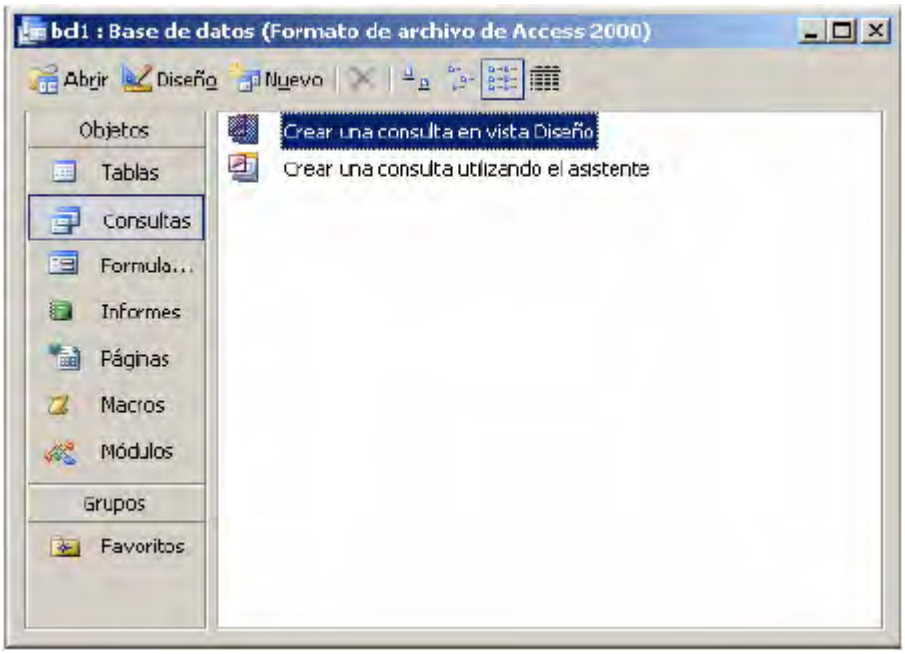
6.8.2.5. Consulta de paso a través o consulta SQL:

Envía instrucciones a una base de datos SQL.

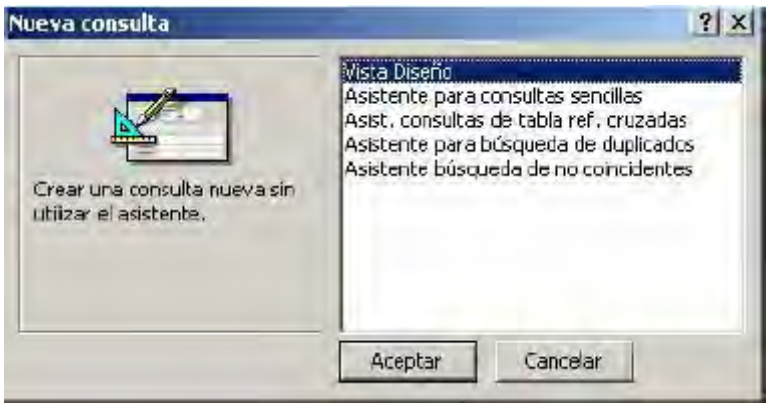
6.8.3. ¿Cómo crear una consulta?

Para crear una consulta hay que seleccionar la pestaña « 

» de la ventana de la base de datos. Si no se está en la ventana Base de datos, para volver a ella se pulsa el botón «base de datos» «  ». En la ventana Consultas puede aparecer algún elemento creado si se ha guardado algún filtro:



Para crear una consulta nueva se pulsa el botón «Nuevo». Aparecerá esta ventana:



Estas son las opciones que ofrece Access para crear una consulta:

1. Vista diseño: permite realizar una consulta sin la ayuda del asistente.

2. Asistente para consultas sencillas: crea una consulta automáticamente, según los campos seleccionados.

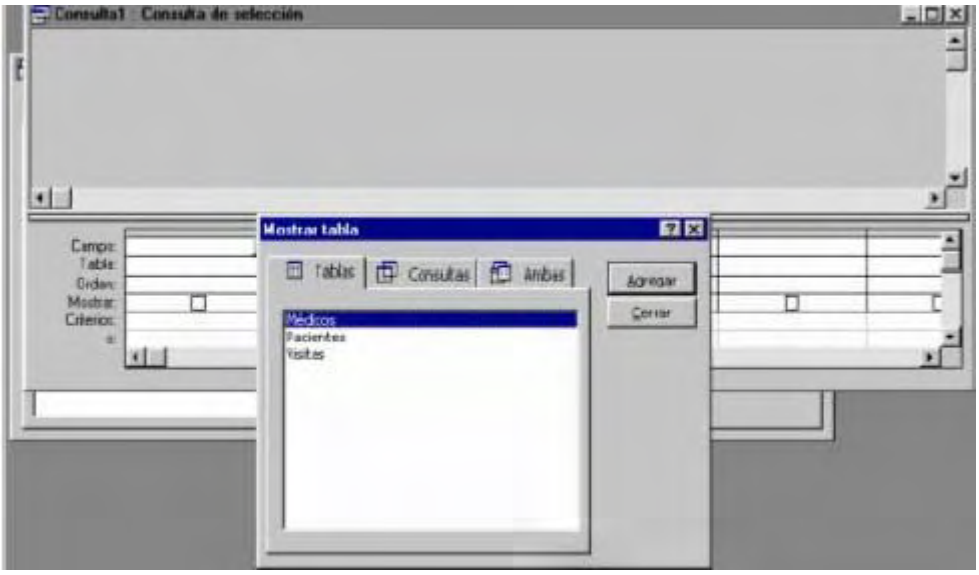
3. Asistente para consultas de referencias cruzadas: crea una consulta que muestra los datos con un formato compacto, parecido al de una hoja de cálculo.

4. Asistente para consultas destinados a buscar duplicados: crea una consulta en la que se buscan registros con valores duplicados en un campo.

5. Asistentes para consultas destinados a buscar no-coincidentes: crea una consulta que busca registros que no tienen registros relacionados en otra tabla.

6.8.4. Crear una consulta sin asistentes

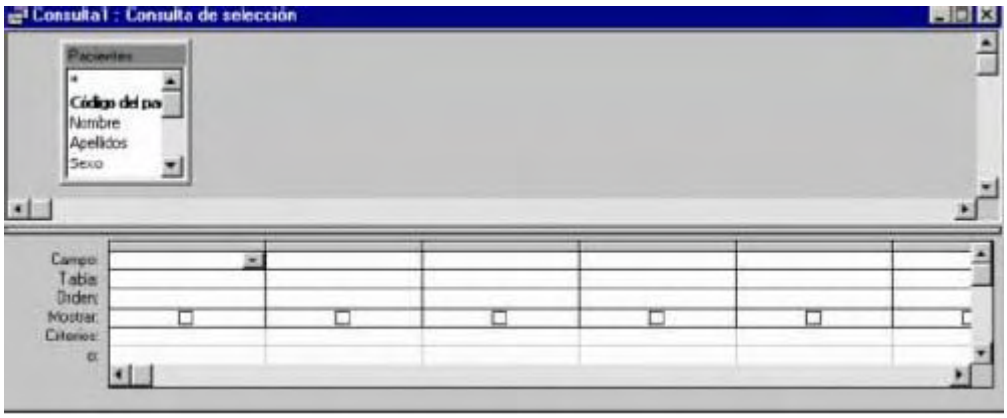
Dentro de la ventana de Nueva Consulta se selecciona la opción Vista Diseño y se pulsa Aceptar. Automáticamente se abrirá la ventana de la consulta e inmediatamente otra ventana donde se debe elegir en que tabla/s se va a realizar la consulta.



En este ejemplo se va a elegir la tabla Pacientes. Tras seleccionar la tabla se pulsa «Agregar». Se pueden seleccionar más tablas, e incluso consultas. Una vez se hayan elegido los elementos sobre los que se quiera realizar la consulta se pulsa «Cerrar».

Si tras cerrar se quiere volver a abrir la ventana de agregar Tablas se pulsa el botón «  » o se selecciona el menú «Consulta|Mostrar Tabla».

El aspecto de la ventana de una consulta es este:



Al igual que en los filtros, la ventana se divide en dos secciones:

- La superior: muestra los elementos sobre los que se va a realizar la consulta.
- La inferior: muestra los criterios que se van a aplicar en la consulta a la tabla o consulta seleccionada.

El proceso de creación de los criterios es muy similar al de los filtros. En las consultas hay un elemento más, que da la opción de que un criterio de selección se vea o no. Esta opción es «Mostrar».

6.8.4.1. Campo

En esta casilla se pueden seleccionar todos los campos de la tabla (con el «*») o de uno en uno, seleccionando cada uno en una columna.

La forma de incluir el nombre de un campo en esta casilla es:

1. Arrastrando el nombre del campo desde la sección superior.
2. Haciendo doble clic en la tabla de la sección superior.
3. Haciendo clic sobre la casilla campo y pulsando sobre la flecha que aparece. Se desplegará una lista de los campos para seleccionar.

6.8.4.2. Tabla

En esta casilla figura la tabla de la que procede el campo seleccionado en esa columna. Esta opción es muy importante cuando se trabaja con campos de varias tablas.

6.8.4.3. Orden

Ascendente, descendente o sin orden. Este orden se aplicará a los registros que se obtengan en la consulta según la columna en la que se esté indicando el orden. El criterio se establece en un campo y se ordenan los resultados en función del campo que se quiera.

Para seleccionar el tipo de orden que se quiere se hace clic sobre la casilla Orden, aparecerá una flecha en la zona derecha de la casilla. Al pinchar sobre la flecha aparece un menú con los tipos de orden aplicables a la consulta.

6.8.4.4. Mostrar

Esta casilla tiene un pequeño cuadrado, al hacer clic dentro de este cuadrado se está indicando al programa que se muestre ese campo. Esto tiene sentido cuando se quiere indicar un criterio más para restringir la búsqueda, pero no se quiere mostrar en el resultado de la búsqueda. Si la casilla esta activada « ☑ » este criterio aparecerá. Si no está activada « ☐ » ese criterio se utilizará en la búsqueda pero no será visible en la tabla dinámica.

6.8.4.5. Criterios

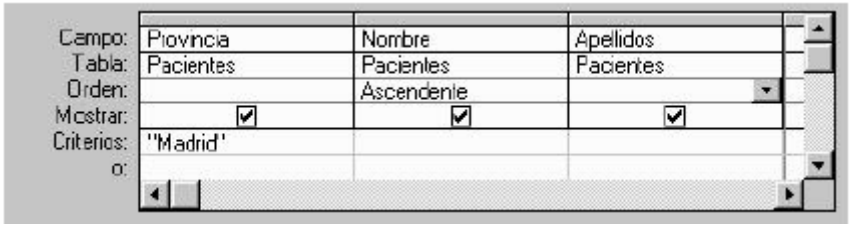
En esta casilla se introduce la condición que debe cumplir un campo para que el registro aparezca en la respuesta a la consulta.

Cuando se ejecuta la consulta, el programa analiza la expresión de la casilla criterios. Dentro del campo se buscan todos los valores que coincidan con el criterio. Los criterios pueden introducirse en uno o más campos de una consulta. O incluso introducir varios criterios en un mismo campo.

No es necesario rellenar todas las casillas en cada columna. Todo depende de lo que se quiera pedir al programa.

Una forma más sencilla de ver una búsqueda es un ejemplo. Con la tabla de Pacientes se va a realizar el siguiente ejemplo: Buscar los pacientes que vivan en Madrid.

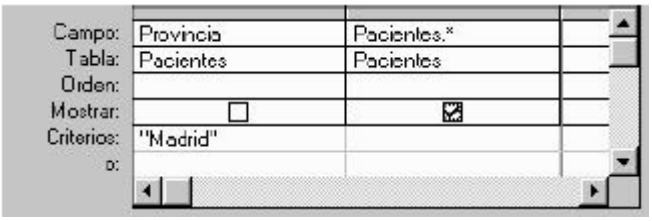
En este ejemplo sólo se establece un criterio en un campo: Provincia, y el criterio Madrid. De todos aquellos registros en los cuales el campo Provincia sea Madrid se le pide que muestre: el nombre y el apellido de los pacientes, además del criterio, que no está oculto. Si no se le añaden más campos no mostrará más datos del registro que cumple el criterio.



Para ejecutar la consulta se pulsa el botón «  » o se selecciona el menú «Consulta|Ejecutar».

El programa devolverá el o los registros que cumplan con ese criterio, si es que los hay. Para mostrar los resultados se abrirá la Hoja de respuesta dinámica en la Vista Diseño. En esta hoja aparecerán únicamente los datos solicitados en la Consulta.

Para modificar los criterios de una consulta se cambia a la Vista Diseño. Si se quiere que se muestren todos los datos de los pacientes procedentes de Madrid habrá que hacerlo de la siguiente forma.



Se ha establecido el criterio Madrid en el campo Provincia. A continuación en lugar de detallar cada campo en una columna se han seleccionado todos los campos en una sola celda con el asterisco. Se ha ocultado el criterio, pero se muestran todos los campos de la Tabla Pacientes. Se ejecuta la consulta pulsando « » el resultado será este:

6.8.5. Establecer criterios

Dentro de la casilla criterio se escribe una expresión, bien de texto, numérica o numérica con operadores. Si no se incluye ningún operador con la expresión (texto o número) el programa actúa como si el operador fuera "=", y sólo habrá resultado en la búsqueda si existe un registro idéntico a la expresión del criterio.

Gracias a los operadores, además de palabras, se pueden imponer otro tipo de condiciones a las búsquedas dentro de los registros.

6.8.5.1. Rangos de valores

Para buscar un rango de valores dentro de un campo se utilizan estos operadores:

“Entre...Y”: por ejemplo para seleccionar los pacientes mayores de 18 años pero menores de 50. En la casilla Criterios se escribirá: Entre 18 Y 50. <,>,>=, <=, <>: operadores de comparación.

6.8.5.2. Lista de valores

Se puede poner más de un criterio, especificando cada uno de ellos. La redacción de esta expresión deberá ser de la siguiente forma: el operador En seguido de una lista de valores entre paréntesis, separando los valores con el punto y coma. Por

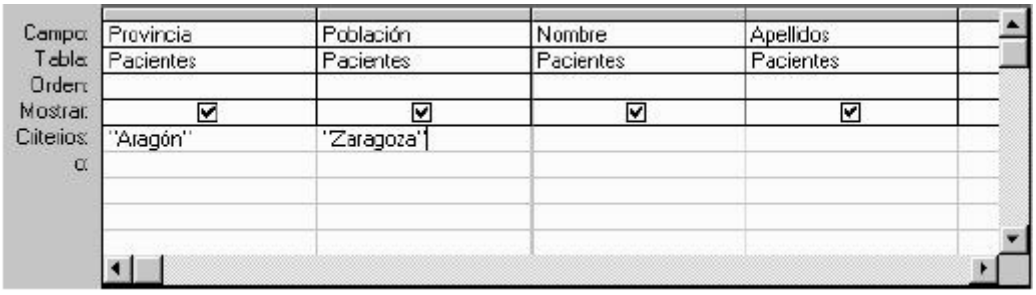
ejemplo para seleccionar todos los pacientes de tres provincias, en el campo Provincia se utilizaría este criterio:

En ("Madrid";"Aragón"; "País Vasco")

De esta forma el programa seleccionará todos aquellos registros que en el campo provincia tengan uno de estos tres valores.

6.8.5.3. Varios criterios

En diferentes campos: criterio “Y”: cuando se escriben varios criterios en el mismo renglón el programa buscará un registro que cumpla todos los criterios.



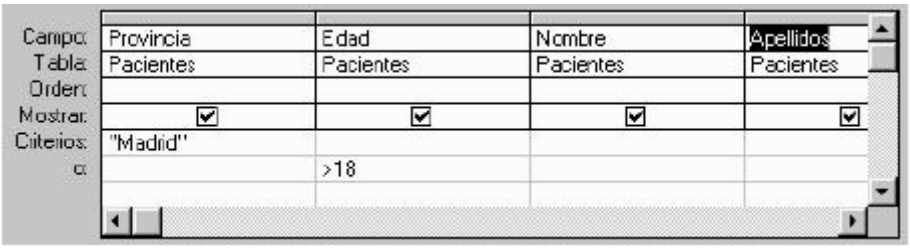
Campo:	Provincia	Población	Nombre	Apellidos
Tabla:	Pacientes	Pacientes	Pacientes	Pacientes
Orden:				
Mostrar:	☑	☑	☑	☑
Criterios:	"Aragón"	"Zaragoza"		
o:				

El criterio “Y “se puede utilizar en un mismo campo: >80 Y <100

En un mismo campo: criterio “O”: cuando se pone un criterio en la casilla Criterio, otro en la casilla “o”, y si se quiere más criterios en las filas de debajo. El programa buscará un registro que cumpla al menos uno de los criterios.

Campo:	Población	Nombre	Apellidos
Tabla:	Pacientes	Pacientes	Pacientes
Orden:			
Mostrar:	☑	☑	☑
Criterios:	"Sevilla"		
o:	"Jaén"		
	"Córdoba"		
	"Málaga"		
	"Granada"		

Este criterio también se puede utilizar en distintos campos:



Campo:	Provincia	Edad	Nombre	Apellidos
Tabla:	Pacientes	Pacientes	Pacientes	Pacientes
Orden:				
Mostrar:	☑	☑	☑	☑
Criterios:	"Madrid"			
o:		>18		

Seleccionará los registros que o el campo provincia sea Madrid, o bien el campo edad sea mayor de 18.

6.8.5.4. Los comodines

Se utilizan para buscar datos genéricos, que empiezan por una letra, que terminan por otra...

- * representa cualquier número de caracteres, por ejemplo: todos los nombres que terminen por "ez": *ez. Otro ejemplo: buscar las personas que tienen un nombre compuesto y uno de los dos nombres es "Luis". Se debe escribir el siguiente criterio: Como *Luis* o Como * + Luis + *. Access agregará las comillas.

- ? representa un solo carácter, por ejemplo Jiménez/Gimenez: ?imenez.

- # representa cualquier dígito en la posición especificada. Por ejemplo: 12#45,el programa buscará un registro en el cual los dos primeros dígitos sean 12 y los dos últimos 45.

Al introducir una expresión con uno de estos tres operadores, Access añadirá automáticamente el operador "Como".

6.8.5.5. Selección de registros con datos o sin datos

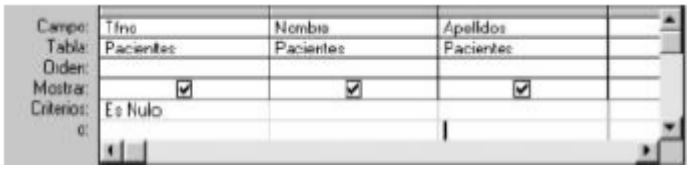
Se puede seleccionar un registro por el criterio de si en el campo hay un dato, o si está vacío.

Porque contiene algún dato: “Es Negado Nulo”

Pacientes que tienen teléfono:

Porque no contiene ningún dato: vacío: “Nulo o es Nulo”

Pacientes que no tienen teléfono:



6.8.5.6. Fecha actual

Se pueden seleccionar los registros que tengan la fecha actual. Por ejemplo en una empresa para seleccionar los pedidos que haya que entregar ese día.

En criterios se escribe Fecha()

6.8.5.7. Condición variable

Para hacer una condición variable; que realice una pregunta cada vez que se abra o ejecute la consulta, se coloca la pregunta entre corchetes [ ] debajo del campo que se desea variar. (ver consultas con parámetros)

6.8.5.8. Criterios con cálculo

Dentro de un criterio se puede realizar un cálculo haciendo referencia a otro campo. Por ejemplo en la base de datos de una empresa de venta de material de construcción se va a consultar:

Campo:	Nombre cliente	Cantidad	Precio Unidad	Importe: [Cantidad]*[Precio Unidad]
Tabla:	Ventas	Ventas	Ventas	
Orden:				
Mostrar:	☑	☑	☑	☑
Criterios:				>[Cantidad]*10
o:				

Se pide que liste a aquellos clientes cuyo Importe sea mayor a la cantidad por 10.

Se ha creado el campo importe ya que no existía, que es un campo calculado.

Los campos van siempre entre corchetes.

Dentro del criterio se ha realizado una operación y en función del resultado se han seleccionado o no los registros.

Consulta1 : Consulta de selección

Nombre cliente	Cantidad	Precio Unidad	Importe
Agroman	1000	50	50000
Indur	130	500	65000
Eco	5000	30	150000
Inda	1000500	40	40020000
Empo	50	18000	900000

Registro: 1 de 5

6.8.6. Guardar las consultas

Siempre es recomendable que se prueben varias veces las consultas para asegurarse que el resultado es el esperado.

Una vez se comprueba que el resultado es el que se esperaba, hay que pensar en guardar o no la consulta. Todo depende de la frecuencia con la que se vaya a usar esa consulta. Si se va a usar más veces es conveniente guardarla para no tener que rediseñarla la próxima vez.

La consulta se puede guardar desde la vista Diseño o la vista Hoja de datos. Se selecciona el botón guardar «  » o el menú «Archivo|Guardar».

6.8.7. Campos calculados

Se puede crear un campo que realice una operación con varios campos de una misma tabla.

En el ejemplo de le empresa de venta de materiales de construcción se ha creado el campo calculado Importe.

Pasos para crear un campo calculado:

1. Se selecciona una columna en blanco, se escribe el nombre del nuevo campo seguido de dos puntos Importe.

2. Se escribe la operación, cuando se hace referencia a un campo este debe ir entre corchetes [ ].

   Importe: [Cantidad] *[Precio Unidad]

Si sólo se va a operar con un campo, se puede seleccionar el campo de la lista de campos y al añadir un símbolo de operación: /*-+ el programa añadirá el corchete al campo y pondrá un nombre al nuevo campo. El nombre será Expr, Expr1...

6.8.8. Crear consultas con asistentes

6.8.8.1. Asistente para consultas sencillas

El Asistente para consultas sencillas crea consultas que recuperan datos de los campos especificados en una tabla o consulta, o en varias tablas o consultas.

Si se desea, el asistente también puede sumar, contar y obtener el promedio de los valores de grupos de registros o de todos los registros y puede calcular el valor mínimo o máximo de un campo. No obstante, no es posible limitar los registros recuperados mediante el establecimiento de criterios.

6.8.8.2. Asistente para consultas de referencias cruzadas

Una consulta de tabla de referencias cruzadas calcula totales resumidos basándose en los valores de cada fila y columna. Calcula una suma, una media, un recuento u otros tipos de totales de los registros y luego agrupa el resultado en dos tipos de información: uno hacia abajo, en el lado izquierdo de la hoja de datos y otro a lo largo de la parte superior.

6.8.8.3. Asistente para consultas de buscar duplicados

Con este asistente se puede determinar si existen registros duplicados en una tabla o determinar qué registros de una tabla comparten el mismo valor. Por ejemplo, se pueden buscar valores duplicados en un campo de dirección para determinar si existen registros duplicados para el mismo paciente.

También se pueden buscar valores duplicados en un campo de población para todos los pacientes de una misma ciudad.

Puede ser una herramienta útil si se han importado datos desde otra base de datos, ya que permite depurarlos.

6.8.8.4. Asistente para buscar registros no coincidentes

Mediante este Asistente se pueden buscar registros en una tabla que no tenga registros relacionados en otra tabla. Por ejemplo, puede buscar pacientes que no hayan realizado ninguna visita.

También es útil si se han importado datos desde otra base de datos, ya que permite depurarlos.

6.8.9. Consultas con parámetros

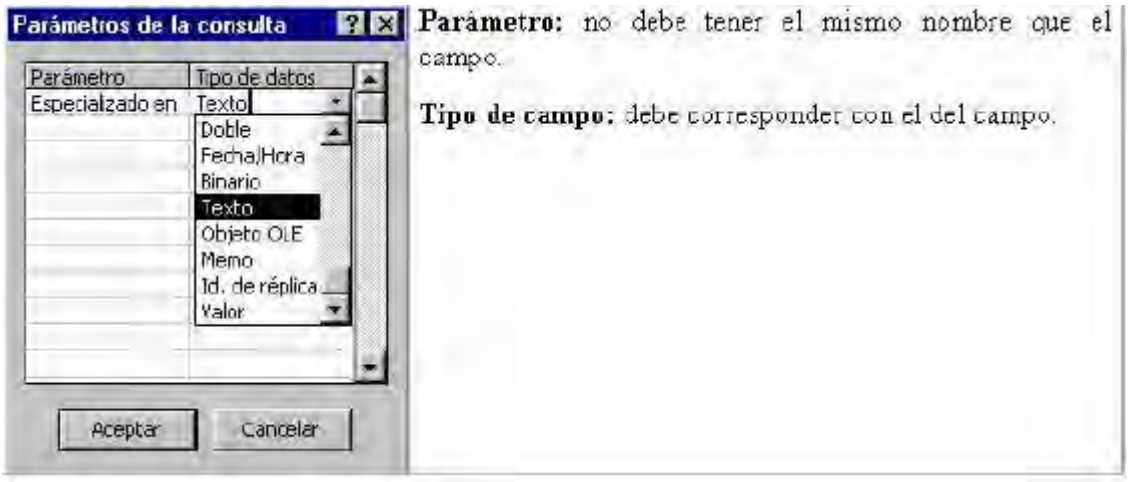
Se recurre a ellas cuando se quiere hacer una consulta que permita pedir un dato antes de ejecutarla y buscar según ese dato.

Una consulta que tiene uno o varios parámetros necesitará que éstos se indiquen para buscar datos en la tabla.

Por ejemplo, para realizar una consulta sobre los nombres de los médicos que trabajan en un determinado departamento de un hospital, se tendrá que hacer una consulta por especialidad utilizando el procedimiento habitual. Si se utiliza una consulta diseñada con parámetros se podrá decir qué especialidad se busca cada vez que se ejecute una nueva consulta.

El proceso es muy parecido al de una consulta normal. Se selecciona la tabla, se seleccionan los campos y los criterios. En este caso se va a utilizar la tabla de médicos.

Después de crear una consulta normal se selecciona el menú «Consulta|Parámetros». Se abrirá esta ventana:



Se crean los parámetros deseados, cada parámetro tiene un nombre y un tipo de datos. El nombre no puede ser igual al nombre de un campo de la tabla y el tipo de datos debe corresponder con el tipo de datos del campo por el que se va a buscar.

En el campo por el que se va a buscar hay que añadir el criterio. En este caso el criterio es el nombre del parámetro entre corchetes:

Campo:	Especialidad	Nombre	Apellidos
Tabla:	Médicos	Médicos	Médicos
Orden:			
Mostrar:	☑	☑	☑
Criterios:	[Especializado en]		
o:			

Cuando se ejecute la consulta aparecerá una ventana que pide el valor del parámetro.

Dentro del recuadro en blanco se escribe la especialidad sobre la que se desea obtener la lista de médicos que trabajan allí. Después se pulsa «Aceptar». Esta será la hoja de respuestas dinámica:

Especialidad	Nombre	Apellidos
Medicina Intem	Gonzalo	Latorre
Medicina Intem	Pedro	Pérez
Medicina Intem	Gerardo	Martínez

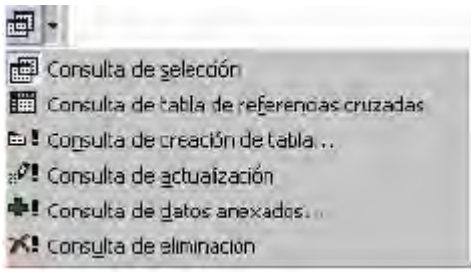
6.8.10. Consultas basadas en más de una tabla

Se crean de la misma forma que las anteriores, sólo que a la hora de agregar tablas se puede agregar más de una. De esta manera es posible reflejar las relaciones entre las tablas, y recuperar los registros relacionados de las dos tablas.

De cada tabla se seleccionan los campos que se necesitan, arrastrándolos y pegándolos en las celdas de la consulta. Se agregan los criterios que se crean convenientes, de la misma manera que en las consultas basadas en una sola tabla.

6.8.11. Consultas avanzadas

Hasta ahora la consulta que se ha visto ha sido la consulta de selección, una consulta muy sencilla. Para seleccionar otro tipo de consulta más compleja se debe seleccionar el botón tipo de consulta:



6.8.11.1. Consulta de creación de tabla

Este tipo de consulta puede ser usada cuando se quiera crear una tabla nueva a partir de registros provenientes de ejecutar una consulta. La nueva tabla no heredará las propiedades de los campos, ni la clave principal que tuviera la tabla origen.

6.8.11.2. Consulta de actualización

Este tipo de consultas pueden ser usadas cuando se quiera actualizar varios registros de una tabla, de una sola vez. Se pueden ver los registros a actualizar antes de ejecutar la consulta y que sean modificados permanentemente.

6.8.11.3. Consulta de datos anexados

Este tipo de consultas pueden ser usadas cuando se quiera añadir registros a una tabla de otra que ya contenga algunos. Entonces, se podrán agregar datos que estaban en otra tabla de Access o bien en otros formatos de Tabla, como pueden ser DBase, Paradox.

6.8.11.4. Consulta de eliminación

Este tipo de consultas pueden ser usadas cuando se quiera eliminar varios registros de una tabla que cumplan determinados criterios. Es posible ver los registros que van a ser eliminados antes de ejecutar la consulta.

6.9. ¿CÓMO RELACIONAR TABLAS?

6.9.1. Diferencia de una base de datos relacional

La diferencia de las bases de datos relacionales con respecto a una base de datos plana consiste en que los datos sólo se introducen una sola vez en una tabla, pero gracias a las relaciones pueden aparecer en las tablas que se quiera.

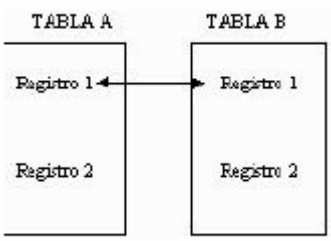
Cualquier modificación sólo hay que realizarla una sola vez y automáticamente se realizará en todas las demás tablas. De este modo se ahorra mucho tiempo, espacio y exactitud en los datos que siempre estarán actualizados independientemente de la tabla en la que estemos.

6.9.2. Tipos de relaciones

Existen tres tipos de relaciones, que se explican a continuación. Más adelante se verá cómo quedan guardadas relaciones de este tipo en Access.

6.9.2.1. Relación uno a uno

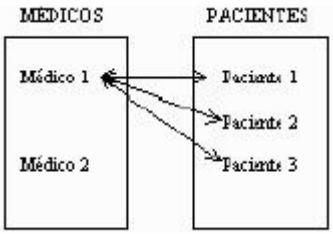
Cada registro de la tabla A se relaciona sólo con un registro de una tabla B y cada registro de la tabla B se relaciona sólo con un registro de la tabla A.



Relaciones de este tipo se almacenan guardando en la tabla el identificador de la otra tabla con la que mantiene la relación.

6.9.2.2. Relación uno a varios

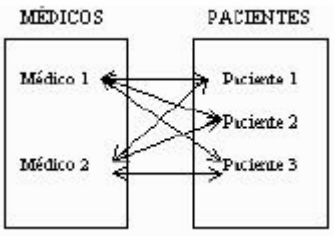
Cada registro de la tabla A está relacionado con varios registros de la tabla B y cada registro de la tabla B está relacionado con un sólo un registro de la tabla A.



Aplicando esto al ejemplo2 de este manual, una relación de este tipo se daría entre la tabla pacientes y la tabla médicos, ya que el mismo médico se hará cargo de varios pacientes. Un solo registro de la tabla de médicos se relaciona con varios registros de la tabla de pacientes. Pero un registro de la tabla de pacientes sólo se relaciona con un registro de la tabla médicos.

6.9.2.3. Relación varios a varios

Cada registro de la tabla A puede estar relacionado con más de un registro de la tabla B y cada registro de la tabla B puede estar relacionado con más de un registro de la tabla A.



Si existiera una base de datos con dos tablas: médicos y pacientes, con una relación directa entre ellos, un médico atendería muchos pacientes y un mismo paciente podría ser atendido por varios médicos. Varios registros de la tabla de médicos se relacionarían con varios registros de la tabla de pacientes.

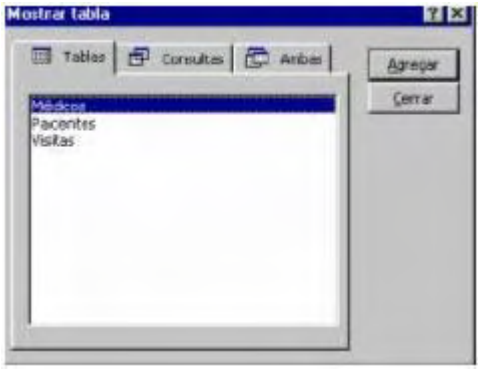
Relaciones de este tipo se almacenan creando una tabla especial donde se colocan los identificadores de cada tabla y otros campos que puedan ser de utilidad, por ejemplo la fecha, la hora, comentarios acerca de la visita médica, etc. En el ejemplo 1 esta tabla especial es la tabla Visitas, donde aparecen tanto el código del médico como el del paciente.

6.9.3. Crear relaciones ente dos tablas

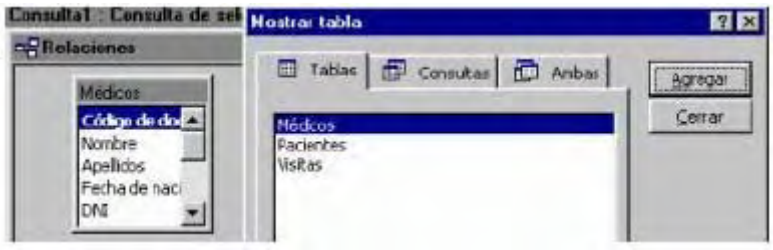
Para crear una relación entre las tablas de una base de datos primero es necesario cerrar todas las tablas. Con las tablas abiertas no se puede crear o modificar una relación. Para poder utilizar la integridad referencial será necesario que las tablas no tengan ningún registro.

Desde la ventana Base de datos, se pulsa el botón «  » o se selecciona el menú «Herramientas|Relaciones». Automáticamente se abrirá la ventana Relaciones totalmente vacía.

Para añadir las tablas que van a estar relacionadas se pulsa el botón «Mostrar tabla» o se selecciona el menú «Relaciones|Mostrar Tabla». Aparecerá una ventana con el listado de las tablas:

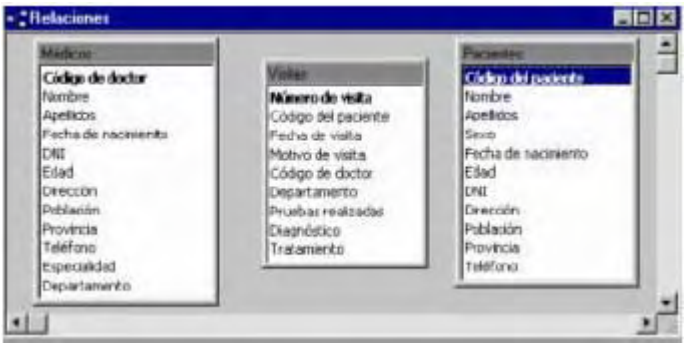


Se seleccionan aquellas «tablas/consultas» que van a formar parte de una relación y se pulsa «Agregar». Después de pulsar «Agregar» en la ventana Relaciones aparecerá la tabla en un recuadro con todos los campos.



Cuando ya no se quieran agregar más tablas/consultas se pulsa el botón «Cerrar». Quedará abierta únicamente la ventana Relaciones.

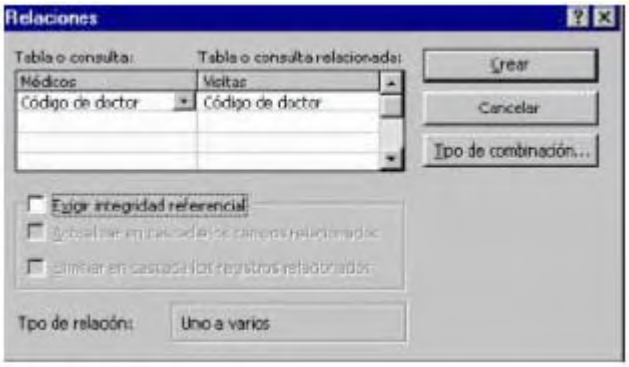
En este caso se van a incluir las tres tablas del ejemplo 1: Médicos, Pacientes y Visitas.



Para crear las relaciones entre estas tres tablas se relacionará primero médicos con visitas y luego pacientes con visitas.

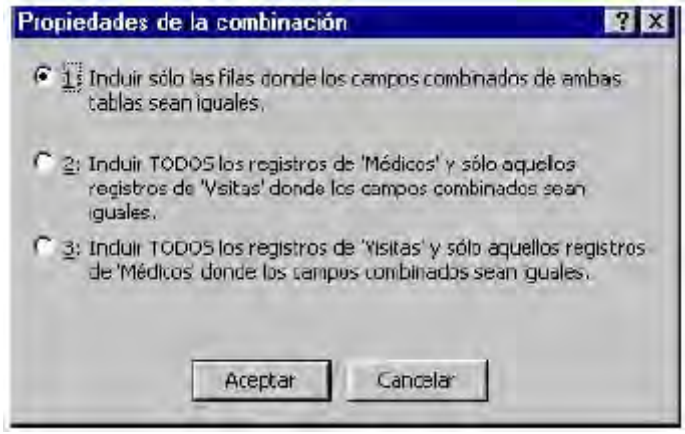
Para relacionar médicos con visitas el campo en común es el código del doctor. Este dato está almacenado en la tabla médicos, por tanto, el campo se arrastrará desde médicos hasta Visitas. Para arrastrar el campo primero se selecciona, se hace clic, y sin soltar el botón del ratón se arrastra hasta situar el cursor sobre el campo Código del doctor de la tabla Visitas. Al arrastrar el campo el cursor se convertirá en un rectángulo pequeño.

Tras arrastrar el campo se abrirá esta ventana:

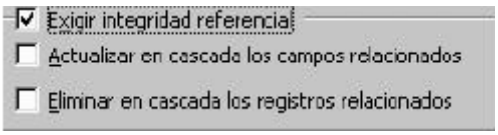


Dentro de la ventana hay dos columnas en las dos debe estar un campo con un contenido similar. No importa la coincidencia del nombre sino del contenido. Médicos es la tabla primaria en esta relación (es la que contiene los datos) y Visitas es la tabla secundaria (tomará los datos de médicos a través del campo común).

Si se pulsa el botón Tipo de combinación... se abrirá una ventana explicando los tres tipos de combinaciones. Automáticamente aparece seleccionada la primera combinación. En este ejemplo se puede dejar así.



Exigir integridad referencial



La integridad referencial son unas normas que mantienen la coherencia de datos entre dos tablas relacionadas. Estas normas son:

1. No puede haber registros en la tabla secundaria que no estén en la primaria.

2. No se puede borrar un registro de la tabla principal si hay registros en la secundaria.

Para poder exigir integridad referencial en una relación de uno a varios es necesario que:

1. El campo relacionado de la tabla principal sea la clave principal.

2. Los campos contengan el mismo tipo de datos (si es autonumérico-numérico).

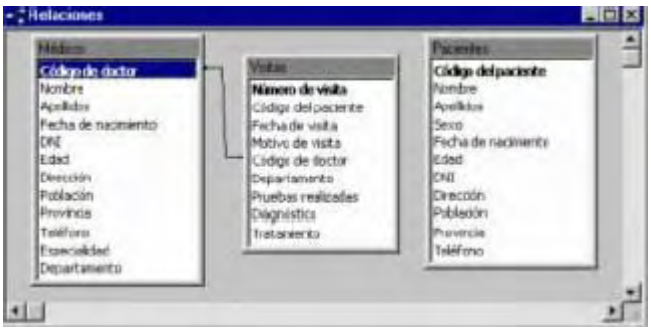
3. Ambas tablas deben pertenecer a la misma base de datos.

Access verificará que los campos cumplen todas las condiciones para que haya integridad referencial. Si no se cumplen todas las condiciones no permitirá que esa relación tenga integridad referencial.

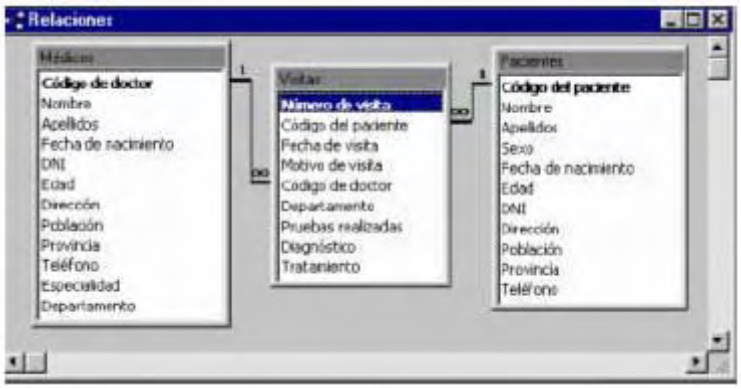
- Actualizar en cascada los campos relacionados se está indicando que si se modifica el valor de un campo desde un lado de la relación automáticamente se actualicen en todos los registros relacionados.

- Eliminar en cascada los registros relacionados si se borra un registro de un lado de la relación se borrarán automáticamente todos los registros que estaban relacionados con él.

Cuando ya se han especificado las características de la relación se pulsa el botón Crear. Entre las dos tablas relacionadas aparecerá una línea. Esta línea simboliza la relación entre las dos tablas. Si la relación cumple la integridad referencial la línea será más gruesa.



A continuación se creará la relación entre Pacientes y Visitas. Y se exigirá integridad referencial en las dos relaciones. Para exigir la integridad referencial se hace doble clic sobre la línea de relación, se volverá abrir la ventana de la relación.



Una vez se ha terminado de crear las relaciones entre las tablas se guardan. Para guardar se selecciona el botón guardar o el menú «Archivo|Guardar». Después de guardar ya se puede cerrar la ventana de relaciones. Si se cierra antes de guardar, se abrirá un mensaje de aviso.

Access permite guardar las relaciones en la Base de Datos. Esto será muy importante para que siempre que se lleven a cabo modificaciones en los datos se tenga en cuenta que las relaciones están presentes entre los mismos y no se puedan infringir las reglas de consistencia vigentes.

6.10. LOS INFORMES

Un informe es un objeto de Access. Los informes no guardan información, sólo son una presentación gráfica de los datos contenidos en tablas o los hallados en consultas. Esta presentación gráfica está orientada a la impresión de los datos. El diseño puede ser en columnas o en etiquetas, dependiendo de la utilidad que se le vaya a dar al impreso.

6.10.1. Diferentes formas de crear un informe

Dentro de la ventana de la base de datos se selecciona la pestaña de «Informes» « 

» . Para crear un informe nuevo se hace clic sobre el botón «Nuevo». Para modificar el diseño de un informe ya creado se selecciona «Diseño». Para ver cómo se imprimirá el informe se selecciona «Vista Previa».

Tras seleccionar «Nuevo» aparecerá la ventana dónde se indican los diferentes modos de creación de un informe:

1. Vista Diseño: permite crear manualmente un informe sin ayuda de los asistentes.

2. Asistente para informes: el asistente guía al usuario para la creación de varios modelos de informes.

3. Autoinforme columnas: es una forma automática de crear un informe en columnas. Sólo hay que seleccionar la tabla o la consulta de la cual van a extraerse los datos. Los nombres de los campos aparecen en la columna izquierda y a la derecha el dato. En cada registro vuelven a aparecer los nombres de los campos.

4. Autoinforme tabular: esta opción crea de forma automáticamente un informe. El diseño es en filas y columnas, como en una hoja de cálculo. En la primera fila aparecen los nombres de los campos y en el resto de las filas los datos.

5. Asistente para gráficos: es un asistente que crea informes gráficos. El asistente ayuda a elegir los campos a incluir en el gráfico y el tipo de gráfico que se quiere.

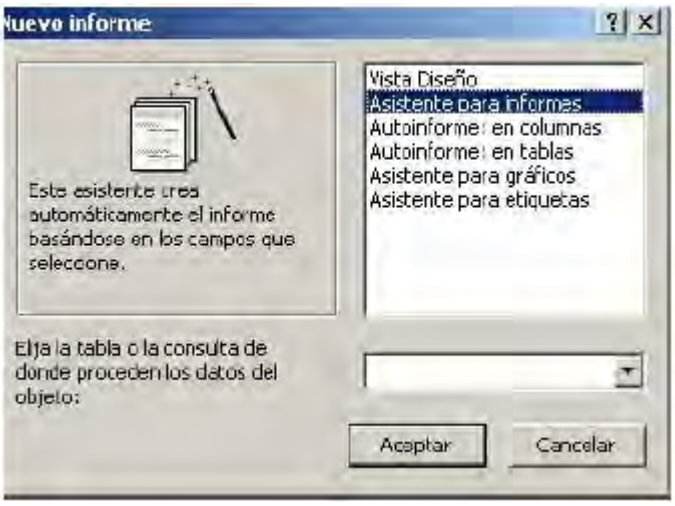
Con este asistente se seleccionarán campos que se desea que pasen a formar parte de un gráfico donde, por ejemplo, se pueden hacer comparaciones entre datos visualmente.

6. Asistente para etiquetas: ayuda a la creación de informes tipo etiquetas. Con este formato se pueden confeccionar etiquetas postales. El asistente ayuda a la elección de los campos a incluir en la etiqueta y al diseño de la misma.

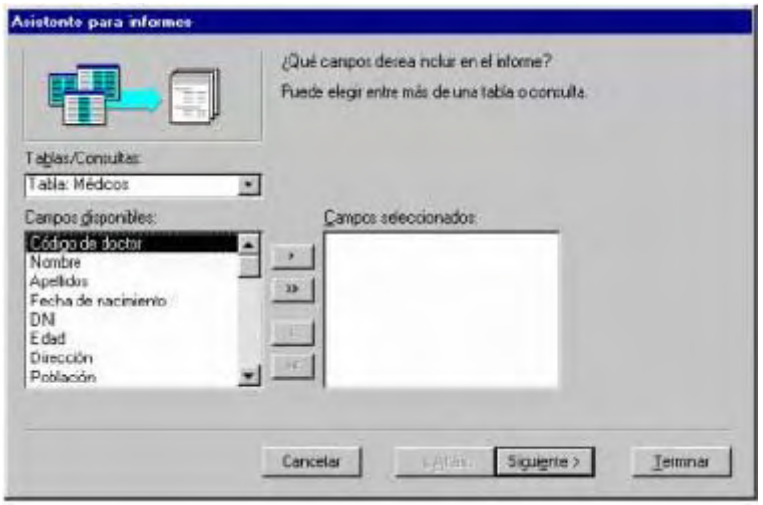
6.10.2 Asistente para informes

Con este asistente se pueden crear informes genéricos con diseños variados.

Tras haber seleccionado «Asistente para informes» en la ventana de «nuevo Informe», se debe seleccionar la tabla o la consulta con la que se va a elaborar el informe:



Se pulsa «Aceptar».



En esta ventana se seleccionan los campos que se quiere que aparezcan en el informe. Si se quiere en esta ventana se puede variar de tabla o incluso tomar campos de diferentes tablas.

Para seleccionar los campos se utilizan estos botones:

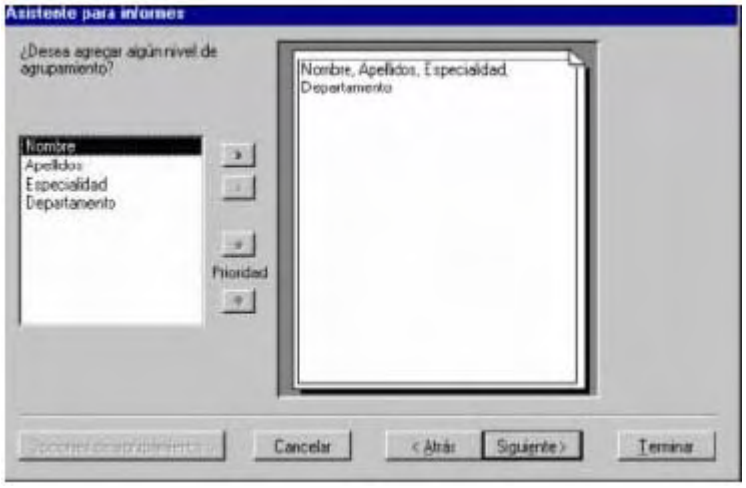
- Seleccionar un campo

» Seleccionar todos los campos

‹ Deseleccionar un campo

« Deseleccionar todos los campos

Una vez ya se han seleccionado los campos que aparecerán en el informe se pulsa Siguiente:



En esta ventana se especifican los niveles de agrupamiento. Por ejemplo agrupar los registros por la especialidad. Para añadir o quitar un agrupamiento se utilizan las flechas que hay entre los dos recuadros.

- Recuadro izquierdo: muestra los campos seleccionados para el informe.
- Recuadro derecho: muestra los campos del informe agrupados según se haya seleccionado. Si no se toca nada los campos aparecerán sin agrupación, tal y como están en la imagen superior.
- Botones de agrupamiento:

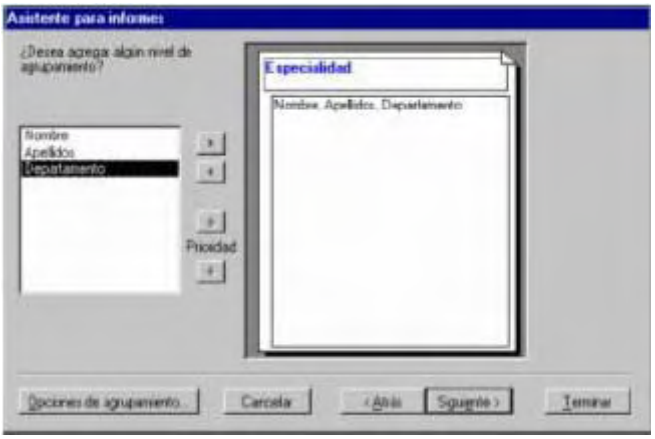
  › Sitúa en el nivel superior el campo seleccionando, creando grupos en el recuadro derecho

  ‹ Elimina el campo creado y vuelve a situar ese campo con los demás

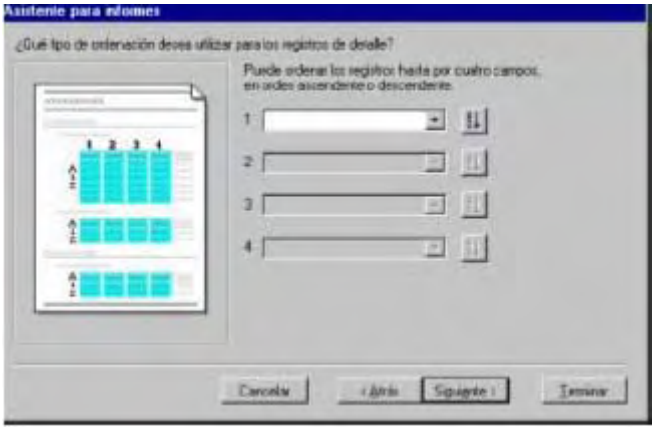
  ▲ Aumenta la prioridad del grupo seleccionado

  ▼ Disminuye la prioridad del grupo seleccionado

Para agrupar los campos del informe por especialidad se selecciona el campo especialidad en el recuadro izquierdo y se pulsa «›». La ventana quedará así:

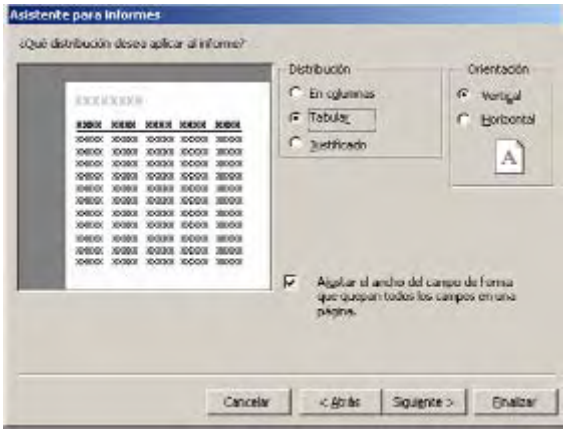


Una vez determinado el agrupamiento de los campos se hace clic sobre «Siguiente»:



En esta ventana se establece si se quiere modificar el orden en el que aparecerán los registros. Se pueden ordenar por uno o por varios campos de forma ascendente o descendente. En los campos en blanco se selecciona el nombre del campo y pulsando la tecla «  » variará a «  », varía el orden a aplicar, ascendente o descendente.

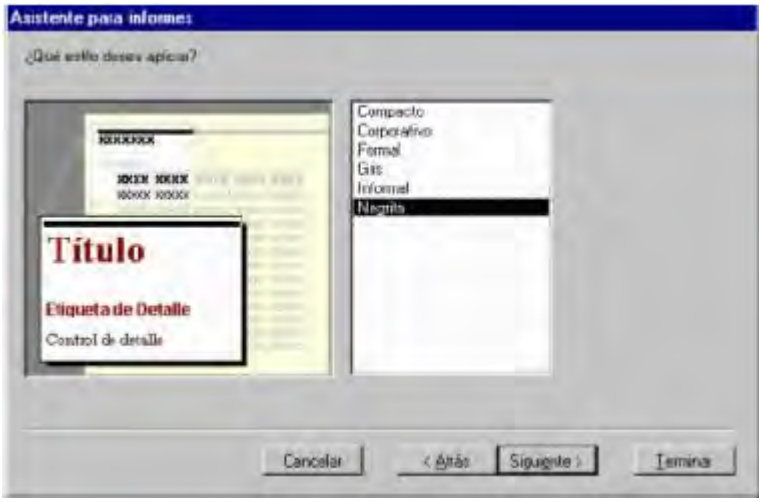
Tras haber determinado el orden de los registros se hace clic sobre «Siguiente» para continuar.



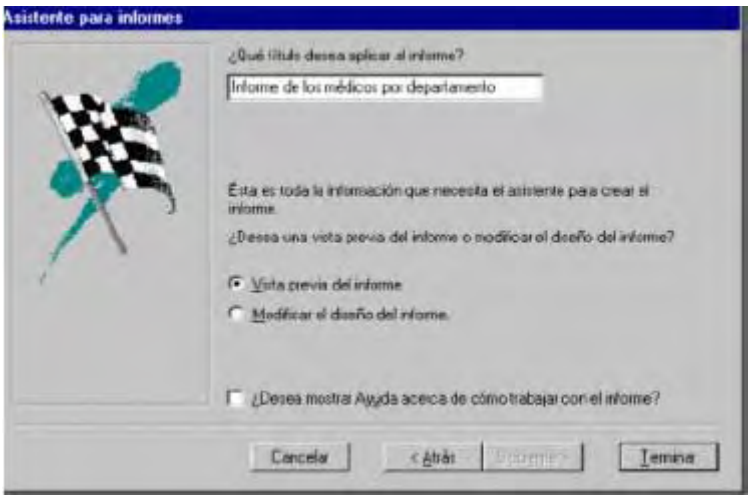
En esta ventana se selecciona el tipo de distribución del informe. Seleccionando cada tipo de distribución la pantalla izquierda mostrará gráficamente de qué se trata. Para un informe sólo se puede utilizar un tipo de distribución. También se puede modificar la orientación de la página.

Con la opción «Ajustar el ancho del campo...» cabrán todos los campos en una sola línea del ancho de la página, pero es posible que no se vea el contenido completo. Esto se puede intentar mejorar, bien cambiando posteriormente el tamaño de la letra, o bien seleccionando menos campos.

Tras pulsar «Siguiente» aparecerá esta ventana para seleccionar un diseño predeterminado de los que presenta el programa.



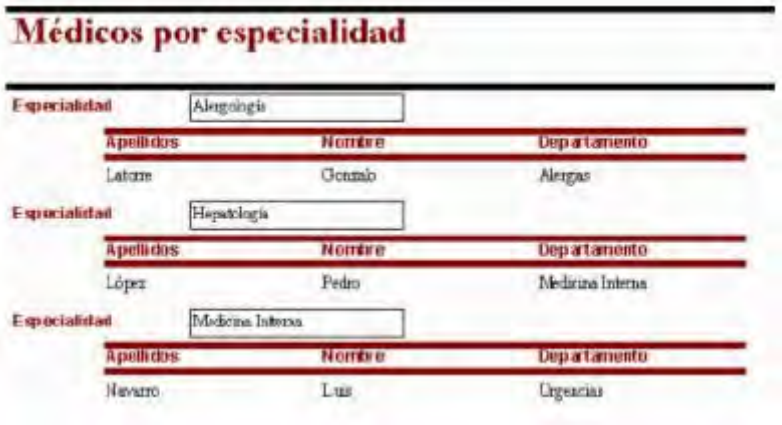
Según se vayan seleccionando los nombres del recuadro derecho el aspecto del recuadro izquierdo variará. Este diseño no tiene por qué ser definitivo, se podrá modificar posteriormente. Se selecciona un diseño y se pulsa «Siguiente».



En esta última ventana se le asigna un nombre al informe que se ha creado y se decide si se quiere ver el aspecto del informe; Vista previa o si se quiere modificar el diseño del informe.

Una vez se seleccionan las opciones que se quieran utilizar se pulsa «Terminar». La creación del informe ha finalizado.

Si se ha seleccionado ver la vista previa del informe lo que se verá será esto. Para modificarlo se pulsa en el botón de «cambio de vista» «  ».



Si se ha seleccionado modificar el diseño del informe la ventana que se abrirá será ésta. Para ver cómo va quedando se pulsa el botón «cambio de vista» «  ».

Si una vez confeccionado y cerrado el informe se desea hacer algún cambio, se selecciona la pestaña «Informes» y se pulsa el botón «Diseño».

Desde Vista Diseño se hacen todos los cambios referentes al aspecto del informe. Se pueden modificar las fuentes de los campo, añadir logotipos, texto.

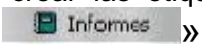
Para cambiar un campo se selecciona el campo haciendo clic sobre él. A continuación, se busca en la hoja de propiedades la que se quiera modificar.

Para ver cómo quedará el informe impreso se utiliza la Vista preliminar.

6.10.3. Asistente para etiquetas

Las etiquetas es un tipo de informe de Access que permite imprimir información de una tabla o consulta en forma de etiquetas.

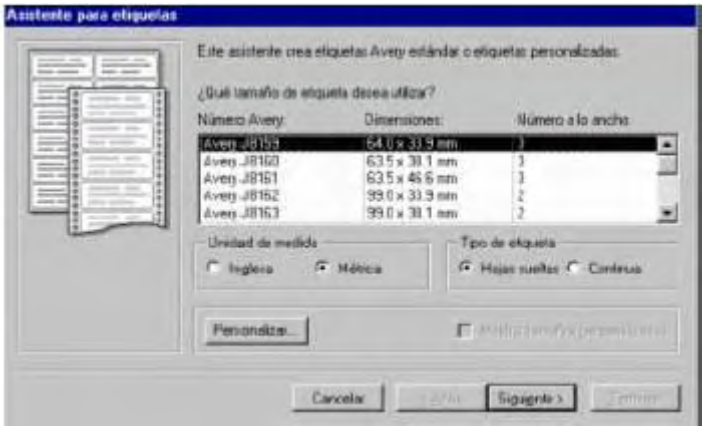
La creación de etiquetas utilizando el asistente resulta muy sencilla. Al igual que con el asistente para informes sólo hay que ir completando las pantallas que aparezcan.

Para comenzar a crear las etiquetas hay que situarse en la carpeta informes pulsando la pestaña « 

» y luego hacer doble click en el botón «Nuevo» y aparecerá la siguiente ventana:

Donde se elige el tipo de informe que se va a crear y el modo de crearlo.

Para crear etiquetas con el asistente se selecciona «Asistente para etiquetas». La tabla que se va a utilizar es la de Pacientes. Tras seleccionar el asistente y la tabla se pulsa «Aceptar».



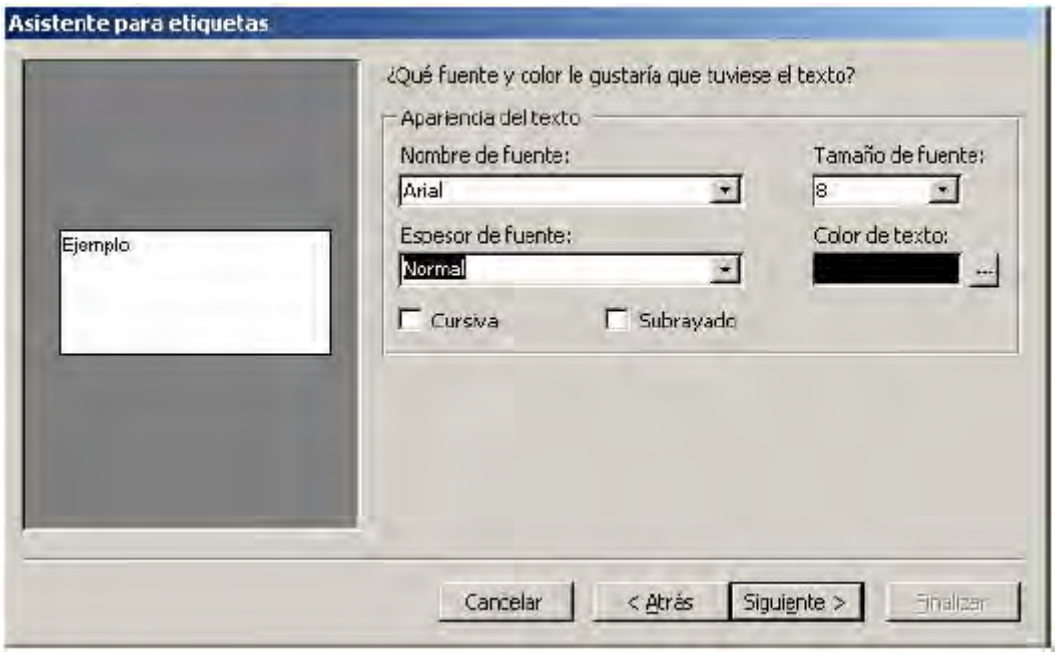
En esta ventana se debe elegir el tamaño de las etiquetas. Todo depende del papel de impresión de etiquetas que se tenga. Dependiendo del modelo habrá una, dos o tres etiquetas por fila. En la caja de las etiquetas suele venir el nombre del modelo, que está en la primera columna.

En la segunda columna se encuentran las dimensiones de la etiqueta, para seleccionar por medida. Estas dimensiones se pueden ver en dos tipos de medidas, inglesa o métrica.

Dependiendo de la que esté seleccionada en Unidad de medida.

Además del tamaño de las etiquetas se puede seleccionar si el papel en el que se imprimirán las etiquetas es continuo o si son hojas sueltas.

Tras seleccionar todo se pulsa «Siguiente».



En esta ventana se selecciona el formato del texto de las etiquetas. El nombre de la fuente, el tamaño, si es normal, fina, negrita, y el color de la letra.

A continuación se seleccionarán los campos que aparecerán en la etiqueta.

Se pueden seleccionar todos los campos o sólo aquellos que interesen.

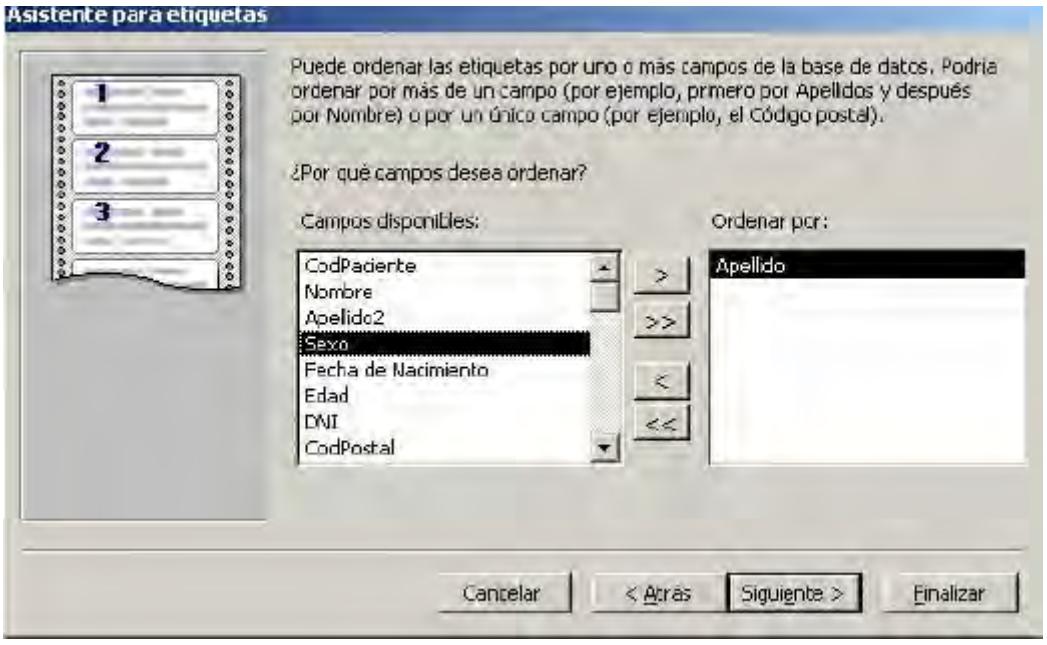
Para seleccionar un campo se hace doble clic sobre el campo o bien se utiliza el botón que hay entre los dos recuadros.

Los campos aparecerán todos en la misma línea si no se pulsa la tecla «intro» después de insertar un campo.

En este ejemplo se han seleccionado estos campos, con este orden:

En la primera línea se han insertado dos campos. En las siguientes se ha pulsado «intro» después de añadir cada campo.

El orden que presenten las etiquetas puede ser el que se utilizó cuando se introdujeron los datos u otro distinto.



En esta ventana Access permite seleccionar el campo por el que se ordenarán las etiquetas para la impresión. En este caso se ha decidido ordenar las etiquetas de los pacientes por el apellido.

En esta ventana se indica el nombre del informe y el paso que se dará a continuación, bien ver la vista previa de las etiquetas, o bien modificar el diseño de las etiquetas en Vista Diseño.

Tras seleccionar todo se pulsa «Finalizar».

6.11. LAS MACROS

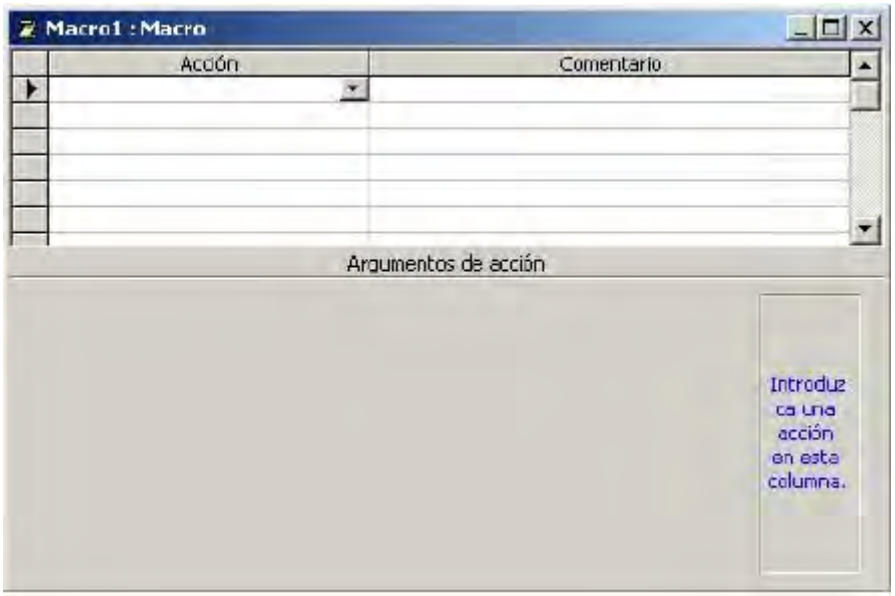
Una macro es un objeto más de la base de datos. Este objeto ejecuta unas instrucciones concretas de forma automática, en el orden determinado por el usuario.

Una macro puede ser: enviar a imprimir un informe de forma automática, abrir automáticamente un formulario, o una hoja de datos de una tabla, o ejecutar automáticamente una consulta.

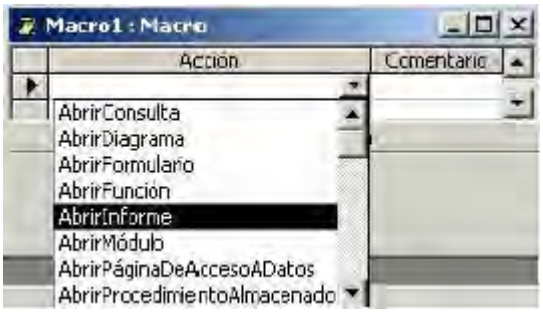
Las macros se ejecutan a través de botones de comando, estos botones se pueden incluir en un informe o un formulario.

6.11.1. ¿Cómo se crea una macro?

Las macros se crean exactamente igual que los demás objetos de Access. Se selecciona la pestaña «Macros» y se pulsa «Nuevo». Se abrirá la ventana para definir las macros.



Para comenzar la macro se debe hacer clic sobre la flecha que aparece en la celda de «Acción»:

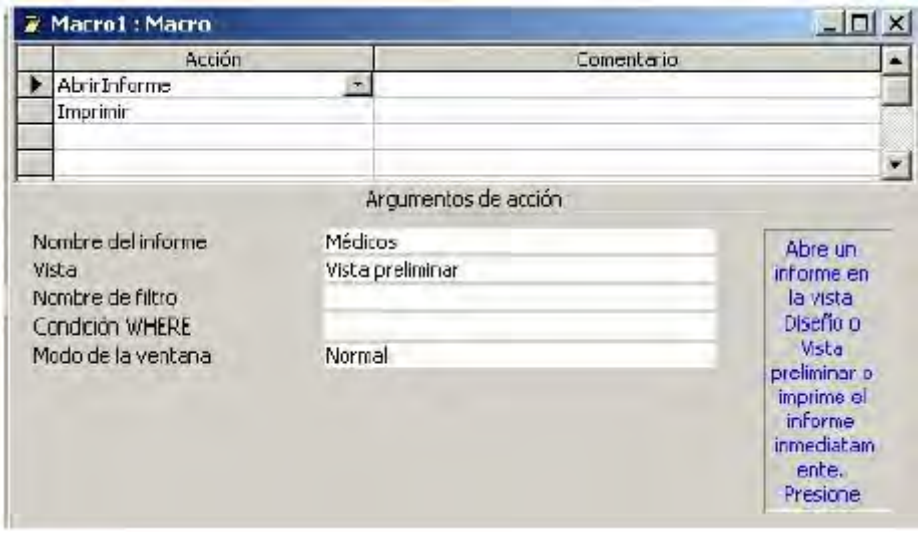


En la columna «Acción» se indican en cada fila las diferentes acciones a ejecutar en secuencia.

En la columna «Comentario» se puede escribir una explicación de lo que realiza esa acción.

En la parte inferior de esta pantalla, una vez se ha seleccionado una acción, aparecen los «Argumentos de Acción». Dependiendo de la acción seleccionada tendrá más o menos argumentos.

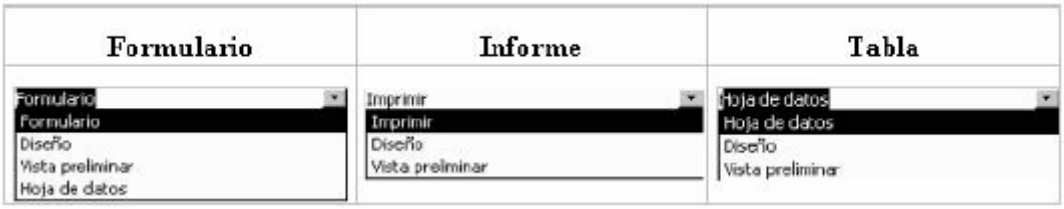
Cada acción tiene un número de argumentos con valores distintos. Un argumento con el mismo nombre puede actuar de forma distinta dependiendo del objeto sobre el que actúe.



6.11.1.1. Argumentos de acción

Nombre del informe/Formulario/Tabla: objeto de la base de datos sobre el que recae la acción. Al hacer clic en el campo aparecerá una flecha en la zona derecha, si se pulsa, se despliega la lista de todos los informes que haya en la base de datos (si la Acción es Abrir Informe).

- Vista: indica la vista en la que se activará el objeto seleccionado. Variará según el objeto del que se trate, un informe tiene tres vistas, pero un formulario tiene cuatro vistas.

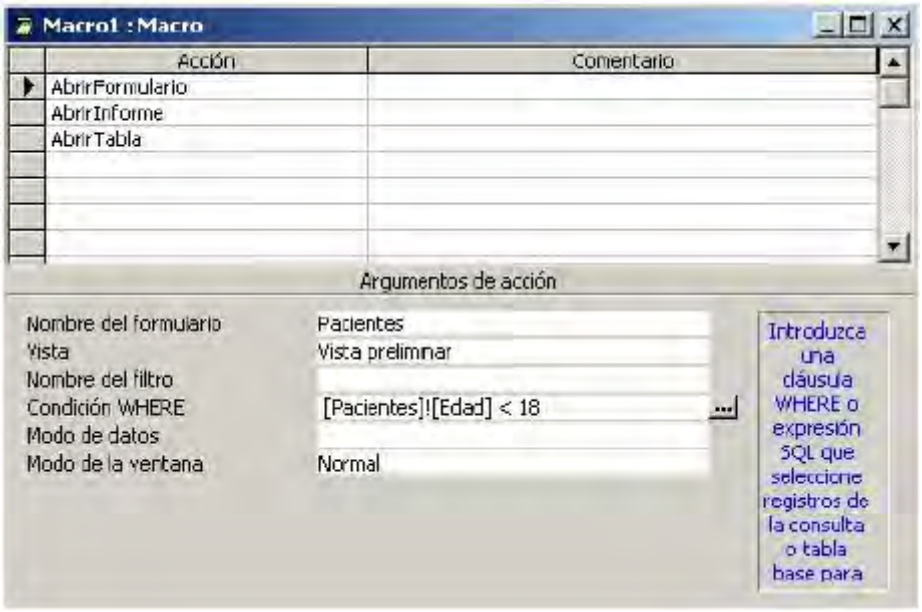


- Nombre del filtro: con el filtro se utilizan criterios de selección, de esa forma sólo se obtendrán determinados datos por sus características. Se puede utilizar un filtro o una consulta existente, con la condición de que coincida en número de campos con el objeto que se desee abrir.

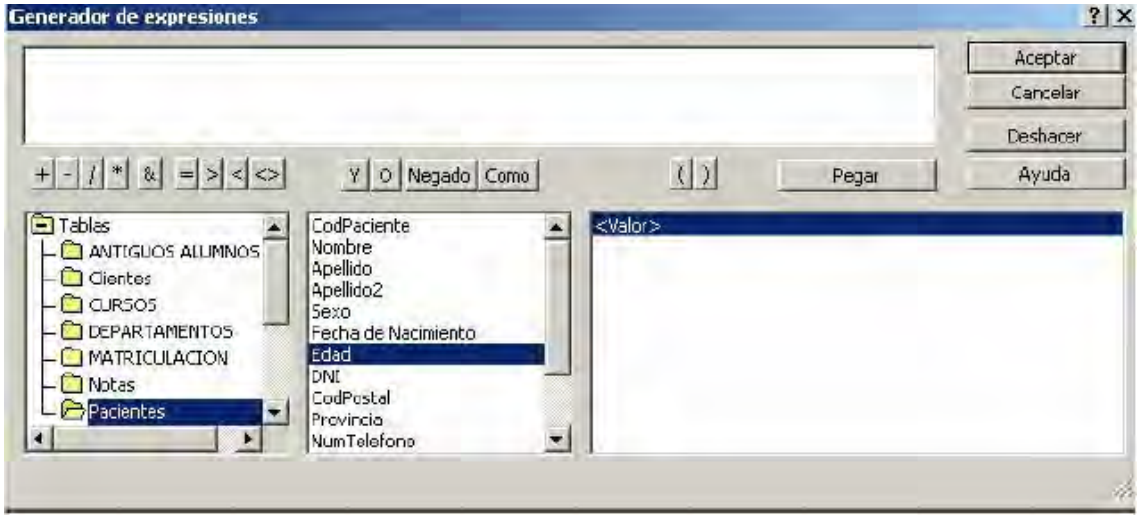
- Condición “WHERE”: es una cláusula del lenguaje SQL, que permite seleccionar determinados registros de una tabla o consulta.

Si se seleccionan nombre de filtro y la condición WHERE entonces se ejecutará sobre el resultado de aplicar el filtro.

Un ejemplo de la condición WHERE:



Para redactar la condición se hace clic sobre los puntos suspensivos, se abrirá el «Generador de expresiones»:



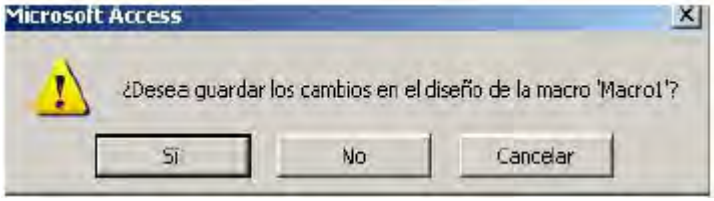
Para seleccionar un campo de una tabla o de un formulario se abre la carpeta que corresponda. Luego se selecciona el campo y después la condición que debe cumplir para que sea seleccionado.

- Modo de la ventana: esta opción es el modo en el que se trabajará el objeto seleccionado. Los posibles modos son normal, oculta, icono, diálogo. El modo normal muestra el objeto según las propiedades normales.

6.11.2. Guardar una macro

Una vez se han determinado todas las acciones sucesivas, y los argumentos de acción de cada acción, se guarda la macro pulsando el botón «guardar» «  ». Aparecerá la siguiente ventana en la que se debe adjudicar un nombre a la macro creada:

Si no se pulsara el botón «guardar» y se cerrara la ventana de la macro Access preguntará si se desea almacenar la macro o no:



6.11.3. Ejecutar una macro

Desde la ventana de la base de datos, seleccionando la pestaña «macros», aparecerán todas las macros creadas.

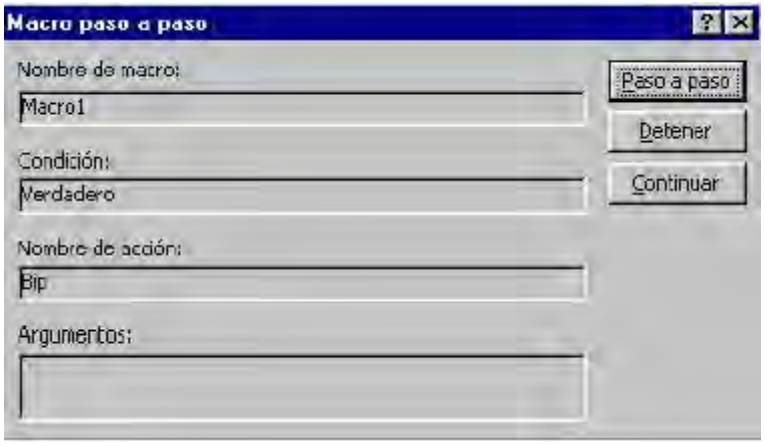
Se selecciona la macro que se desee ejecutar.

Se pulsa el botón «Ejecutar». Automáticamente se ejecutará la macro correspondiente.

Si aún se está en la ventana de la macro se puede ejecutar seleccionando el botón ejecutar «  » o el menú «Macro|Ejecutar».

Tras pulsar ejecutar todas las acciones se realizarán de una sola vez. Si se desea observar primero cada uno de los pasos de la macro se selecciona el botón «paso a paso» «  » y después se pulsa el botón «Ejecutar» «».

Aparecerá cada acción en una ventana con sus argumentos y la posibilidad de ver en cada paso lo que se ha puesto en la Macro, para depurarla en el caso de que tenga algún error.



- Paso a paso: pulsando este botón se ejecuta la acción que se muestre en ese momento en la ventana, y muestra la ventana de la siguiente acción.

- Detener para detener la macro. Se cerrará la ventana de Paso a paso y se cancelará la macro.

- Continuar: continúa ejecutando la macro pero sin mostrar la ventana de Paso a paso antes de realizar cada acción.

Para modificar una macro, se selecciona la carpeta macros en la ventana de la base de datos. Se selecciona la macro a modificar y se hace clic en el botón Diseño. Entonces se quita o modifica lo que se crea conveniente.

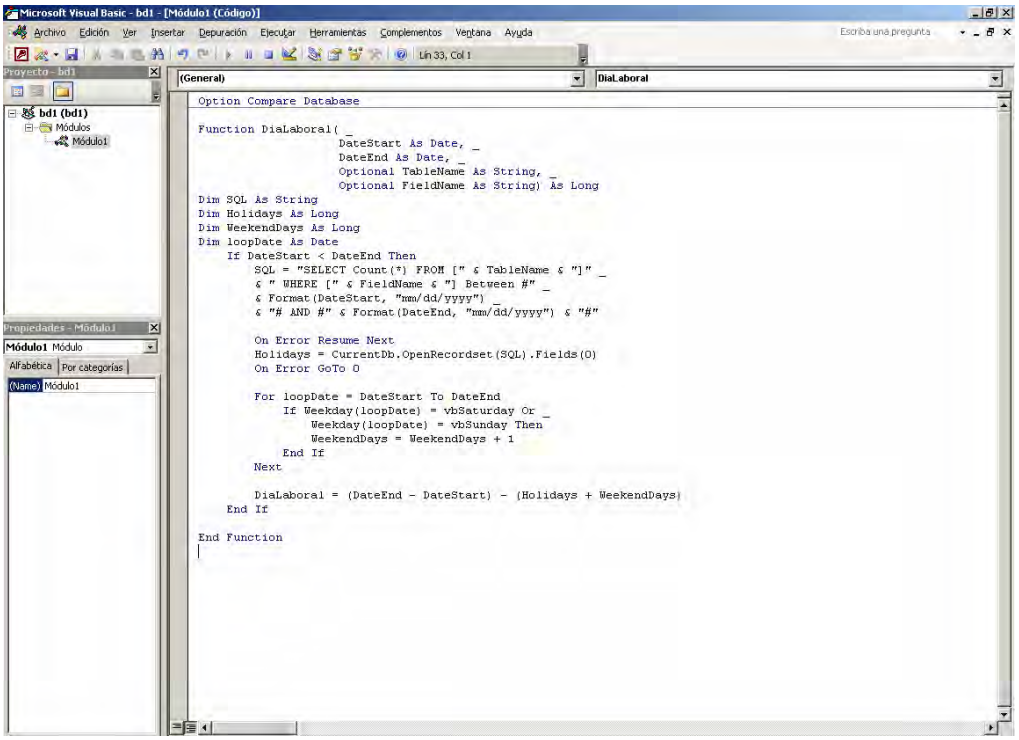
6.12. LOS MÓDULOS

Los módulos permiten ampliar las posibilidades de Access. Con ellos se pueden hacer procedimientos y funciones que no estén ya creados en Access para efectuar distintas operaciones en la base de datos. Los procedimientos y funciones se programan en lenguaje Visual Basic.

6.12.1. ¿Cómo se crea un módulo?

Los módulos se crean exactamente igual que los demás objetos de Access. Se selecciona la pestaña «Módulos» y se pulsa «Nuevo». Se abrirá el editor de Visual Basic para definir el módulo.

El siguiente ejemplo es una función para encontrar días labores entre dos fechas. Lo primero que necesitamos es crear una tabla la cual contenga los días festivos, teniendo en cuenta que no sean lo que se presentan en sábados y domingos, ya que el sistema restara los sábados y domingos como laborales. Nuestra tabla se llamara Festivos, y el campo en nuestra tabla se llama Días, este campo Días es en formato fecha. Creamos un modulo nuevo, definimos la función.



6.11.2. Guardar un módulo

Una vez se han terminado de definir la función, se guarda el módulo pulsando el botón «guardar» «  ». Aparecerá la siguiente ventana en la que se debe adjudicar un nombre a la macro creada:

Si no se pulsara el botón guardar y se cerrara la ventana de Visual Basic Access cerrará sin guardar.

6.11.3. Ejecutar un módulo

Algunos módulos están diseñados para ejecutarse automáticamente al abrir la base de datos, otros optimizan una consulta y por último, como es el caso del ejemplo, se ejecutan desde un formulario.

Para este ejemplo vamos a abrir el formulario donde se desea encontrar los días laborales, creamos un campo un escribimos:

=Módulo1([Fecha 1],[Fecha 2],"tblDiasFestivos","DiaFestivo")

PARTE TERCERA:

CONSULTAS A LA BASE DE DATOS MEDIANTE UN LENGUAJE DE CONSULTAS

Objetivo: Al finalizar el alumno tendrá los conocimientos básicos del manejo de un lenguaje de consultas de Bases de Datos.

CAPÍTULO 7: SQL

Aunque el lenguaje SQL se considere un lenguaje de consultas, contiene muchas otras capacidades además de la consulta en bases de datos. Incluye características para definir la estructura de los datos, para la modificación de los datos en la base de datos y para la especificación de restricciones de seguridad.

No se pretende proporcionar un manual de usuario completo para SQL. Por el contrario, se presentan las construcciones y conceptos fundamentales de SQL que se pueden utilizar en ACCESS como apoyo para la construcción de consultas. Las distintas implementaciones de SQL pueden diferenciarse en detalles, o pueden admitir sólo un subconjunto del lenguaje completo.

7.1. INTRODUCCIÓN

IBM desarrolló la versión original en su Laboratorio de Investigación de San José (San José Research Center, actualmente Centro de Investigación de Almadén, Almadén Research Center). IBM implementó el lenguaje, originalmente denominado Sequel, como parte del proyecto System R, a principios de 1970. El lenguaje Sequel ha evolucionado desde entonces y su nombre ha pasado a ser SQL (Structured Query Language, Lenguaje estructurado de consultas). Actualmente, numerosos productos son compatibles con el lenguaje SQL. SQL se ha establecido como el lenguaje estándar de bases de datos relacionales.

En 1986, ANSI (American National Standards Institute, Instituto Nacional Americano de Normalización) e ISO (International Standards Organization, Organización Internacional de Normalización), publicaron una norma SQL, denominada SQL-86. En 1987, IBM publicó su propia norma de SQL corporativo, Interfaz de bases de datos para arquitecturas de aplicación a sistemas (Systems Application Architecture Database Interface, SAA-SQL). En 1989 se publicó una norma extendida para SQL denominada SQL-89 y actualmente los sistemas de bases de datos son normalmente compatibles al menos con las características de SQL-89. La siguiente versión de la norma fue SQL-92 y la versión mas reciente es SQL:1999.

El lenguaje SQL tiene varios componentes:

- **Lenguaje de definición de datos** (LDD). El LDD de SQL proporciona órdenes para la definición de esquemas de relación, borrado de relaciones, creación de índices y modificación de esquemas de relación.

- **Lenguaje interactivo de manipulación de datos** (LMD). El LMD de SQL incluye un lenguaje de consultas, basado tanto en el álgebra relacional como en el cálculo relacional de tuplas. Incluye también órdenes para insertar, borrar y modificar tuplas de la base de datos.

- **Definición de vistas.** El LDD de SQL incluye órdenes para la definición de vistas.

- **Control de transacciones.** SQL incluye órdenes para la especificación del comienzo y final de transacciones.

- **SQL incorporado y SQL dinámico.** SQL dinámico e incorporado define cómo se pueden incorporar las instrucciones SQL en lenguajes de programación de propósito general, tales como C, C++, Java, PL/I, Cobol, Pascal y Fortran.

- **Integridad.** El LDD de SQL incluye órdenes para la especificación de las restricciones de integridad que deben satisfacer los datos almacenados en la base de datos. Las actualizaciones que violen las restricciones de integridad se rechazan.

- **Autorización.** El LDD de SQL incluye órdenes para especificar derechos de acceso para las relaciones y vistas.

En este capítulo se estudia el LMD Y las características básicas del LDD de SQL. También se describe brevemente SQL incorporado y dinámico, incluyendo las normas ODBC y JDBC para la interacción con una base de datos desde programas escritos en lenguajes C y Java.

Los ejemplos de este capítulo se basarán en una empresa bancaria, con los siguientes esquemas de relación:

Esquema-sucursal = (nombre-sucursal, ciudad-sucursal, activo)

Esquema-cliente = (nombre-cliente, calle-cliente, ciudad-cliente)

Esquema-préstamo = (número-préstamo, nombre-sucursal, importe)

Esquema-prestatario = (nombre-cliente, número-préstamo)

Esquema-cuenta = (número-cuenta, nombre-sucursal, saldo)

Esquema-impositor = (nombre-cliente, número-cuenta)

Nótese que se usan nombres separados por guiones para los esquemas, relaciones y atributos para facilitar su lectura. Sin embargo, en los sistemas SQL actuales, los guiones no son partes válidas de un nombre (se tratan como el operador menos). Una forma simple de traducir los nombres que se usan aquí a nombres SQL válidos es reemplazar todos los guiones por el símbolo de subrayado («_»). Por ejemplo, se usa nombre_sucursal en lugar de nombre-sucursal.

7.2. ESTRUCTURA BÁSICA

Una base de datos relacional consiste en un conjunto de relaciones, a cada una de las cuales se le asigna un nombre único. SQL permite el uso de valores nulos para indicar que el valor o bien es desconocido, o no existe. Se fijan criterios que permiten al usuario especificar a qué atributos no se puede asignar valor nulo.

La estructura básica de una expresión SQL consiste en tres cláusulas: **select, from y where.**

- La cláusula **select** corresponde a la operación proyección del álgebra relacional Se usa para listar los atributos deseados del resultado de una consulta.

- La cláusula **from** corresponde a la operación producto cartesiano del álgebra relacional. Lista las relaciones que deben ser analizadas en la evaluación de la expresión.

- La cláusula **where** corresponde al predicado selección del álgebra relaciona!. Es un predicado que engloba a los atributos de las relaciones que aparecen en la cláusula **from**.

Un hecho histórico desafortunado es que el término **select** tiene un significado diferente en SQL que en el álgebra relaciona!. A continuación se resaltan las diferentes interpretaciones, a fin de minimizar la posible confusión.

Una consulta típica en SQL tiene la forma

select $A_1, A_2, \cdots, A_n$
from $r_1, r_2, \ldots, r_m$
where P

Cada *Ai* representa un atributo, y cada *ri* una relación. *P* es un predicado. La consulta es equivalente a la expresión del álgebra relacional

$$\Pi A_1, A_2, \ldots, A_n(\sigma_P(r_1 x r_2 x \ldots x r_m))$$

Si se omite la cláusula **where**, el predicado P es cierto. Sin embargo, con diferencia a la expresión del álgebra relacional, el resultado de la consulta SQL puede contener varias copias de algunas tuplas. SQL forma el producto cartesiano de las relaciones incluidas en la cláusula **from**, lleva a cabo la selección del álgebra relacional usando el predicado de la cláusula **where** y entonces proyecta el resultado sobre los atributos de la cláusula **select**. En la práctica, SQL puede convertir la expresión en una forma equivalente que puede ser procesada más eficientemente.

7.2.1. La cláusula select

El resultado de una consulta SQL es, por supuesto, una relación. Considérese una consulta simple, usando el ejemplo bancario, «Obtener los números de todas las sucursales en la relación préstamo»:

select *nombre-sucursal*
from *préstamo*

El resultado es una relación consistente en el único atributo nombre-sucursal.

Los lenguajes formales de consulta están basados en la noción matemática de que una relación es un conjunto. Así, nunca aparecen tuplas duplicadas en las relaciones. En la práctica, la eliminación de duplicados consume tiempo. Sin embargo, SQL (como la mayoría de los lenguajes de consulta comerciales) permite duplicados en las relaciones, así como en el resultado de las expresiones SQL. Así, la consulta anterior

listará cada nombre-sucursal una vez por cada tupla en la que aparece en la relación préstamo.

En aquellos casos donde se quiera forzar la eliminación de duplicados, se insertará la palabra clave **distinct** después de **select**. Por lo tanto, se puede reescribir la consulta anterior como

select distinct *nombre-sucursal*
from *préstamo*

si se desean eliminar los duplicados.

Es importante resaltar que SQL permite usar la palabra clave **all** para especificar explícitamente que no se eliminan los duplicados:

select all *nombre-sucursal*
from *préstamo*

Como de manera predeterminada se realiza la retención de duplicados, de ahora en adelante no se usará la palabra clave **all** en los ejemplos. Para asegurar la eliminación de duplicados en el resultado de los ejemplos de consultas, se usará la cláusula **distinct** siempre que sea necesario. En la mayoría de las consultas donde no se utiliza **distinct**, el número exacto de copias duplicadas de cada tupla que resultan de la consulta no es importante. Sin embargo, el número es importante en ciertas aplicaciones.

El símbolo asterisco «*» se puede usar para denotar «todos los atributos». Así, el uso de *préstamo.* * en la cláusula **select** anterior indicaría que todos los atributos de préstamo serían seleccionados. Una cláusula **select** de la forma **select *** indica que se deben seleccionar todos los atributos de todas las relaciones que aparecen en la cláusula **from**.

La cláusula **select** puede contener también expresiones aritméticas que contengan los operadores, **+**, **-**, ***** y **/** operando sobre constantes o atributos de la tuplas. Por ejemplo, la consulta

select *nombre-sucursal, número-préstamo,*
  importe * 100
from *préstamo*

devolverá una relación que es igual que la relación préstamo, salvo que el atributo importe está multiplicado por 100.

SQL también proporciona tipos de datos especiales, tales como varias formas del tipo fecha y permite varias funciones aritméticas para operar sobre esos tipos.

7.2.2. La cláusula where

A continuación se ilustra con un ejemplo el uso de la cláusula **where** en SQL. Considérese la consulta «Obtener todos los números de préstamo para préstamos hechos en la sucursal con nombre Navacerrada, en los que el importe sea superior a 1.200 €». Esta consulta puede escribirse en SQL como

select *número-préstamo*
from *préstamo*
where *nombre-sucursal* = 'Navacerrada' **and** *importe> 1200*

SQL usa las conectivas lógicas **and, or y not** (en lugar de los símbolos matemáticos /\, v y -.) en la cláusula where. Los operandos de las conectivas lógicas pueden ser expresiones que contengan los operadores de comparación <, <=, >, >=, = y <>. SQL permite usar los operadores de comparación para comparar cadenas y expresiones aritméticas, así como tipos especiales, tales como el tipo fecha.

SQL incluye un operador de comparación **between** para simplificar las cláusulas where que especifica que un valor sea menor o igual que un valor y mayor o igual que otro valor. Si se desea obtener el número de préstamo de aquellos préstamos por importes entre 90.000 € y 100.000 €, se puede usar la comparación **between** para escribir

select *número-préstamo*
from *préstamo*
where *importe* **between** 90000 **and** 100000

en lugar de

select *número-préstamo*
from *préstamo*
where *importe* <= 100000 **and** *importe* >= 90000

De forma análoga, se puede usar el operador de comparación **not between.**

7.2.3. La cláusula from

Finalmente, se estudia el uso de la cláusula **from**. La cláusula from define por sí misma un producto cartesiano de las relaciones que aparecen en la cláusula. Escribir una expresión SQL para la reunión natural es una tarea relativamente fácil, puesto que la reunión natural se define en términos de un producto cartesiano, una selección y una proyección.

La expresión del álgebra relacional se escribe como sigue:

$$\Pi_{nombre-cliente,número-préstamo,importe}(prestario \otimes prestario)$$

para la consulta «Para todos los clientes que tienen un préstamo en el banco, obtener los nombres, números de préstamo e importes». Esta consulta puede escribirse en SQL como

select *nombre-cliente, prestatario. número-préstamo, importe*
from *prestatario, préstamo*
where *prestatario .número-préstamo = préstamo. número-préstamo*

Nótese que SQL usa la notación nombre-relación.nombre-atributo, como lo hace el álgebra relacional, para evitar ambigüedad en los casos en que un atributo aparece en el esquema de más de una relación. También se podría haber escrito prestatario.nombre-cliente en lugar de nombre-cliente, en la cláusula select. Sin

embargo, como el atributo nombre-cliente aparece sólo en una de las relaciones de la cláusula from, no existe ambigüedad al escribir nombre-cliente.

Se puede extender la consulta anterior y considerar un caso más complicado en el que se pide además qué clientes poseen un préstamo en la sucursal Navacerrada: «Obtener los nombres, números de préstamo e importes de todos los clientes que tienen un préstamo en la sucursal Navacerrada». Para escribir esta consulta será necesario establecer dos restricciones en la cláusula **where**, relacionadas con la conectiva lógica **and**:

select *nombre-cliente, prestatario.número-préstamo, importe*
from *prestatario, préstamo*
where *prestatario. número-préstamo = préstamo. número-préstamo* **and** *nombre-sucursal*= 'Navacerrada'

SQL-92 incluye extensiones para llevar a cabo reuniones naturales y reuniones externas en la cláusula **from**.

7.2.4. La operación renombramiento

SQL proporciona un mecanismo para renombrar tanto relaciones como atributos. Para ello utiliza la cláusula as, que tiene la forma siguiente.

nombre-antiguo ***as*** *nombre-nuevo*

la cláusula **as** puede aparecer tanto en select como en from.

Considérese de nuevo la consulta anterior:

select distinct *nombre-cliente, prestatario.número-préstamo, importe*
from *prestatario, préstamo*
where *prestatario. número-préstamo = préstamo.número-préstamo*

El resultado de esta consulta es una relación con los atributos siguientes:

nombre-cliente, número-préstamo, importe.

Los nombres de los atributos en el resultado se derivan de los nombres de los atributos de la relación que aparece en la cláusula **from**.

Sin embargo, no se pueden derivar siempre los nombres de este modo. En primer lugar, dos relaciones que aparecen en la cláusula **from** pueden tener atributos con el mismo nombre, en cuyo caso, un nombre de atributo se duplica en el resultado. En segundo lugar, si se incluye una expresión aritmética en la cláusula **select**, los atributos resultantes no tienen el mismo nombre. Y en tercer lugar, incluso si un nombre de atributo se puede derivar de las relaciones base, como en el ejemplo anterior, se puede querer cambiar el nombre del atributo en el resultado. Para todo ello, SQL proporciona una forma de renombrar los atributos de una relación resultado.

Por ejemplo, si se quisiera renombrar el atributo número-préstamo, asociándole el nombre de id-préstamo, se podría reescribir la consulta anterior del siguiente modo

select *nombre-cliente, prestatario.número-préstamo* **as** *id-préstamo, importe*

from *prestatario, préstamo*
where *prestatario. número-préstamo = préstamo .número-préstamo*

7.2.5. Variables tupla

La cláusula **as** es particularmente útil en la definición del concepto de variables tupla, como se hace en el cálculo relacional de tuplas. Una variable tupla en SQL se debe asociar con una relación concreta. Las variables tupla se definen en la cláusula **from** mediante el uso de la cláusula **as**. Como ejemplo, a continuación se reescribe la consulta «Obtener los nombres y números de préstamo de todos los clientes que tienen un préstamo en el banco» como sigue:

select *nombre-cliente, T.número-préstamo, S.importe*
from *prestatario* **as** *T, préstamo* **as** S
where *T.número-préstamo = S.número-préstamo*

Nótese que se define la variable tupla en la cláusula **from**, colocándola después del nombre de la relación a la cual está asociada y detrás de la palabra clave **as** (la palabra clave **as** es opcional). Al escribir expresiones de la forma nombre-relación.nombre-atributo, el nombre de la relación es, en efecto, una variable tupla definida implícitamente.

Las variables tupla son de gran utilidad para comparar dos tuplas de la misma relación. Hay que recordar que, en los casos de este tipo, se puede usar la operación renombramiento del álgebra relaciona!. Si se desea formular la consulta «Obtener los nombres de todas las sucursales que poseen un activo mayor que al menos una sucursal situada en Barcelona», se puede escribir la siguiente expresión SQL

select distinct *T.nombre-sucursal*
from *sucursal* **as** *T, sucursal* **as** S
where *T.activo* > *S.activo* **and** *S.ciudad-sucursal* = 'Barcelona'

Obsérvese que no se puede utilizar la notación sucursal.activo, puesto que no estaría claro a qué aparición ce sucursal se refiere.

SQL permite usar la notación $(v_1, v_2, ..., v_n)$ para designar una tupla de aridad *n* que contiene los valores $v_1, v_2, ..., v_n$. Los operadores de comparación se pueden utilizar sobre tuplas, y el orden se define lexicográficamente. Por ejemplo $(a_1, a_2) \leq (b_1, b_2)$ es cierto si $(a_1 < b_1)$ o si se cumple que $(a_1 = b_1) \wedge (a_2 \leq b_2)$ análogamente, dos tuplas son iguales si lo son todos sus atributos.

7.2.6. Operaciones sobre cadenas

SQL especifica las cadenas encerrándolas entre comillas simple, como 'Navacerrada', como se vio anteriormente. Un carácter comilla que sea parte de una cadena se puede especificar usando dos caracteres comilla; por ejemplo, la cadena «El carácter ' se puede ver en esta cadena» se puede especificar como 'El carácter '' se puede ver en esta cadena'.

La operación más usada sobre cadenas es el encaje de patrones, para el que se usa el operador **like**. Para la descripción de patrones se utilizan los dos caracteres especiales siguientes:

- Tanto porciento (%): El carácter % encaja con cualquier subcadena.
- Subrayado (_): El carácter _ encaja con cualquier subcadena.

Los patrones son muy sensibles, esto es, los caracteres mayúsculas no encajan con los caracteres en minúscula o viceversa. Para ilustrar el encaje de patrones, considérense los siguientes ejemplos:

- 'Nava%' encaja con cualquier cadena que empiece con «Nava»
- '%cer%' encaja con cualquier cadena que contenga «cer» como subcadena, por ejemplo 'Navacerrada', 'Cáceres' y 'Becerril'.
- '_ _ _' encaja con cualquier cadena de tres caracteres.
- '_ _ _%' encaja con cualquier cadena de al menos tres caracteres.

Los patrones se expresan en SQL utilizando el operador de comparación **like**. Considérese la consulta siguiente: «Obtener los nombres de todos los clientes cuyas calles contengan la subcadena 'Mayor'». Esta consulta se podría escribir como sigue

select *nombre-cliente*
from *cliente*
where *calle-cliente* **like** '%Mayor%'

Para que los patrones puedan contener los caracteres especiales patrón (esto es, % y _), SQL permite la especificación de un carácter de escape. El carácter de escape se utiliza inmediatamente antes de un carácter especial patrón para indicar que ese carácter especial va a ser tratado como un carácter normal. El carácter de escape para una comparación **like** se define utilizando la palabra clave **escape**. Para ilustrar esto, considérense los siguientes patrones, los cuales utilizan una barra invertida (\) como carácter de escape:

- **like** 'ab\%cd%' **escape** '\' encaja con todas las cadenas que empiecen por ab%cd .
- **like** 'ab\\cd%' **escape** '\' encaja con todas las cadenas que empiecen por ab\cd

SQL permite buscar discordancias en lugar de concordancias utilizando el operador de comparación **not like**.

SQL también proporciona una variedad de funciones que operan sobre cadenas de caracteres, tales como la concatenación (usando «ll»), la extracción de subcadenas, el cálculo de la longitud de las cadenas, la conversión a mayúsculas y minúsculas, etc. SQL:1999 también ofrece una operación similar **to** que proporciona un encaje de patrones más potente que la operación **like**; la sintaxis para especificar patrones es similar a la usada en Unix para expresiones regulares.

7.2.7. Orden en la presentación de las tuplas

SQL ofrece al usuario cierto control sobre el orden en el cual se presentan las tuplas de una relación. La cláusula **order by** hace que las tuplas resultantes de una consulta se presenten en un cierto orden. Para listar en orden alfabético todos los clientes que tienen un préstamo en la sucursal Navacerrada se escribirá:

select distinct *nombre_cliente*
from *prestatario, préstamo*
where *prestatario. número-préstamo*
= *préstamo.número-préstamo* **and**
nombre-sucursal = 'Navacerrada'
order by *nombre-cliente*

De manera predeterminada la cláusula **order by** lista los elementos en orden ascendente. Para especificar el tipo de ordenación se puede incluir la cláusula **dese** para orden descendente o **ase** para orden ascendente. Además, se puede ordenar con respecto a más de un atributo. Si se desea listar la relación préstamo en orden descendente para importe. Si varios préstamos tienen el mismo importe, se ordenan ascendentemente según el número de préstamo. Esta consulta en SQL se escribe del modo siguiente:

select *
from *préstamo*
order by *importe* **dese**, *número-préstamo* **ase**

Para ejecutar una consulta que contiene la cláusula **order by**, SQL tiene que llevar a cabo una ordenación. Como ordenar un gran número de tuplas puede ser costoso, es conveniente ordenar sólo cuando sea estrictamente necesario.

7.2.8. Duplicados

La utilización de relaciones con duplicados se ha mostrado útil en diversas situaciones. SQL no sólo define formalmente las tuplas que están en el resultado de una consulta, sino también el número de copias de cada una de esas tuplas que aparece en el resultado. La semántica de duplicados de una consulta SQL se puede definir utilizando versiones de los operadores relacionales para multiconjuntos. A continuación se definen las versiones multiconjunto de varios de los operadores del álgebra relacional. Dadas las relaciones multiconjunto r_1 y r_2,

1. Si existen c_1 copias de la tupla t_1 en r_1 y t_1 satisface la selección σ_θ, entonces hay c_1 copias de t_1 en $\sigma_\theta(r_1)$.

2. Para cada copia de la tupla t_1 en r_1, hay una copia de la tupla $\Pi_A(t_1)$ en $\Pi_A(r_1)$, donde $\Pi_A(t_1)$ denota la proyección de la tupla única t_1.

3. Si existen c_1 copias de la tupla t_1 en r_1 y c_2 copias de la tupla t_2 en r_2 entonces hay c_1 * c_2 copias de la tupla $t_1.t_2$ en r_1 x r_2.

Por ejemplo, supóngase que las relaciones r_1 con esquema (A, B) y r_2 con esquema (C) son los multiconjuntos siguientes:

$$r_1 = \{(1, a), (2, a)\} \qquad r_2 = \{(2), (3), (3)\}$$

Entonces, $\Pi_B(r_1)$ sería $\{(a), (a)\}$, mientras que $\Pi_B(r_1)\, x\, r_2$ sería

$$\{(a,2), (a,2), (a,3), (a,3), (a,3), (a,3)\}$$

Se puede ahora definir cuántas copias de cada tupla aparecen en el resultado de una consulta SQL. Una consulta SQL de la forma

select $A_1, A_2, ..., A_n$
from $r_1, r_2, ..., r_n$
where P

es equivalente a la expresión del álgebra relacional

$$\Pi_{A_1, A_2, ..., A_n}(\sigma_P(r_1 x r_2 x ... x r_m))$$

usando las versiones multiconjunto de los operadores relacionales σ, Π y x.

7.3. OPERACIONES SOBRE CONJUNTOS

Las operaciones de SQL-92 **union**, **intersect** y **except** operan sobre relaciones y corresponden a las operaciones del álgebra relacional $\cup, \cap$ y -. Al igual que la unión, intersección y diferencia de conjuntos en el álgebra relacional, las relaciones que participan en las operaciones han de ser compatibles; esto es, deben tener el mismo conjunto de atributos.

A continuación se demuestra cómo se pueden formular en SQL varias de las consultas de ejemplo del banco utilizando consultas que incluyen las operaciones union, intersect y except de dos conjuntos. Los dos conjuntos utilizados serán: el conjuntos de todos los clientes que tienen una cuenta en el banco, que puede obtenerse con:

select *nombre-cliente*
from *impositor*

y el conjunto de todos los clientes que tienen un préstamo en el banco, que puede obtenerse con:

select *nombre-cliente*
from *prestario*

A partir de ahora, las letras *i* y *p* se utilizarán para hacer referencia a las relaciones obtenidas como resultado de las dos consultas anteriores.

7.3.1. La operación unión

Para encontrar todos los clientes que poseen un préstamo, una cuenta o las dos cosas en el banco, se escribirá:

(**select** *nombre-cliente*
from *impositor)*
union
(**select** *nombre-cliente*
from *prestario)*

A diferencia de la cláusula select, la operación union (unión) elimina duplicados automáticamente. Así, en la consulta anterior, si un cliente -por ejemplo, Santos- tiene varias cuentas o préstamos (o ambas cosas) en el banco, entonces Santos aparecerá sólo una vez en el resultado.

Para conservar los duplicados, se utilizará **union all** en lugar de **union**:

(**select** *nombre-cliente*
from *impositor)*
union all
(**select** *nombre-cliente*
from *prestario)*

El número de tuplas duplicadas en el resultado es igual al número total de duplicados que aparecen en *i* y *p.* Así, si Santos tuviese tres cuentas y dos préstamos en el banco, entonces en el resultado aparecerían cinco tuplas con el nombre de Santos.

7.3.2. La operación intersección

Para encontrar todos los clientes que tienen tanto un préstamo como una cuenta en el banco, se escribirá:

(**select distinct** *nombre-cliente*
from *impositor)*
intersect
(**select distinct** *nombre-cliente*
from *prestario)*

La operacion **intersect** (intersección) elimina duplicados automáticamente. Así, en la consulta anterior, si un cliente -por ejemplo, Santos- tiene varias cuentas o préstamos (o ambas cosas) en el banco, entonces Santos aparecerá solo una vez en el resultado.

Para conservar los duplicados se utilizará **intersect all** en lugar de **intersect**:

(**select distinct** *nombre-cliente*
from *impositor)*
intersect all
(**select distinct** *nombre-cliente*

from *prestario)*

El número de tuplas duplicadas en el resultado es igual al mínimo número de duplicados que aparecen en *i* y *p.* Así, si Santos tuviese tres cuentas y dos préstamos en el banco, entonces en el resultado de la consulta aparecerían dos tuplas con el nombre de Santos

7.3.3. La operación except

Para encontrar todos los clientes que tienen cuenta pero no tienen ningún préstamo en el banco se escribirá:

(**select distinct** *nombre-cliente*
from *impositor)*
except
(**select distinct** *nombre-cliente*
from *prestario)*

La operacion **except** (excepto) elimina duplicados automáticamente. Así, en la consulta anterior, una tupla con el nombre de Santos aparecerá en el resultado (exactamente una vez), sólo si Santos tiene una cuenta en el banco, pero no tiene ningún préstamo en el mismo.

Para conservar los duplicados, se utilizará **except all** en lugar de **except**:

(**select distinct** *nombre-cliente*
from *impositor)*
except all
(**select distinct** *nombre-cliente*
from *prestario)*

El número de copias duplicadas de una tupla en el resultado es igual al número de copias duplicadas de dicha tupla en *i* menos el número de copias duplicadas de la misma tupla en *p,* siempre que la diferencia sea positiva. Así, si Santos tuviese tres cuentas y un préstamo en el banco, entonces en el resultado aparecerían dos tuplas con el nombre de Santos. Si, por el contrario, dicho cliente tuviese dos cuentas y tres préstamos en el banco, no habrá ninguna tupla con el nombre de Santos en el resultado.

7.4. FUNCIONES DE AGREGACIÓN

Las funciones de agregación son funciones que toman una colección (un conjunto o multiconjunto) de valores como entrada y producen un único valor como salida. SQL proporciona cinco funciones de agregación primitivas:

- Media: **avg**
- Mínimo: **min**
- Máximo: **max**

- Total: **sum**
- Cuenta: **count**

La entrada a **sum** y **avg** debe ser una colección de números, pero los otros operadores pueden operar sobre colecciones de datos de tipo no numérico, tales como las cadenas.

Como ejemplo, considérese la consulta «Obtener la media de saldos de las cuentas de la sucursal Navacerrada». Esta consulta se puede formular del modo siguiente:

select avg *(saldo)*
from *cuenta*
where *nombre-sucursal* = 'Navacerrada'

El resultado de esta consulta será una relación con un único atributo, que contendrá una única fila con un valor numérico correspondiente al saldo medio de la sucursal Navacerrada. Opcionalmente se puede dar un nombre al atributo resultado de la relación, usando la cláusula **as**.

Existen situaciones en las cuales sería deseable aplicar las funciones de agregación no sólo a un único conjunto de tuplas sino también a un grupo de conjuntos de tuplas; esto se especifica en SQL usando la cláusula **group by**. El atributo o atributos especificados en la cláusula group by se usan para formar grupos. Las tuplas con el mismo valor en todos los atributos especificados en la cláusula group by se colocan en un grupo.

Como ejemplo, considérese la consulta «Obtener el saldo medio de las cuentas de cada sucursal».

Dicha consulta se formulará del modo siguiente

select *nombre-sucursal,* **avg** *(saldo)*
from *cuenta*
group by *nombre-sucursal*

La conservación de duplicados es importante al calcular una media. Supóngase que los saldos de las cuentas en la (pequeña) sucursal de nombre «Galapagar» son 1.000 €, 3.000 €, 2.000 € y 1.000 €. El saldo medio es 7.000/4 = 1.750 €. Si se eliminasen duplicados se obtendría un resultado erróneo (6.000/3 = 2.000 €).

Hay casos en los que se deben eliminar los duplicados antes de calcular una función de agregación. Para eliminar duplicados se utiliza la palabra clave **distinct** en la expresión de agregación. Como ejemplo considérese la consulta «Obtener el número de impositores de cada sucursal». En este caso un impositor sólo se debe contar una vez, sin tener en cuenta el número de cuentas que el impositor pueda tener. La consulta se formulará del modo siguiente:

select *nombre-sucursal,* **count** (**distinct** *nombre-cliente)*
from *impositor, cuenta*
where *impositor.número-cuenta = cuenta.número-cuenta*
group by *nombre-sucursal*

A veces es más útil establecer una condición que se aplique a los grupos que una que se aplique a las tuplas. Por ejemplo, podemos estar interesados sólo en aquellas sucursales donde el saldo medio de cuentas es superior a 1.200 €. Esta condición no es aplicable a una única tupla; se aplica a cada grupo construido por la cláusula **group by**. Para expresar este tipo de consultas se utiliza la cláusula **having** de SQL. Los predicados de la cláusula having se aplican después de la formación de grupos, de modo que se pueden usar las funciones de agregación. Esta consulta se expresa en SQL del modo siguiente:

select *nombre-sucursal,* **avg** *(saldo)*
from *cuenta*
group by *nombre-sucursal* **having avg** (saldo) > 1200

A veces se desea tratar la relación entera como un único grupo. En casos de este tipo no se usa la cláusula **group by**. Considérese la consulta «Obtener el saldo medio de todas las cuentas». Esta consulta se formulará del modo siguiente:

select avg *(saldo)*
from *cuenta*

Con mucha frecuencia se usa la función de agregación **count** para contar el número de tuplas de una relación. La notación para esta función en SQL es **count (*)**. Así, para encontrar el número de tuplas de la relación cliente, se escribirá

select count (*)
from *cliente*

SQL no permite el uso de **distinct** con **count (*)**. Sí se permite, sin embargo, el uso de **distinct** con **max** y **min**, incluso cuando el resultado no cambia. Se puede usar la palabra clave **all** en lugar de **distinct** para especificar la retención de duplicados, pero como all se especifica de manera predeterminada, no es necesario incluir dicha cláusula.

Si en una misma consulta aparece una cláusula **where** y una cláusula **having**, se aplica primero el predicado de la cláusula **where**. Las tuplas que satisfagan el predicado de la cláusula **where** se colocan en grupos según la cláusula **group by**. La cláusula **having**, si existe, se aplica entonces a cada grupo; los grupos que no satisfagan el predicado de la cláusula **having** se eliminan. La cláusula **select** utiliza los grupos restantes para generar las tuplas resultado de la consulta.

Para ilustrar el uso de la cláusula **where** y la cláusula **having** dentro de la misma consulta considérese el ejemplo «Obtener el saldo medio de cada cliente que vive en Madrid y tiene como mínimo tres cuentas».

select *impositor. nombre-cliente* , **avg** *(saldo)*
from *impositor, cuenta, cliente*
where *impositor. número-cuenta*
= *cuenta.número-cuenta* **and**
impositor.nombre-c liente
= *cliente.nombre-cliente* **and**
ciudad-cliente = 'Madrid'
group by *impositor. nombre-cliente*
having count (**distinct** *impositor. número-cuenta*) >= 3

7.5. VALORES NULOS

SQL permite el uso de valores nulos para indicar la ausencia de información sobre el valor de un atributo.

En un predicado se puede usar la palabra clave especial **null** para comprobar si un valor es nulo. Así, para encontrar todos los números de préstamo que aparecen en la relación préstamo con valores nulos para importe se escribe

select *número-préstamo*
from *préstamo*
where *importe* **is null**

El predicado **is not null** pregunta por la ausencia de un valor nulo.

El uso de un valor nulo en las operaciones aritméticas y de comparación causa varias complicaciones.

El resultado de una expresión aritmética (incluyendo por ejemplo +,-,* o /) es nulo si cualquiera de los valores de entrada es nulo. SQL trata como desconocido el resultado de cualquier comparación que implique un valor nulo (aparte de **is null** e **is not null**).

Dado que el predicado en una cláusula **where** puede incluir operaciones booleanas tales como **and**, **or** y **not** sobre los resultados de las comparaciones, las definiciones de estas operaciones se extienden para manejar el valor desconocido.

- **and**: el resultado de cierto **and** desconocido es desconocido falso **and** desconocido es falso, mientras que desconocido **and** desconocido es desconocido.

- **or**: el resultado de cierto **or** desconocido es cierto, falso **or** desconocido es desconocido, mientras que desconocido **or** desconocido es desconocido.

SQL define el resultado de una instrucción SQL de la forma

select… from $R_1,...,R_n$ **where** P

para contener (proyecciones de) tuplas en $R_1 x...xR_n$ para las que el predicado P se evalúa a cierto. Si el predicado se evalúa a falso o desconocido para una tupla de $R_1 x...xR_n$ (la proyección de) la tupla no se añade al resultado.

SQL también permite determinar si el resultado de una comparación es desconocido en lugar de cierto o falso usando las cláusulas **is unknown** (es desconocido) e **is not unknown** (no es desconocido)

La existencia de valores nulos también complica el procesamiento de los operadores de agregación. Supóngase que algunas tuplas en la relación préstamo tienen valor nulo para el atributo importe. Considérese en ese caso la siguiente consulta, que calcula el total de todas las cantidades prestadas:

select sum *(importe)*
from *préstamo*

Los valores que van a ser sumados en la consulta anterior incluyen valores nulos, puesto que algunas tuplas tienen valor nulo para el atributo importe. En lugar de decir que la suma total es nula, la norma SQL establece que el operador **sum** debería ignorar los valores nulos de su entrada.

En general, las funciones de agregación tratan los valores nulos según la regla siguiente: todas las funciones de agregación excepto **count(*)** ignoran los valores nulos de la colección de datos de entrada. Como resultado de ignorar los valores nulos, la colección de valores de entrada puede resultar vacía. El cálculo de **count** de una colección vacía se define como 0 y todas las demás operaciones de agregación devuelven un valor nulo cuando se aplican sobre una colección de datos vacía. El efecto de los valores nulos en algunas de las construcciones más complicadas de SQL puede ser más sutil.

En SQL: 1999 se introdujo un tipo de datos boleano, que puede tomar los valores **cierto**, **falso** y **desconocido**. Las funciones de agregación **some** (algún) y **every** (cada), que significan exactamente lo que se espera de ellas, se pueden aplicar a una colección de valores booleanos.

7.6. SUBCONSULTAS ANIDADAS

SQL proporciona un mecanismo para las subconsultas anidadas. Una subconsulta es una expresión **select-from-where** que se anida dentro de otra consulta. Un uso común de subconsultas es llevar a cabo comprobaciones sobre pertenencia a conjuntos, comparación de conjuntos y cardinalidad de conjuntos. Estos usos se estudiarán en los apartados siguientes.

7.6.1. Pertenencia a conjuntos

SQL utiliza el cálculo relacional para las operaciones que permiten comprobar la pertenencia de una tupla a una relación. La conectiva in comprueba la pertenencia a un conjunto, donde el conjunto es la colección de valores resultado de una cláusula **select**. La conectiva **not in** comprueba la no pertenencia a un conjunto. Como ejemplo considérese de nuevo la consulta «Encontrar todos los clientes que tienen tanto un préstamo como una cuenta en el banco». Anteriormente escribimos esta consulta como la intersección de dos conjuntos: el conjunto de los impositores del banco y el conjunto de los prestatarios del banco. Sin embargo, existe un enfoque alternativo consistente en encontrar todos los tenedores de cuentas en el banco que son miembros del conjunto de prestatarios. Claramente, esta formulación genera el mismo resultado que la anterior, pero obliga a formular la consulta usando la conectiva **in** de SQL. A continuación, se van a obtener todos los tenedores de cuentas formulando así la siguiente subconsulta:

(**select** *nombre-cliente*
from *impositor)*

A continuación es necesario encontrar aquellos clientes que son prestatarios del banco y que aparecen en la lista de tenedores de cuenta, obtenida como resultado de la subconsulta anterior. Esto se consigue anidando la subconsulta en un **select** más externo. La consulta resultante es la siguiente:

```
select distinct nombre-cliente
from prestatario
where nombre-cliente in (select nombre-cliente
from impositor)
```

Este ejemplo muestra que es posible escribir la misma consulta de diversas formas en SQL. Esta flexibilidad es de gran utilidad, puesto que permite al usuario pensar en una consulta del modo que le parezca más natural. Más adelante se verá que existe una gran cantidad de redundancia en SQL.

En el ejemplo anterior se comprobaba la pertenencia a un conjunto en una relación de un solo atributo. También es posible comprobar la pertenencia a un conjunto en una relación cualquiera. Así, se puede formular la consulta «Listar los clientes que tienen tanto una cuenta como un préstamo en la sucursal Navacerrada» de un modo distinto al visto anteriormente:

```
select distinct nombre-cliente
from prestatario, préstamo
where prestatario. número-préstamo =
        préstamo .número-préstamo and
        nombre-sucursal = 'Navacerrada' and
        (nombre-sucursal, nombre-cliente) in
                (select nombre-sucursal, nombre-cliente
                from impositor, cuenta
                where impositor. número-cuenta
                        = cuenta. número-cuenta)
```

A continuación, se ilustra el uso de la constructora **not in**. Por ejemplo, para encontrar todos los clientes que tienen un préstamo en el banco, pero no tienen una cuenta en el banco, se puede escribir

```
select distinct nombre-cliente
from prestatario
where nombre-cliente not in (select nombre-cliente from impositor)
```

Los operadores **in** y **not in** también se pueden usar sobre conjuntos enumerados. La consulta siguiente selecciona los nombres de los clientes que tienen un préstamo en el banco y cuyos nombres no son ni «Santos» ni «Gómez».

```
select distinct nombre-cliente
from prestatario
where nombre-cliente not in ('Santos', 'Gómez')
```

7.6.2. Comparación de conjuntos

Considérese la consulta «Obtener los nombres de todas las sucursales que poseen un activo mayor que al menos una sucursal situada en Barcelona». En el apartado variables tupla se formulaba esta consulta del modo siguiente:

```
select distinct T.nombre-sucursal
from sucursal as T, sucursal as S
where T.activo > S.activo and S.ciudad-sucursal = 'Barcelona'
```

SQL ofrece, sin embargo, un estilo alternativo de formular la consulta anterior. La expresión: «mayor que al menos una» se representa en SQL por **> some**. Esta constructora permite reescribir la consulta en una forma más parecida a la formulación de la consulta en lenguaje natural.

```
select nombre-sucursal
from sucursal
where activo> some (select activo
                    from sucursal
                    where ciudad-sucursal = 'Barcelona')
```

La subconsulta

```
(select activo
from sucursal
where ciudad-sucursal = 'Barcelona')
```

genera el conjunto de todos los valores de activo para todas las sucursales sitas en Barcelona. La comparación **> some**, en la cláusula **where** de la cláusula **select** más externa, es cierta si el valor del atributo activo de la tupla es mayor que al menos un miembro del conjunto de todos los valores de activo de las sucursales de Barcelona.

SQL también permite realizar las comparaciones **< some, <= some, >= some, = some y <> some.** Como ejercicio, se puede verificar que **= some** es idéntico a **in**, mientras que **<= some** no es lo mismo que **not in**. En SQL, la palabra clave **any** es sinónimo de **some**. Las versiones más antiguas de SQL sólo admitían **any**. Sin embargo, versiones posteriores añadieron la alternativa **some** para evitar la ambigiledad lingilística de la palabra inglesa **any**.

Ahora la consulta se modificará ligeramente a fin de obtener los nombres de todas las sucursales que tienen un activo superior al de todas las sucursales de Barcelona. La constructora **> all** corresponde a la expresión «superior a todas». Utilizando esta constructora la consulta se podría formular del modo siguiente:

```
select nombre-sucursal
from sucursal
where activo > all (select activo
                    from sucursal
                    where ciudad-sucursal = 'Barcelona')
```

Al igual que con **some**, SQL también permite utilizar las comparaciones **< all**, **<= all**, **>= all**, **= all** y **<> all**. Como ejercicio se puede verificar que **<= any** es lo mismo que **not in**.

Como otro ejemplo de comparaciones considérese la consulta «Encontrar la sucursal que tiene el mayor saldo medio». En SQL, las funciones de agregación no se pueden componer. Así, no está pemiitido el uso de **max (avg (...)).** Por ello, para la formulación de esta consulta se seguirá la estrategia siguiente: para empezar se formula una consulta para encontrar todos los saldos medios, y luego se anida ésta

como subconsulta de una consulta más larga que encuentre aquellas sucursales para las que el saldo medio es mayor o igual que todos los saldos medios:

```
select nombre-sucursal
from cuenta
group by nombre-sucursal
having avg (saldo) >= all (select avg (saldo)
                           from cuenta
                           group by nombre-sucursal)
```

7.6.3. Comprobación de relaciones vacías

SQL incluye la posibilidad de comprobar si una subconsulta no produce ninguna tupla como resultado. La constructora **exists** devuelve el valor cierto si la subconsulta argumento no es vacía.

Usando la constructora **exists** se puede formular la consulta «Obtener los clientes que tienen tanto una cuenta como un préstamo en el banco» de otra nueva forma:

```
select nombre-cliente
where exists (select *
              from impositor
              where impositor. nombre-cliente =
              prestatario .nombre-cliente)
```

Utilizando la constructora **not exists** se puede comprobar la inexistencia de tuplas en el resultado de una subconsulta. Además, es posible usar la constructora **not exists** para simular la operación de continencia de conjuntos (es decir, superconjunto). Así, se puede escribir la expresión «la relación *A* contiene a la relación *B»* como «**not exists** (B **except** A) ». Aunque no forma parte de SQL estándar, el operador **contains** aparece en algunos sistemas relacionales. Para ilustrar el operador **not exists** considérese otra vez la consulta «Obtener todos los clientes que tienen una cuenta en todas las sucursales de Barcelona». Será necesario comprobar para cada cliente si el conjunto de todas las sucursales en las que dicho cliente tiene cuenta contiene al conjunto de todas las sucursales de Barcelona. Utilizando el operador except se puede formular la consulta del modo siguiente:

```
select distinct S.nombre-cliente
from impositor as S
where not exists ((select nombre-sucursal
                   from sucursal
                   where ciudad-sucursal = 'Barcelona')
                  except
                     (select R.nombre-sucursal
                     from impositor as T, cuenta as R
                     where T.mímero-cuenta
                     = R.número-cuenta and S.nombre-cliente
                     = T.nombre-cliente ))
```

En este ejemplo, la subconsulta

```
(select nombre-sucursal
from sucursal
```

where *ciudad-sucursal* = 'Barcelona')

obtiene todas las sucursales de Barcelona. Por otro lado, la subconsulta

(**select** *R.nombre-sucursal*
from *impositor* as *T, cuenta* as *R*
where *T.número-cuenta = R.número-cuenta* **and**
S.nombre-cliente = T.nombre-cliente)

obtiene todas las sucursales en las cuales el cliente *S.nombre-cliente* tiene una cuenta. Por último, el **select** más externo toma cada cliente y comprueba si el conjunto de todas las sucursales en las que dicho cliente tiene cuenta, contiene al conjunto de todas las sucursales de Barcelona.

En consultas que contengan subconsultas se aplica una regla de visibilidad para las variables tupla. En una subconsulta, sólo se pueden usar variables tupla que estén definidas en la propia subconsulta o en cualquier consulta que contenga a dicha subconsulta. Si una variable tupla está definida tanto localmente (en una subconsulta) como globalmente (en una consulta que contenga a la subconsulta) se aplica la definición local. Esta regla es análoga a la utilizada para las variables en los lenguajes de programación.

7.6.4. Comprobación de tuplas duplicadas

SQL incluye la posibilidad de comprobar si una subconsulta produce como resultado tuplas duplicadas. La constructora **unique** devuelve el valor cierto si la subconsulta que se le pasa como argumento no produce tuplas duplicadas. Usando la constructora **unique** se puede formular la consulta «Obtener todos los clientes que tienen sólo una cuenta en la sucursal de nombre Navacerrada» del siguiente modo:

select *T.nombre-cliente*
from *impositor* **as** *T*
where unique (**select** *R.nombre-cliente*
from *cuenta, impositor* **as** *R*
where *T.nombre-cliente*
= *R.nombre-cliente* **and** *R.número-cuenta*
= *cuenta. número-cuenta* **and** .
cuenta. nombre-sucursal = 'Navacerrada')

La existencia de tuplas duplicadas en una subconsulta se puede comprobar utilizando la constructora **not unique**. Para ilustrar esta constructora considérese la consulta «Obtener todos los clientes que tienen al menos dos cuentas en la sucursal Navacerrada», que se puede formular del modo siguiente:

select distinct *T.nombre-cliente*
from *impositor* **as** *T*
where not unique (**select** *R.nombre-cliente*
from *cuenta, impositor* as *R*
where *T.nombre-cliente*
= *R.nombre-cliente* **and**
R.número-cuenta =
cuenta.número-cuenta **and**

cuenta.número-sucursal
= 'Navacerrada')

Formalmente, la comprobación hecha por la constructora unique sobre una relación debería fallar si y sólo si en la relación existieran dos tuplas t_1 y t_2 tales que $t_1 = t_2$. Como la comprobación $t_1 = t_2$ sólo falla si cualquier campo de t_1 o de t_2 es nulo, entonces es posible que el resultado de unique sea cierto incluso si existen varias copias de una tupla, siempre que al menos uno de los atributos de la tupla sea nulo.

7.7. VISTAS

Una vista en SQL se define utilizando la orden **create view**. Para definir una vista se le debe dar un nombre y se debe construir la consulta que genere dicha vista. La forma de la orden **create view** es la siguiente:

create view v **as** <expresión de consulta>

donde <expresión de consulta> puede ser cualquier consulta válida. El nombre de la vista se representa por v. Nótese que la notación usada para la definición de una vista en el álgebra relacional se basa en esta de SQL.

Como ejemplo considérese la vista consistente en los nombres de sucursales y los nombres de los clientes que tienen una cuenta o un préstamo en esa sucursal. Si se denomina esta vista como todos-los-clientes se definirá del modo siguiente:

```
create view todos-los-clientes as
    (select nombre-sucursal, nombre-cliente
    from impositor, cuenta
    where impositor.número-cuenta
          = cuenta.número-cuenta)
  union
    (select nombre-sucursal, nombre-cliente
    from prestatario, préstamo
    where prestatario.número-préstamo
    = préstamo .número-préstamo)
```

Los nombres de los atributos de una vista se pueden indicar explícitamente de la forma siguiente:

```
create view total-préstamos-sucursal
          (nombre-sucursal, total-préstamos) as
select nombre-sucursal, sum (importe)
from préstamo
group by nombre-sucursal
```

La vista anterior contiene para cada sucursal la suma de los importes de todos los préstamos de esa sucursal. Como la expresión **sum** (importe) no tiene nombre, el nombre del atributo se especifica explícitamente en la definición de la vista.

Los nombres de vistas pueden aparecer en cualquier lugar en el que pudiera aparecer un nombre de relación. Usando la vista *todos-los-clientes,* se pueden listar todos los clientes de la sucursal Navacerrada, escribiendo

```
select nombre-cliente
from todos-los-clientes
where nombre-sucursal = 'Navacerrada'
```

7.8. CONSULTAS COMPLEJAS

Las consultas complejas son a menudo difíciles o imposibles de escribir como un único bloque SQL o una unión, intersección o diferencia de bloques SQL (un bloque SQL consiste en una única instrucción **select from where**, posiblemente con cláusulas **group by** y **having**). Aquí se estudian dos formas de componer varios bloques SQL para expresar una consulta compleja: las relaciones derivadas y la cláusula **with**.

7.8.1. Relaciones derivadas

SQL permite el uso de una expresión de subconsulta en la cláusula **from**. Si se usa una expresión de este tipo se debe dar un nombre a la relación resultado y se pueden renombrar los atributos usando la cláusula **as**. Por ejemplo, considérese la subconsulta

```
(select nombre-sucursal, avg (saldo)
  from cuenta
  group by nombre-sucursal)
as media-sucursal (nombre-sucursal, saldo-medio)
```

Esta subconsulta produce una relación consistente en los nombres de todas las sucursales y sus correspondientes saldos de cuenta medios. El resultado de la subconsulta recibe el nombre de media-sucursal y contiene los atributos nombre-sucursal y saldo-medio.

Para ilustrar el uso de una expresión de subconsulta en la cláusula **from** considérese la consulta «Obtener el saldo medio de las cuentas de aquellas sucursales donde dicho saldo medio sea superior a 1.200 €». En el apartado funciones de agregación se formulaba esta consulta utilizando la cláusula **having**. Ahora se puede reescribir dicha consulta sin usar esta cláusula de la siguiente forma:

```
select nombre-sucursal, saldo-medio
from (select nombre-sucursal, avg (saldo)
        from cuenta
        group by nombre-sucursal)
      as resultado (nombre-sucursal, saldo-medio)
where saldo-medio> 1200
```

En esta formulación no es necesario el uso de la cláusula **having** puesto que la relación temporal resultado se calcula en la cláusula **from**, y los atributos de resultado se pueden usar directamente en la cláusula **where**.

Supóngase como otro ejemplo que se desea hallar el máximo del total de saldos de todas las sucursales. La cláusula **having** no sirve en este caso, pero se puede escribir fácilmente esta consulta usando una subconsulta en la cláusula **from**, como se muestra a continuación:

```
select max(saldo-total)
from (select nombre-sucursal, sum(saldo)
        from cuenta
        group by nombre-sucursal) as
                    total-sucursal (nombre-sucursal,
                    saldo-total)
```

7.8.2. La cláusula with

Las consultas complicadas son mucho más fáciles de formular y de entender si se descomponen en vistas más simples y después se combinan, al igual que se estructuran los programas, descomponiendo sus tareas en procedimientos. Sin embargo, son distintas a la definición de procedimientos en cuanto a que una cláusula **create view** crea una definición de vista en la base de datos y esa definición de vista permanece en la base de datos hasta que se ejecuta una orden **drop view** nombre-vista.

La cláusula **with** proporciona una forma de definir una vista temporal cuya definición está disponible sólo para la consulta en la que aparece esta cláusula. Considérese la siguiente consulta, que selecciona cuentas con el saldo máximo; si hay muchas cuentas con el mismo saldo máximo, todas ellas se seleccionan.

```
with saldo-máximo(valor) as
     select max (saldo)
     from cuenta
select número-cuenta
from cuenta, saldo-máximo
where cuenta.saldo = saldo-máximo. valor
```

La cláusula **with** introducida en SQL:1999 se incluye actualmente sólo en algunas bases de datos.

Se podría haber escrito la consulta anterior usando una subconsulta anidada tanto en la cláusula **from** como en la **where**. Sin embargo, el uso de subconsultas anidadas hace que la consulta sea más difícil de leer y entender. La cláusula **with** hace que la lógica de la consulta sea más clara; también permite usar una definición de vista en varios lugares de una consulta.

Por ejemplo, supóngase que se desea encontrar todas las sucursales donde el depósito de cuentas es mayor que la media del total de depósitos de cuentas en todas las sucursales. Se puede escribir la consulta con la cláusula **with** como se muestra a continuación.

```
with total-sucursal(nombre-sucursal,valor) as
       select nombre-sucursal, sum(saldo)
       from cuenta
       group by nombre-sucursal
with total-media-sucursal(valor) as
```

select avg(sa *valor ldo)*
from *total-sucursal*
select *nombre-sucursal*
from *total-sucursal, total-media-sucursal*
where *total-sucursal. valor> = total-media-sucursal. valor*

Por supuesto, se puede crear una consulta equivalente sin la cláusula **with,** pero sería más complicada y difícil de entender. Como ejercicio, se puede escribir la consulta equivalente.

7.9. MODIFICACIÓN DE LA BASE DE DATOS

Hasta ahora nos hemos limitado a la extracción de información de una base de datos. A continuación se mostrará cómo añadir, eliminar o cambiar información utilizando SQL.

7.9.1. Borrado

Un borrado se expresa de igual modo que una consulta. Se pueden borrar sólo tuplas completas, es decir, no se pueden borrar valores de atributos concretos. Un borrado se expresa en SQL del modo siguiente:

delete from *r*
where *P*

donde P representa un predicado y r representa una relación. La declaración **delete** selecciona primero todas las tuplas t en r para las que $P(t)$ es cierto y a continuación las borra de r. La cláusula **where** se puede omitir, en cuyo caso se borran todas las tuplas de r.

Hay que señalar que una orden **delete** opera sólo sobre una relación. Si se desea borrar tuplas de varias relaciones, se deberá utilizar una orden **delete** por cada relación. El predicado de la cláusula **where** puede ser tan complicado como el **where** de cualquier cláusula **select,** o tan simple como una cláusula **where** vacía. La consulta

delete from *préstamo*

borra todas las tuplas de la relación préstamo (los sitemas bien diseñados requerirán una confirmación del usuario antes de ejecutar una consulta tan devastadora).

A continuación se muestran una serie de ejemplos de borrados en SQL.

- Borrar todas las cuentas de la sucursal Navacerrada.

delete from *cuenta*
where *nombre-sucursal* = 'Navacerrada'

- Borrar todos los préstamos en los que la cantidad esté comprendida entre 1.300 € y 1.500 €.

```
delete from préstamo
where importe between 1300 and 1500
```

- Borrar las cuentas de todas las sucursales de Navacerrada.

```
delete from cuenta
where nombre-sucursal in (select nombre-sucursal
from sucursal
where ciudad-sucursal = 'Navacerrada')
```

El borrado anterior selecciona primero todas las sucursales con sede en Navacerrada y a continuación borra todas las tuplas cuenta pertenecientes a esas sucursales.

Nótese que, si bien sólo se pueden borrar tuplas de una sola relación cada vez, se puede utilizar cualquier número de relaciones en una expresión **select-from-where** anidada en lá cláusula **where** de un **delete.** La orden **delete** puede contener un **select** anidado que use una relación de la cual se van a borrar tuplas. Por ejemplo, para borrar todas las cuentas cuyos saldos sean inferiores a la media del banco se puede escribir:

```
delete from cuenta
where saldo < (select avg (saldo)
                from cuenta)
```

La orden **delete** comprueba primero cada tupla de la relación cuenta para comprobar si la cuenta tiene un saldo inferior a la media del banco. A continuación se borran todas las tuplas que no cumplan la condición anterior, es decir, las que representan una cuenta con un saldo menor que la media. Es importante realizar todas las comprobaciones antes de llevar a cabo ningún borrado (si se borrasen algunas tuplas antes de que otras fueran comprobadas, el saldo medio podría haber cambiado y el resultado final del borrado dependería del orden en que las tuplas fueran procesadas).

7.9.2. Inserción

Para insertar datos en una relación, o bien se especifica la tupla que se desea insertar o se formula una consulta cuyo resultado sea el conjunto de tuplas que se desean insertar. Obviamente, los valores de los atributos de la tuplas que se inserten deben pertenecer al dominio de los atributos. De igual modo, las tuplas insertadas deberán ser de la aridad correcta.

La instrucción **insert** más sencilla corresponde a la de inserción de una tupla. Supongamos que se desea insertar en la base de datos el hecho de que hay una cuenta C-9732 en la sucursal Navacerrada y que dicha cuenta tiene un saldo de 1.200 €. La inserción se puede formular del modo siguiente:

```
insert into cuenta
        values ('C-9732', 'Navacerrada', 1200)
```

En este ejemplo los valores se especifican en el mismo orden en que los atributos se listan en el esquema de relación. Para beneficio de los usuarios, que pueden no recordar el orden de los atributos, SQL permite que los atributos se especifiquen en la cláusula **insert.** Así, el siguiente ejemplo tiene una función idéntica al anterior:

```
insert into cuenta (nombre-sucursal, número-cuenta, saldo)
    values ('Navacerrada', 'C-9732', 1200)

insert into cuenta (número-cuenta, nombre-sucursal, saldo)
    values ('C-9732', 'Navacerrada', 1200)
```

Generalmente se desea insertar las tuplas que resultan de una consulta. Por ejemplo, si a todos los clientes tenedores de préstamos en la sucursal Navacerrada se les quisiera regalar, como gratificación, una cuenta de ahorro con 200 € por cada cuenta de préstamo que tienen, se podría escribir:

```
insert into cuenta
    select nombre-sucursal, número-préstamo, 200
    from préstamo
    where nombre-sucurs.al = 'Navacerrada'
```

En lugar de especificar una tupla, como se hizo en los primeros ejemplos de este apartado, se utiliza una instrucción **select** para especificar un conjunto de tuplas. La instrucción **select** se evalúa primero, produciendo un conjunto de tuplas que a continuación se insertan en la relación cuenta. Cada tupla tiene un nombre-sucursal (Navacerrada), un número-préstamo (que sirve como número de cuenta para la nueva cuenta) y un saldo inicial de la cuenta (200 €).

Además es necesario añadir tuplas a la relación impositor; para hacer esto, se escribirá:

```
insert into impositor
    select nombre-cliente, número-préstamo
    from prestatario, préstamo
    where prestatario.número-préstamo
        = préstamo. número-préstamo and
        nombre-sucursal = 'Navacerrada'
```

Esta consulta inserta en la relación impositor una tupla (nombre-cliente, número-préstamo) por cada nombre-cliente que posea un préstamo en la sucursal Navacerrada, con número de préstamo número-préstamo.

Es importante que la evaluación de la instrucción select finalice completamente antes de llevar a cabo ninguna inserción. Si se realizase alguna inserción antes de que finalizase la evaluación de la instrucción select, una consulta del tipo:

```
insert into cuenta
    select *
    from cuenta
```

podría insertar un número infinito de tuplas. La primera tupla de la relación cuenta se insertaría de nuevo en cuenta, creando así una segunda copia de la tupla. Como esta segunda copia ya sería parte de cuenta, la instrucción select podría seleccionarla, insertando así una tercera copia en la relación cuenta. Esta tercera copia podría ser seleccionada a continuación por el select e insertar una cuarta copia y así

infinitamente. Evaluando completamente toda la instrucción select antes de realizar ninguna inserción se evitan este tipo de problemas.

Por ahora, en el estudio de la instrucción **insert** sólo' se han considerado ejemplos en los que se especificaba un valor para cada atributo de las tuplas insertadas. Es posible indicar sólo valores para algunos de los atributos del esquema. A cada uno de los atributos restantes, se les asignará un valor nulo, que se denota por **null** Como ejemplo considérese la consulta:

insert into *cuenta*
values ('C-401', *null, 1200)*

en la que se sabe que la cuenta C-401 tiene un saldo de 1.200 €, pero no se conoce el nombre de la sucursal. Considérese ahora la consulta

select *número-cuenta*
from *cuenta*
where *nombre-sucursal* = 'Navacerrada'

Como el nombre de la sucursal de la cuenta C-40 1 es desconocido, no se puede determinar si es igual a «Navacerrada».

Se puede prohibir la inserción de valores nulos utilizando el LDD de SQL

7.9.3. Actualizaciones

En determinadas situaciones puede ser deseable cambiar un valor dentro de una tupla, sin cambiar todos los valores de la misma. Para este tipo de situaciones se utiliza la instrucción **update.** Al igual que ocurre con **insert** y **delete,** se pueden elegir las tuplas que van a ser actualizadas mediante una consulta.

Por ejemplo, si hubiera que realizar el pago de intereses anuales y todos los saldos se incrementasen en un 5 %, habría que formular la siguiente actualización:

update *cuenta*
set *saldo* = *saldo* * 1.05

Esta actualización se aplica una vez a cada tupla de la relación cuenta.

Si se paga el interés sólo a las cuentas con un saldo de 1.000 € o superior, se puede escribir

update *cuenta*
set *saldo* = *saldo* * 1.05
where *saldo* >= 1000

En general, la cláusula **where** de la instrucción **update** puede contener cualquier constructor legar en la cláusula **where** de una instrucción **select** incluyendo instrucciones **select** anidadas). Como con **insert** y **delete**, un select anidado en una instrucción update puede referenciar la relación que se esté actualizando. Como antes, SQL primero comprueba todas las tuplas de la relación para determinar las que se deberían actualizar y después realiza la actualización. Por ejemplo, se puede escribir «Pagar un interés del 5% a las cuentas cuyo saldo sea mayor que la media» como sigue:

```
update cuenta
set saldo = saldo * 1.05
where (saldo> select avg(saldo)
              from cuenta)
```

Si se supone que las cuentas con saldos superiores a 10.000 € reciben un 6% de interés, mientras que las demás un 5%, se deberán escribir dos instrucciones de actualización:

```
update cuenta
set saldo = saldo * 1.06
where saldo> 10000

update cuenta
set saldo = saldo * 1.05
where saldo <= 10000
```

Obsérvese que el orden en el que se ejecutan dos instrucciones de actualización es importante. Si se invierte el orden de las dos instrucciones anteriores, una cuenta con un saldo igualo muy poco inferior a 10.000 € recibiría un 11,3% de interés.

SQL ofrece una constructora **case**, que se puede usar para formular las dos instrucciones de actualización anteriores en una única instrucción de actualización, evitando el problema del orden de actualización.

```
update cuenta
set saldo = case
when saldo <= 10000 then saldo * 1.05
                    else saldo * 1.06
            end
```

La forma general de la instrucción case es la siguiente:

```
case
        when pred1 then result1
        when pred2 then result2
        …
        when predn then resultn
        else result0
end
```

La operación devuelve $result_1$, donde i es el primero de $result_1$, $result_2$, …, $result_n$ que se satisface; si ninguno de ellos se satisface, la operación devuelve $result_0$. Las instrucciones case se pueden usar en cualquier lugar donde se espere un valor.

7.9.4. Actualización de vistas

Aunque las vistas son una herramienta útil para las consultas, plantean problemas significativos si con ellas se expresan las actualizaciones, las inserciones o los

borrados. La dificultad radica en que las modificaciones de la base de datos expresadas en términos de vistas deben traducirse en modificaciones de las relaciones reales en el modelo lógico de la base de datos. Como ejemplo considérese la siguiente definición de vista:

create view *préstamo-sucursal* **as**
select *nombre-sucursal, número-préstamo*
from *préstamo*

Como SQL permite que el nombre de una vista aparezca en cualquier lugar en el que pueda aparecer el nombre de una relación, se puede formular:

insert into *préstamo-sucursal*
values ('Navacerrada', 'P-307')

SQL representa esta inserción mediante una inserción en la relación préstamo, puesto que préstamo es la relación real a partir de la cual se construye la vista préstamo-sucursal. Por lo tanto, debería especificarse un valor para el atributo importe. Este valor es un valor nulo. De este modo, la inserción anterior es equivalente a la inserción de la tupla

('P-307', 'Navacerrada', *null)*

en la relación préstamo.

La anomalía de la actualización de vistas se agrava cuando una vista se define en términos de varias relaciones. Como resultado, muchas bases de datos basadas en SQL imponen la siguiente restricción a las modificaciones de vistas:

- Una modificación de una vista es válida sólo si la vista en cuestión se define en términos de la base de datos relacional real, esto es, del nivel lógico de la base de datos (y sin usar agregación).

Bajo esta restricción, las operaciones **update, insert** y **delete** realizadas sobre el ejemplo de la vista todos-los-clientes definida anteriormente, estarían prohibidas.

7.9.5. Transacciones

Una transacción consiste en una secuencia de instrucciones de consulta y actualizaciones. La norma SQL especifica que una transacción comienza implícitamente cuando se ejecuta una instrucción SQL. Una de las siguientes instrucciones SQL debe finalizar la transacción:

- **Commit work** compromete la transacción actual; es decir, hace que los cambios realizados por la transacción sean permanentes en la base de datos. Después de que se comprometa la transacción se inicia una nueva transacción automáticamente.

- **Rollback work** causa el retroceso de la transacción actual; es decir, deshace todas las actualizaciones realizadas por las instrucciones SQL de la transacción; así, el estado de la base de datos se restaura al que existía previo a la ejecución de la transacción.

La palabra **work** es opcional en ambas instrucciones.

El retroceso de transacciones es útil si se detecta alguna condición de error durante la ejecución de una transacción. El compromiso es similar, bajo un punto de vista, a guardar los cambios de un documento que se esté modificando, mientras que el retroceso es similar a abandonar la sesión de modificación sin guardar los cambios. Una vez que una transacción haya ejecutado **commit work,** sus efectos no se pueden deshacer con **rollback work.** El sistema de bases de datos garantiza que en el caso de una caída, los efectos de la transacción se retrocederán si no se hubo ejecutado **commit work.** En el caso de fallo de alimentación o caída del sistema, el retroceso ocurre cuando el sistema se reinicia.

Por ejemplo, para transferir dinero de una cuenta a otra es necesario actualizar dos saldos de cuenta. Las dos instrucciones de actualización formarían una transacción. Un error durante la ejecución de una de las instrucciones de una transacción resultaría en deshacer los efectos de las instrucciones anteriores de la transacción, de manera que la base de datos no se quedase en un estado parcialmente actualizado.

Si un programa termina sin ejecutar ninguna de estas órdenes, las actualizaciones se comprometen o retroceden. La norma no especifica cuál de los dos se debe realizar, con lo que la elección es dependiente de la implementación. En muchas implementaciones SQL, cada instrucción SQL es de manera predeterminada una transacción y se compromete tan pronto como se ejecuta. El compromiso automático de las instrucciones SQL individuales se puede des activar si es necesario ejecutar una transacción que conste de varias instrucciones SQL. La forma de des activación del compromiso automático depende de la implementación SQL específica.

Una alternativa mejor, que es parte de la norma SQL: 1999 (pero actualmente sólo soportada por algunas implementaciones SQL), es permitir encerrar varias instrucciones SQL entre las palabras clave **begin atomic ... end**. Todas las instrucciones entre las palabras clave forman así una única transacción.

7.10. REUNIÓN DE RELACIONES

Además de proporcionar el mecanismo básico de producto cartesiano para reunir tuplas de relaciones, permitido en versiones anteriores, SQL también proporciona varios mecanismos para reunir relaciones, incluyendo reuniones condicionales y reuniones naturales, así como varias formas de reunión externa. Estas operaciones adicionales se usan a menudo como subconsultas dentro de la cláusula **from.**

7.10.1. Ejemplos

En la figura siguiente se muestran las relaciones préstamo y prestatario que usarán las distintas operaciones de reunión que ilustraremos.

número-préstamo	nombre-sucursal	importe
P-170	Centro	3.000
P-230	Moralzarzal	4.000
P-260	Navacerrada	1.700

préstamo

nombre-cliente	número-préstamo
Santos	P-170
Gómez	P-230
López	P-155

prestario

Figura 7.1 *Las relaciones préstamo y prestario*

Comenzaremos con un ejemplo simple de reunión interna. En la figura siguiente se muestra el resultado de la expresión:

préstamo **inner join** *prestatario* **on**
préstamo .número-préstamo
= prestatario. número-préstamo

La expresión calcula la reunión zeta de las relaciones préstamo y prestatario, donde la condición de reunión es préstamo.número-préstamo = prestatario.número-préstamo. Los atributos del resultado son los atributos del lado izquierdo de la relación, seguidos por los del lado derecho de la misma.

número-préstamo	nombre-sucursal	importe	nombre-cliente	número-préstamo
P-170	Centro	3.000	Santos	P-170
P-230	Moralzarzal	4.000	Gómez	P-230

Figura 7.2 *Resultado de préstamo* ***inner join*** *prestatario* ***on*** *préstamo .número-préstamo = prestatario. número-préstamo*

Obsérvese que el atributo número-préstamo aparece dos veces en la figura (la primera aparición es debida a la relación préstamo y la segunda a prestatario). La norma SQL no requiere que los nombres de atributo en los resultados sean únicos. Se debería usar una cláusula as para asignar nombres únicos de atributos en los resultados de las consultas y subconsultas.

Renombramos la relación resultado de una reunión y los atributos de la relación resultado utilizando la cláusula **as**, como se ilustra a continuación:

préstamo **inner join** *prestatario* **on**
préstamo.número-préstamo
= prestatario .número-préstamo
as *ps (sucursal, número-préstamo, importe,*
cliente, número-préstamo-cliente)

La segunda aparición de número-préstamo se ha renombrado como número-préstamo-cliente. El orden de los atributos en el resultado de una reunión natural es importante a la hora de renombrados.

A continuación se muestra un ejemplo del uso de la operación reunión externa por la izquierda (**left outer join**):

préstamo **left outer join** *prestatario* **on**
préstamo.número-préstamo
= prestatario.número-préstamo

La reunión externa por la izquierda se calcula del modo siguiente. Primero se calcula el resultado de la reunión interna, como se vio anteriormente. A continuación, para cada tupla t de la unión interna, perteneciente a la relación del lado izquierdo (préstamo) que no encaje con ninguna tupla de la relación del lado derecho (prestatario), se añade al resultado de la reunión una tupla r, como se indica a continuación. Los atributos de la tupla r que se derivan de la relación del lado izquierdo se rellenan con los valores de la tupla t y el resto de los atributos de r se rellenan con valores nulos. La relación resultante se muestra en la figura 7.3.

número-préstamo	nombre-sucursal	importe	nombre-cliente	número-préstamo
P-170	Centro	3.000	Santos	P-170
P-230	Moralzarzal	4.000	Gómez	P-230
P-260	Navacerrada	1.700	null	null

Figura 7.3 *Resultado de préstamo* ***left outer join*** *prestatario* ***on*** *préstamo.número-préstamo = prestatario.número-préstamo*

Las tuplas (P-170, Centro, 3.000) y (P-230, Moralzarzal, 4.000) se reúnen con las tuplas de prestatario y aparecen en el resultado de la reunión interna, y por ello en el resultado de la reunión externa por la izquierda. Por otra parte, la tupla (P-260, Navacerrada, 1.700) no encaja con ninguna tupla de prestatario en la reunión natural y por eso en el resultado de la reunión externa por la izquierda aparece la tupla (P-260, Navacerrada, 1.700. null, null).

Finalmente, se incluye un ejemplo del uso de la operación reunión natural (**natural join**).

préstamo **natural inner join** *prestatario*

Esta expresión calcula la reunión natural de dos relaciones. El único nombre de atributo común en préstamo y prestatario es número-préstamo. El resultado de la expresión anterior se muestra en la figura siguiente.

número-préstamo	nombre-sucursal	importe	nombre-cliente
P-170	Centro	3.000	Santos
P-230	Moralzarzal	4.000	Gómez

Figura 7.4 *Resultado de préstamo* ***natural inner join*** *prestatario*

Este resultado es similar al resultado de la reunión interna con la condición **on** mostrada en la figura 7.2, puesto que tienen la misma condición de reunión. Sin embargo, el atributo número-préstamo aparece sólo una vez en el resultado de la reunión natural, mientras que aparece dos veces en el resultado de la reunión con la condición **on**.

7.10.2. Tipos y condiciones de reunión

En el apartado anterior se muestran ejemplos de los operadores de reunión incluidos en SQL-92. Las operaciones de reunión toman como entrada dos relaciones y devuelven como resultado otra relación. Aunque las expresiones de reunión externa

se usan normalmente en la cláusula **from**, se pueden utilizar en cualquier lugar en el que cabría usar cualquier otra relación.

Cada variante de las operaciones de reunión en SQL-92 está formado por un tipo de reunión y una condición de reunión. La condición de reunión indica las tuplas pertenecientes a las dos relaciones que encajan y los atributos que se incluyen en el resultado de la reunión. El tipo de reunión define cómo se tratan las tuplas de cada relación que no encajan con ninguna tupla de la otra relación (basado en la condición de reunión). En la figura 7.5 se muestran algunos de los tipos y condiciones de reunión. El primer tipo de reunión es la reunión interna y los otros tres son tres tipos de reuniones externas. Las tres condiciones de reunión son: la reunión **natural** y la condición **on**, ya vistas anteriormente, y la condición **using**, que se verá más adelante.

Tipos de reunión
inner join
left outer join
rigth outer join
full outer join

Condiciones de reunión
natural
on <predicado>
using $(A_1, A_2, ..., A_n)$

Figura 7.5 *Tipos y condiciones de runión*

El uso de una condición de reunión es obligatorio en las reuniones externas, pero es opcional en las reuniones internas (ya que si se omite, el resultado será el producto cartesiano de las dos relaciones). La palabra clave **natural** aparece sintácticamente delante del tipo de reunión, como se mostró anteriormente, mientras que las condiciones **on** y **using** aparecen al final de la expresión de reunión. Las palabras clave **inner** y **outer** son opcionales, ya que el resto del tipo de reunión permite deducir si la reunión es una reunión interna o externa.

El significado de la condición de reunión **natural**, en términos de las tuplas de las relaciones que encajan, es inmediato. La ordenación de los atributos, dentro del resultado de la reunión natural es el siguiente: los atributos de reunión (es decir, los atributos comunes a las dos relaciones) aparecen en primer lugar, en el orden en el que aparezcan en la relación del lado izquierdo. A continuación están los demás atributos que no son de reunión de la relación del lado izquierdo y, al final, todos los atributos que no son de reunión de la relación del lado derecho de la relación.

El tipo de relación reunión externa por la derecha (**right outer join**) es simétrico al de reunión externa por la izquierda (**left outer join**). Las tuplas de la relación del lado derecho que no encajen con ninguna tupla de la relación del lado izquierdo se rellenan con valores nulos y se añaden al resultado de la reunión externa por la derecha.

La siguiente expresión es un ejemplo de la combinación de la condición de reunión natural con el tipo de reunión externa por la derecha.

préstamo **natural right outer join** *prestatario*

El resultado de la expresión anterior se muestra en la figura 7.6. Los atributos del resultado se definen por el tipo de reunión, que es una reunión natural; así, número-préstamo aparece sólo una vez. Las primeras dos tuplas del resultado provienen de la reunión natural de préstamo y prestatario. La tupla de la relación del lado derecho (López, P-155) no encaja en la reunión interna con ninguna tupla de la relación del

lado izquierdo (préstamo). Así, la tupla (P-155, null, López) aparece en el resultado de la reunión.

número-préstamo	nombre-sucursal	importe	nombre-cliente
P-170	Centro	3.000	Santos
P-230	Moralzarzal	4.000	Gómez
P-260	null	null	López

Figura 7.6 *Resultado de préstamo* ***natural right outer join*** *prestatario*

La condición de reunión **using** $(A_1, A_2, ..., A_n)$ es similar a la condición de reunión **natural**, salvo en que los atributos de reunión en este caso son $A_1, A_2, ..., A_n$ en lugar de todos los atributos comunes de ambas relaciones y aparecen sólo una vez en el resultado de la unión.

El tipo de reunión externa completa (**full outer join**) es una combinación de los tipos de reunión externa por la derecha y por la izquierda. Después de calcular el resultado de la reunión interna, las tuplas de la relación del lado izquierdo que no encajen con ninguna tupla de la relación del lado derecho se completan con valores nulos y se añaden al resultado. De forma análoga, las tuplas de la relación del lado derecho que no encajen con ninguna tupla de la relación del lado izquierdo, se completan con valores nulos y se añaden al resultado.

Por ejemplo, la siguiente figura muestra el resultado de la expresión

préstamo **full outer join** *prestatario* **using**
(número-préstamo)

número-préstamo	nombre-sucursal	importe	nombre-cliente
P-170	Centro	3.000	Santos
P-230	Moralzarzal	4.000	Gómez
P-260	null	null	null
P-155	Null	null	López

Figura 7.7 *Resultado de préstamo* ***full outer join*** *prestatario* ***using*** *(número-préstamo)*

Otro ejemplo del uso de la operación reunión externa es la consulta: «Listar todos los clientes que poseen una cuenta pero no tienen un préstamo en el banco». Esta consulta se puede formular como sigue:

select *i-NC*
from *(impositor* **left outer join** *prestatario*
on *impositor. nombre-cliente*
= prestatario.nombre-cliente)
as *db* 1 *(i-NC, número-cuenta, p-NC,*
número-préstamo)
where *p-NC* **is** *null*

De forma análoga, la consulta «Listar todos los clientes que tienen o bien una cuenta o un préstamo en el banco (pero no ambos)» podría formularse usando el operador de reunión natural externa completa, del modo siguiente:

select *nombre-cliente*
from *(impositor* **natural full outer join** *prestatario)*
where *número-cuenta* **is** *null* **or** *número-préstamo*
is *null*

SQL-92 proporciona además otros dos tipos de reunión, llamados **cross join** (reunión cruzada) y **union join** (reunión de unión). El primero es equivalente a una reunión interna sin condición de reunión; el segundo es equivalente a una reunión externa completa con condición falsa, es decir, donde la reunión interna es vacía.

7.11. LENGUAJE DE DEFINICIÓN DE DATOS

En muchos de nuestros análisis sobre SQL y bases de datos relacionales hemos aceptado un conjunto de relaciones como predefinidas. Por supuesto, el conjunto de relaciones en una base de datos se debe especificar en términos de un lenguaje de definición de datos (LDD).

El LDD de SQL permite la especificación no sólo de un conjunto de relaciones, sino también de alguna información relativa a esas relaciones, incluyendo:

- El esquema de cada relación
- El dominio de valores asociado a cada atributo
- Las restricciones de integridad
- El conjunto de índices que se deben mantener por cada relación
- Información de seguridad y autorización para cada relación
- La estructura de almacenamiento físico de cada relación en disco.

Se analizará a continuación la definición de esquemas y de dominios de valores.

7.11.1. Tipos de dominios en SQL

La norma SQL soporta un conjunto de tipos de dominios predefinidos, que incluye los siguientes:

- **char** (n) es una cadena de caracteres de longitud fija, con una longitud n especificada por el usuario. También se puede utilizar la palabra completa character.
- **varchar** (n) es una cadena de caracteres de longitud variable, con una longitud n especificada por el usuario. También se puede utilizar la forma completa **character varying**.
- **int** es un entero (un subconjunto finito de los enteros, que es dependiente de la máquina). También se puede usar la palabra completa **integer**.

- **smallint** es un entero pequeño (un subconjunto del dominio de los enteros, también dependiente de la máquina).

- **numeric** *(p,d)* es un número en coma flotante, cuya precisión la especifica el usuario. El número está formado por *p* dígitos (más el signo), y de esos *p* dígitos, *d* pertenecen a la parte decimal. Así, **numeric** (3,1) permite que el número 44,5 se almacene exactamente, mientras que los números 444,5 y 0,32 no se pueden almacenar exactamente en un campo de este tipo.

- **real**, **double precision** son respectivamente números en coma flotante y números en coma flotante de doble precisión, con precisión dependiente de la máquina.

- **float** (n) es un número en coma flotante, cuya precisión es de al menos n dígitos.

- **date** es una fecha del calendario, que contiene un año (de cuatro dígitos), un mes y un día del mes.
- **time** es la hora del día, expresada en horas, minutos y segundos. Se puede usar una variante, **time**(p), para especificar el número de dígitos decimales para los segundos (el número predeterminado es 0). También es posible almacenar la infomación del uso horario junto al tiempo.

- **timestamp** es una combinación de **date** y **time**. Se puede usar una variante, **timestamp**(p), para especificar el número de dígitos decimales para los segundos (el número predeterminado es 6).

Los valores de fecha y hora se pueden especificar como:

date *'2001-04-25'*
time *'09:30:00'*
timestamp *'2001-04-25 10:29:01.45'*

Las fechas se deben especificar en el formato año seguido de mes y de día, como se muestra. El campo segundos de time y **timestamp** puede tener parte decimal, como se ha mostrado. Se puede usar una expresión de la forma **cast** e **as** t para convertir una cadena de caracteres (o una expresión de tipo cadena) e al tipo t, donde t es **date**, **time** o **timestamp**. La cadena debe estar en el formato adecuado como se indicó al comienzo de este párrafo.

Para extraer campos individuales de un valor d **date** o **time** se puede usar **extract**(campo **from** d), donde campo puede ser **year**, **month**, **day**, **hour**, **minute** o **segundo**.

Las cadenas de caracteres de longitud variable, la fecha y la hora no forman parte de la norma SQL.

SQL permite realizar operaciones de comparación sobre todos los dominios que se listan aquí, y permite realizar operaciones aritméticas y de comparación sobre los diferentes dominios numéricos. SQL también proporciona un tipo de datos llamado **interval** y permite realizar cálculos basados en fechas, horas e intervalos. Por

ejemplo, si x e y son del tipo date, entonces $x - y$ será un intervalo cuyo valor es el número de días desde la fecha x hasta la y. De forma análoga, al sumar o restar un intervalo de una fecha u hora, se obtendrá como resultado otra fecha u hora, respectivamente.

A menudo es útil poder comparar valores de dominios compatibles. Por ejemplo, como cada entero perteneciente al tipo **smallint** es un entero, una comparación $x < y$, donde x es de tipo **smallint** e y es de tipo **int** (o viceversa), es válida. Este tipo de comparación se lleva a cabo transformando primero el número x en un entero. Una transformación de este tipo se denomina coerción. La coerción de tipos se usa normalmente en los lenguajes de programación comunes, así como en los sistemas de bases de datos.

Como ejemplo, supóngase que el dominio de nombre-cliente es una cadena de caracteres de longitud 20 y que el dominio de nombre-sucursal es una cadena de caracteres de longitud 15. Aunque las longitudes de las cadenas de caracteres difieran, la norma SQL los considerará tipos compatibles.

El valor **null** pertenece a todos los dominios. Para ciertos atributos, sin embargo, los valores nulos pueden no ser apropiados. Considérese una tupla de la relación cliente donde el nombre-cliente es nulo. Una tupla de este tipo asociaría una calle y una ciudad a clientes anónimos: es decir, no contendrán información útil. En casos de este tipo, sería deseable prohibir el uso de valores nulos, restringiendo el dominio de nombre-cliente para excluir los valores nulos.

SQL permite incluir en la declaración de dominio de un atributo la especificación **not null** y de este modo se prohíbe la inserción de un valor nulo para ese atributo. Cualquier modificación de la base de datos que conduzca a la inserción de un valor nulo en un dominio especificado como **not null**, generará un diagnóstico de error. Existen muchas situaciones en las que sería deseable la prohibición de valores nulos. Un caso particular donde es esencial prohibir valores nulos es en la clave primaria de un esquema de relación. De este modo, en el ejemplo bancario, en la relación cliente se prohibirá el uso de valores nulos para el atributo nombre-cliente, que es la clave primaria de la relación.

7.11.2. Definición de esquemas en SQL

Un esquema de relación se define utilizando la orden **create table**:

create table $r(A_1D_1, A_2D_2, ..., A_nD_n,$
$< restricción - \text{int} egridad_1 >$
...
$< restricción - \text{int} egridad_k >)$

donde r es el nombre de la relación, cada A_i es el nombre de un atributo del esquema de relación r y D_i es el dominio de los valores del atributo A_i. Las restricciones de integridad válidas incluyen:

- **primary key** $(A_{j1}, A_{j2}, ..., A_{jm})$ *:* la especificación de clave primaria dice que los atributos $A_{j1}, A_{j2}, ..., A_{jm}$, forman la clave primaria de la relación. Los atributos clave primaria deben ser no nulos y únicos; es decir, ninguna tupla puede tener un valor nulo para un atributo de la clave primaria y ningún par de tuplas de la relación pueden ser iguales en todos los atributos clave primaria]. Aunque la especificación de clave primaria es opcional, es generalmente buena idea especificar una clave primaria para cada relación.

- **check** *(P):* la cláusula **check** especifica un predicado *P* que debe satisfacer cada tupla de la relación.

La orden de creación de tabla **create table** también incluye otras restricciones de integridad.

En la Figura 7.8 se representa una definición parcial en SQL de la base de datos bancaria. Obsérvese que, al igual que en capítulos anteriores, no se intenta modelar con precisión el mundo real en el ejemplo de la base de datos bancaria. En el mundo real, muchas personas tienen el mismo nombre por lo que nombre-cliente no sería una clave primaria de cliente; probablemente se usaría un id-cliente como clave primaria. Se usa nombre-cliente como clave primaria para mantener el esquema de la base de datos simple y pequeño.

```
create table cliente
        (nombre-cliente char (20),
        calle-cliente   char (30),
        ciudad-cliente          char (30),
        primary key (nombre-cliente))

create table sucursal (nombre-sucursal char (15),
        ciudad-sucursal char (30),
        activo          integer,
        primary key (nombre-sucursal),
        check (activo >= 0))

create table cuenta (número-cuenta char (10),
        nombre-sucursal char (15),
        saldo integer,
        primary key (número-cuenta),
        check (saldo >= 0))

create table impositor (nombre-cliente char (20),
        número-cuenta char (10),
        primary key (nombre-cliente, número-cuenta))
```

FIGURA 7.8*. Definición de datos en SQL para parte de la base de datos del banco.*

Si como resultado de una inserción o modificación, una tupla toma valores nulos para cualquiera de los atributos que forman parte de la clave primaria, o si tiene el mismo valor que otra tupla de la relación para éstos, SQL notifica el error y la actualización no se lleva a cabo. De forma análoga ocurre lo mismo si falla la condición **check** de una tupla.

De manera predeterminada, **null** es un valor válido para cualquier atributo en SQL, a menos que se especifique con **not null**. Un atributo se puede declarar para que no sea nulo de la forma siguiente.

número-cuenta char(10) **not null**

SQL también soporta una restricción de integridad

unique $(A_{j1}, A_{j2}, ..., A_{jm})$

La especificación **unique** indica que los atributos $A_{j1}, A_{j2}, ..., A_{jm}$, forman una clave candidata; es decir, no puede haber dos tuplas en la relación con todos los atributos que forman la clave candidata iguales. Sin embargo, se permite que los atributos que forman la clave candidata sean nulos, a menos que se hayan declarado como **not null**. Recuérdese que un valor nulo no es igual a ningún otro valor. El tratamiento de los valores nulos aquí es el mismo que para la constructora unique definida en el apartado 7.6.4.

Un uso habitual de la cláusula **check** es el de asegurar que los valores de los atributos satisfacen unas condiciones determinadas, constituyendo así un poderoso sistema de tipos. Por ejemplo, la cláusula check en la orden de creación de una tabla para la relación sucursal comprueba que el valor para el atributo activo es un entero positivo. Considérese el siguiente ejemplo:

create table *estudiante*
(nombre **char** (15) **not null**,
id-estudiante **char** (10) **not null**,
nivel-estudios **char** (15) **not null**,
primary key *(id-estudiante),*
check *(nivel-estudios* **in** ('Graduado', 'Licenciado',
'Doctorado')))

En este ejemplo se utiliza la cláusula **check** para simular un tipo enumerado especificando que nivel-estudios debe ser «Graduado», «Licenciado» o «Doctorado».

Una relación inicialmente está vacía. Se pueden utilizar instrucciones de inserción para introducir datos en la misma. Muchas bases de datos relacionales tienen utilidades de carga para la introducción de un conjunto inicial de tuplas en una relación.

Para borrar una relación de una base de datos SQL, se utiliza la orden **drop table**. Dicha orden borra de la base de datos toda la información sobre la relación eliminada. La instrucción

drop table *r*

tiene una repercusión más drástica que

delete from *r*

La última conserva la relación *r,* pero borra todas sus tuplas. La primera, no sólo borra todas las tuplas de la relación *r,* sino también borra su esquemá. Después de que *r* se elimine no se puede insertar ninguna tupla en dicha relación, a menos que su esquema se vuelva a crear utilizando la instrucción create table.

En SQL-92, la instrucción alter table se utiliza para añadir atributos a una relación existente. La sintaxis de la instrucción es la siguiente:

alter table *r* **add** *A D*

donde *r* es el nombre de una relación existente, *A* es el nombre del atributo que se desea añadir y *D* es el dominio del atributo *A.* Se pueden eliminar atributos de una relación utilizando la orden

alter table *r* **drop** *A*

donde *r* es el nombre de una relación existente y *A* es el nombre de un atributo de la relación. Muchos sistemas de bases de datos no permiten el borrado de atributos, aunque sí permiten el borrado de una tabla completa.

7.12 SQL INCORPORADO

SQL proporciona un lenguaje de consultas declarativo muy potente. La formulación de consultas en SQL es normalmente mucho más sencilla que la formulación de las mismas en un lenguaje de programación de propósito general. Sin embargo, el acceso a una base de datos desde un lenguaje de programación de propósito general se hace necesario al menos por dos razones:

1. No todas las consultas pueden expresarse en SQL, ya que SQL no dispone del poder expresivo de un lenguaje de propósito general. Así, existen consultas que se pueden expresar en lenguajes como Pascal, C, Cobol o Fortran y que no se pueden expresar en SQL. Para formular consultas de este tipo, podemos utilizar SQL dentro de un lenguaje más potente. SQL está diseñado de tal forma que las consultas formuladas puedan optimizarse automáticamente y ejecutarse de manera eficiente (al proporcionar toda la potencia de un lenguaje de programación, la optimización automática es extremadamente difícil.

2. Las acciones no declarativas (como la impresión de un informe, la interacción con un usuario o el envío de los resultados de una consulta a una interfaz gráfica) no se pueden llevar a cabo desde el propio SQL. Normalmente las aplicaciones tienen varios componentes y la consulta o actualización de datos es uno de ellos; los demás componentes se escriben en lenguajes de programación de alto nivel. En el caso de una aplicación integrada, los programas escritos en el lenguaje de programación deben tener la capacidad de acceder a la base de datos.

La norma SQL define la utilización de SQL dentro de varios lenguajes de programación, tales como C, Cobol, Pascal, lava, PUl y Fortran. Un lenguaje en el cual se introducen consultas SQL se denomina lenguaje anfitrión y las estructuras SQL que se admiten en el lenguaje anfitrión constituyen SQL incorporado.

Los programas escritos en el lenguaje anfitrión pueden usar sintaxis SQL para acceder y actualizar datos almacenados en la base de datos. Esta forma incorporada de SQL amplía aún más la capacidad de los programadores de manipular la base de datos. En SQL incorporado, la ejecución de una consulta la realiza el sistema de base de datos y el resultado de la misma se hace disponible al programa, tupla a tupla (registro).

Un programa con SQL incorporado debe tratarse con un preprocesador especial antes de la compilación. Las consultas de SQL incorporado se sustituyen por declaraciones escritas en el lenguaje anfitrión y por llamadas a procedimientos que permiten la ejecución del acceso a la base de datos. Tras esta operación, el programa resultado se compila con el compilador del lenguaje anfitrión. Para identificar las consultas de SQL incorporado, se utiliza la instrucción EXEC SQL, que tiene la siguiente forma:

EXEC SQL <instrucción de SQL incorporado>
END-EXEC

La sintaxis exacta de las consultas en SQL incorporado depende del lenguaje dentro del que se utilicen. Por ejemplo, cuando se utilizan instrucciones de SQL dentro de un programa en C, se debe utilizar un punto y coma en lugar de END-EXEC. La incorporación de SQL en lava (denominada SQLJ) usa la sintaxis

SQL { <instrucción de SQL incorporado> };

En el programa se incluye la instrucción SQL INCLUDE para identificar el lugar donde el preprocesador debe insertar las variables especiales que se usan para la comunicación entre el programa y el sistema de base de datos. Las variables del lenguaje anfitrión se pueden utilizar en las instrucciones de SQL incorporado, pero se precederán por dos puntos (:) para distinguidas de las variables de SQL.

Las instrucciones de SQL incorporado son similares en cuanto a la sintaxis a las instrucciones SQL que se han descrito en este capítulo. Sin embargo, hay varias diferencias que se indican a continuación.

Para formular una consulta relacional se usa la instrucción **declare cursor.** El resultado de la consulta no se calcula aún. En lugar de esto, el programa debe usar las órdenes **open y fetch** (que se analizarán más adelante en este apartado) para obtener las tuplas resultado.

Considerando el esquema bancario que se ha utilizado como ejemplo en este capítulo, supóngase que se tiene una variable del lenguaje anfitrión importe y que se desea encontrar los nombres y ciudades de residencia de aquellos clientes que superan esa cantidad en alguna de sus cuentas. Se puede escribir esta consulta del modo siguiente:

EXEC SQL
declare c cursor for
select *nombre-cliente, ciudad-cliente*
from *impositor, cliente*
where *impositor.nombre-cliente*
= cliente. nombre-cliente **and**
cuenta. número-cuenta
= impositor. número -cuenta **and**
impositor:saldo > :importe
END-EXEC

En la consulta anterior, la variable **c** se denomina cursor de la consulta. Se utiliza esta variable para identificar la consulta en la instrucción **open,** que ocasiona que se evalúe la consulta, y en la instrucción **fetch,** que permite que los valores de una tupla se obtengan como variables del lenguaje anfitrión.

La instrucción **open** para el ejemplo anterior debería ser:

EXEC SQL **open** *c* END-EXEC

Esta instrucción hace que el sistema de base de datos ejecute la consulta y guarde el resultado dentro de una relación temporal. La consulta tiene una variable del lenguaje anfitrión (:importe); la consulta usa el valor de la variable en el momento en que se ejecuta la instrucción **open.**

Si se produce un error como resultado de la consulta SQL, el sistema de base de datos almacena un diagnóstico de error en las variables del área de comunicación SQL (SQLCA), cuya declaración se hace mediante la instrucción SQL INCLUDE.

Un programa de SQL incorporado ejecuta una serie de instrucciones **fetch** para obtener las tuplas del resultado. La instrucción **fetch** necesita una variable del lenguaje anfitrión por cada atributo de la relación resultado. En el ejemplo anterior se necesita una variable para almacenar el valor de nombre-cliente y otra para el valor de ciudad-cliente. Si dichas variables fuesen *nc* y *cc* respectivamente, una tupla de la relación resultado se obtendría mediante la instrucción:

EXEC **SQL fetch** *c* **into** *:nc, :cc* END EXEC

A partir de este momento el programa puede manipular las variables *nc* y *cc,* utilizando las facilidades proporcionadas por el lenguaje anfitrión.

Un único **fetch** devuelve únicamente una tupla. Si se desean obtener todas las tuplas del resultado, el programa debe tener un bucle para iterar sobre todas las tuplas. SQL incorporado ofrece una ayuda para el programador a la hora de programar esta iteración. Aunque una relación es conceptualmente un conjunto, las tuplas resultado de una consulta están físicamente en un determinado orden fijo. Cuando se ejecuta una instrucción **open,** el cursor pasa a apuntar a la primera tupla del resultado. Al ejecutarse una instrucción **fetch,** el cursor se actualiza, pasando a apuntar a la siguiente tupla del resultado. Cuando no quedan más tuplas para ser procesadas, la variable SQLSTATE en SQLCA se establece a '02000' (que significa «sin datos»). Una variable de SQLCA indica que ya no quedan tuplas por analizar en el resultado. Así, se puede utilizar un bucle **while** (o el equivalente, en función del lenguaje anfitrión) para procesar cada tupla del resultado.

La instrucción **close** se debe utilizar para indicar al sistema de base de datos que borre la relación temporal que contenía el resultado de la consulta. Para el ejemplo anterior, la sintaxis de esta instrucción será:

EXEC SQL **close** *c* END-EXEC

Las expresiones de SQL incorporado que se utilizan para modificaciones (actualización, inserción y borrado) de bases de datos no devuelven ningún resultado. Además, son más simples de expresar. Una instrucción de modificación tiene el siguiente aspecto:

EXEC SQL <cualquier actualización, inserción o borrado válidos> END-EXEC

En la expresión de modificación pueden aparecer variables del lenguaje anfitrión precedidas de dos puntos. Si se produce un error en la ejecución de la instrucción, se devuelve un diagnóstico en SQLCA.

Las relaciones de la base de datos también se pueden actualizar con cursores. Por ejemplo, si se desea añadir 100 al atributo saldo de cada cuenta donde el nombre de sucursal sea «Navacerrada», se podría declarar un cursor como:

```
declare c cursor for
select *
from account
where nombre-sucursal = 'Navacerrada'
for update
```

Se puede iterar por las tuplas ejecutando operaciones **fetch** sobre el cursor (como se mostró anteriormente) y después de obtener cada tupla se ejecuta el siguiente código:

```
update cuenta
set saldo = saldo + 1 00
where current of c
```

SQL incorporado permite a un programa en lenguaje anfitrión acceder a la base de datos, pero no proporciona ayuda para presentar los resultados al usuario o al generar informes. La mayoría de productos comerciales de bases de datos incluyen herramientas para ayudar a los programadores de aplicaciones a crear interfaces de usuario e informes con formato.

7.13. SQL DINÁMICO

El componente dinámico de SQL-92 permite que en un programa se construyan y realicen consultas SQL en tiempo de ejecución. En cambio, las instrucciones de SQL incorporado deben estar presentes en tiempo de compilación y se compilan utilizando un preprocesador de SQL incorporado. Por medio de SQL dinámico los programas pueden crear consultas SQL en tiempo de ejecución (tal vez basadas en datos introducidos por el usuario) y pueden ejecutadas inmediatamente o dejarlas preparadas para su ejecución. La preparación de una instrucción SQL dinámica la compila y los usos posteriores de la instrucción preparada usan la versión compilada.

SQL define normas para incorporar las llamadas de SQL dinámico dentro de lenguaje anfitrión, como C, como se muestra en el siguiente ejemplo.

```
char * prog_sql =«update cuenta set saldo
                 = saldo * 1.05
                 where número-cuenta = ?»
EXEC SQL prepare prog_din from :prog_sql;
char cuenta[10] = «C-101»;
EXEC SQL execute prog_din using :cuenta;
```

El programa de SQL dinámico contiene una interrogación '?' que representa una variable que se debe proporcionar en la ejecución del programa.

Sin embargo, esta sintaxis requiere extensiones para el lenguaje o un preprocesador para el lenguaje extendido. Una alternativa que usa ampliamente es una interfaz para programas de aplicación para enviar las consultas SQL o

actualizaciones a un sistema de bases de datos, sin realizar cambios en el propio lenguaje de programación.

En el resto de este apartado se examinan dos normas de conexión a una base de datos SQL y la realización de consultas y actualizaciones. Una, ODBC, es una interfaz para programas de aplicación para el lenguaje C, mientras que la otra es para Java.

Para comprender estas normas es necesario comprender el concepto de sesión SQL. El usuario o aplicación se conecta a un servidor SQL, estableciendo una sesión; ejecuta una serie de instrucciones y, finalmente, desconecta la sesión. Así, todas las actividades del usuario o aplicación están en el contexto de una sesión SQL. Además de las órdenes normales de SQL, una sesión también puede contener órdenes para comprometer el trabajo realizado en la sesión o para retrocederlo.

7.13.1. ODBC

La norma ODBC (Open Database Connectivity, conectividad abierta de bases de datos) define una forma para que un programa de aplicación se comunique con un servidor de bases de datos. ODBC define una interfaz para programas de aplicación (API, Application Program Interface) que pueden usar las aplicaciones para abrir una conexión con una base de datos, enviar consultas y actualizaciones y obtener los resultados. Las aplicaciones como las interfaces gráficas de usuario, los paquetes estadísticos y las hojas de cálculo pueden usar la misma API ODBD para conectarse a cualquier servidor de bases de datos compatible con ODBC.

Cada sistema de bases de datos que sea compatible con ODBC proporciona una biblioteca que se debe enlazar con el programa cliente. Cuando el programa cliente realiza una llamada a la API ODBC, el código de la biblioteca Se comunica con el servidor para realizar la acción solicitada y obtener los resultados.

La figura 7.9 muestra un ejemplo de código C que usa la API ODBC. El primer paso al usar ODBC para comunicarse con un servidor es configurar la conexión con el servidor. Para ello, el programa asigna en primer lugar un entorno SQL, después un manejador para la conexión a la base de datos. ODBC define los tipos HENV, HOBC y RETCOOE. El programa abre a continuación la conexión a la base de datos usando SQLConnect. Esta llamada tiene varios parámetros, incluyendo el manejador de la conexión, el servidor al que conectarse, el identificador de usuario y la contraseña. La constante SQL_NTS denota que el argumento anterior es una cadena terminada con nulo.

```
int ODBCexample()
{
    RETCODE error;
    HENV ent; /* entorno */
    HDBC con; /* conexión a la base de datos */

    SQLAllocEnv(&ent);
    SQLAllocConnect(ent, &con);
    SQLConnect(con, «aura.bell-labs.com», SQL NTS, «avi», SQL NTS, «avipasswd», SQL
    NTS);
    {

        char nombresucursal[80];
```

```
        float saldo;
        int lenOut1, lenOut2;
        HSTMT stmt;

        char * consulta = «select nombre_sucursal, sum (saldo)
                          from cuenta
                          group by nombre_sucursal»;
        SQLAllocStmt(con, &stmt);
        error = SQLExecDírect(stmt, consulta, SQL NTS);
        if (error == SQL SUCCESS) {
            SQLBindCol(stmt, 1, SQL C CHAR, nombresucursal, 80, &lenOut1);
            SQLBindCol(stmt, 2, SQL C FLOAT, &saldo, 0 , &lenOut2);
            while (SQLFetch(stmt) >= SQL SUCCESS) {
                printf (<< %s %g\n», nombresucursal, saldo);
            }
        }
        SQLFreeStmt(stmt, SQL DROP);
    }
    SQLDisconnect( con);
    SQLFreeConnect(con);
    SQLFreeEnv(ent);
}
    ç
```

Figura 7.9 *Código de ejemplo ODBC.*

Una vez que se ha configurado la conexión, el programa puede enviar órdenes SQL a la base de datos usando SQLExecDirect. Las variables del lenguaje C se pueden vincular a los atributos del resultado de la consulta, de forma que cuando se obtenga una tupla resultado usando SQLFetch, sus valores de atributo se almacenan en las variables C correspondientes. La función SQLBindCol realiza esta tarea; el segundo argumento identifica la posición del atributo en el resultado de la consulta, y el tercer argumento indica la conversión de tipos de SQL a C requerida. El siguiente argumento da la dirección de la variable. Para los tipos de longitud variables como los arrays de caracteres, los dos últimos argumentos dan la longitud máxima de la variable y una ubicación donde almacenar la longitud actual cuando se obtenga una tupla. Un valor negativo devuelto para el campo longitud indica que el valor es **null**.

La instrucción SQLFetch está en un bucle **while** que se ejecuta hasta que SQLFetch devuelva un valor diferente de SQL_SUCCESS. En cada obtención de valores, el programa los almacena en variables C como se especifica en las llamadas en SQLBindCol e imprime estos valores.

Al final de la sesión, el programa libera el manejador, se desconecta de la base de datos y libera la conexión y los manejadores del entorno SQL. Un buen estilo de programación requiere que el resultado de cada función se comprueba para asegurarse de que no haya errores; se han omitido la mayoría de estas comprobaciones por brevedad.

Es posible crear una instrucción SQL con parámetros; por ejemplo, considérese la instrucción **insert into account values(?,?, ?)**. Los interrogantes son resguardos para los valores que se proporcionarán después. Esta instrucción se puede «preparar», es decir, compilar en la base de datos y ejecutar repetidamente proporcionando los valores reales para los resguardos -en este caso proporcionando un número de cuenta, nombre de sucursal y saldo para la relación cuenta.

ODBC define funciones para varias tareas, tales como hallar todas las relaciones en la base de datos y los nombres y tipos de las columnas del resultado de una consulta o una relación de la base de datos.

De forma predeterminada, cada instrucción SQL se trata como una transacción separada que se compromete automáticamente. La llamada SQLSetConnectOption(con, SQL_AUTOCOMMIT, 0) desactiva el compromiso automático en la conexión con, y las transacciones se deben comprometer explícitamente con SQLTransact(con, SQL_COMMIT) o retroceder con SQLTransact(con, SQL_ROLLBACK).

Las versiones más recientes de la norma ODBC añaden nueva funcionalidad. Cada versión define niveles de acuerdo que especifican subconjuntos de la funcionalidad definida por el estándar. Una implementación ODBC puede proporcionar sólo las características básicas o puede proporcionar características más avanzadas (nivel 1 o 2). El nivel 1 requiere soporte para la obtención de información del catálogo, como la información sobre las relaciones existentes y los tipos de sus atributos. El nivel 2 requiere más características, como la capacidad de enviar y obtener arrays de valores de parámetros y para obtener información del catálogo más detallada.

Las normas más recientes de SQL (SQL-92 y SQL: 1999) definen una **interfaz en el nivel de llamada** (Call-Level Interface, CLI) que es similar a la interfaz ODBC, pero con algunas pequeñas diferencias.

7.13.2. JDBC

La norma **JDBC** define una API que pueden usar los programas Java para conectarse a los servidores de bases de datos (la palabra JDBC fue originalmente abreviatura de «Java Database Connectivity» -conectividad de bases de datos con Java- pero la forma completa ya no se usa). La figura 7.10 muestra un ejemplo de un programa Java que usa la interfaz JDBC. El programa debe en primer lugar abrir una conexión a una base de datos y después ejecutar instrucciones SQL, pero antes de abrir una conexión, carga los controladores adecuados para la base de datos usando Class.forName. El primer parámetro de la llamada getConnection especifica el nombre de la máquina en la que se ejecuta el servidor (en este caso, aura.bell-labs.com) y el número de puerto que usa para la comunicación (en este caso, 2000). El parámetro también especifica el esquema de la base de datos a usar (en este caso, bdbanco), ya que un servidor de bases de datos puede dar soporte a varios esquemas. El primer parámetro también especifica el protocolo a usar para la comunicación con la base de datos (en este caso, jdbc:oracle:thin:). Obsérvese que JDBC especifica sólo la API, no el protocolo de comunicación. Un controlador JDBC puede dar soporte a varios protocolos y se debe especificar el compatible con la base de datos y el controlador. Los otros dos argumentos de getConnection son un identificador de usuario y una contraseña.

```
public static void ejemploJDBC (String idbd, String idusuario, String contraseña)
{
    try
    {
        Class. forName («oracle.jdbc.driver.OracleDriver»);
        Connection con = DriverManager.getConnection
                ( «jdbc:oracle :thin:@aura.bell-labs.com:2000:bdbanco»,
```

```
                    idusuario, contraseña);
        Statement stmt = con.createStatement();
        try {
            stmt.executeUpdate(
                «insert into cuenta values('C-9732', 'Navacerrada', 1200) »);
        } catch (SQLException sqle)
        {
            System.out.println(«No se pudo insertar la tupla. » + sqle);
        }
        ResultSet rset = stmt.executeQuery
                («select nombre_sucursal, avg (saldo)
                from cuenta
                group by nombre_sucursal»);
        while (rset.next()) {
            System.out.println(rset.getString( «nombre_sucursal») + « » +
                    rset.getFloat(2));
        }
        stmt.close();
        con.close();
        }
        catch (SQLException sqle)
        {
            System.out.println(<<SQLException : » + sqle);
    }
}
```

FIGURA 7.10. *Un ejemplo de código JDBC.*

El programa crea a continuación un manejador para la conexión y lo usa para ejecutar una instrucción SQL y obtener los resultados. En nuestro ejemplo, stmt.executeUpdate ejecuta una instrucción de actualización. El constructor try { ... } catch { ... } permite capturar cualquier excepción (condición de error) que surjan cuando se realizan las llamadas JDBC, e imprime un mensaje apropiado para el usuario.

El programa puede ejecutar una consulta usando stmt.executeQuery. Puede obtener el conjunto de filas en el resultado en ResultSet y leer tupla a tupla usando la función next() en el conjunto de resultados. La figurara 3.10 muestra dos formas de obtener los valores de los atributos en una tupla: usando el nombre del atributo *(nombre-sucursal)* y usando la posición del atributo (2, para denotar el segundo atributo).

También se puede crear una instrucción preparada en la que algunos valores se reemplacen por «?», especificando que los valores actuales se proporcionarán más tarde. Se pueden proporcionar los valores usando setString(). La base de datos puede compilar la consulta cuando esté preparada, y cada vez que se ejecute (con nuevos valores), la base de datos puede rehusar la forma compilada previamente de la consulta. El fragmento de código de la figura 7.11 muestra cómo se pueden usar las instrucciones preparadas.

```
PreparedStatement pStmt = con.prepareStatement(
«insert into cuenta values(?,?,?)»);
pStmt.setString(1, «C-9732»);
pStmt.setString(2, "Navacerrada»);
pStmt.setlnt(3, 1200);
pStmt.executeUpdate();
pStmt.setString(1, «C-9733»);
pStmt.executeUpdate();
```

FIGURA 7.11. *Instrucciones preparadas en código JDBC.*

JDBC proporciona otras características, como los **conjuntos de resultados actualizables.** Puede crear un conjunto de resultados actualizable a partir de una consulta que realice una selección o una proyección de una relación de la base de datos. Una actualización de una tupla en el conjunto de resultados es consecuencia de una actualización de la tupla correspondiente de la relación de la base de datos. JDBC también proporciona una API para examinar esquemas de la base de datos para encontrar los tipos de atributos de un conjunto de resultados.

7.14. OTRAS CARACTERÍSTICAS DE SQL

El lenguaje SQL ha crecido durante las dos décadas pasadas desde un lenguaje simple con pocas características a un lenguaje ciertamente complejo con características para satisfacer a muchos tipos diferentes de usuarios. Se trataron los fundamentos de SQL anteriormente en este capítulo. En este apartado se introducen al lector algunas de las características más complejas de SQL.

7.14.1. Esquemas. catálogos y entornos

Para comprender la motivación de los esquemas y los catálogos, considérese cómo se denominan los archivos en un sistema de archivos. Los sistemas de archivos originales eran planos; es decir, todos los archivos se almacenaban en un directorio. Los sistemas de archivos de la generación actual tienen por supuesto una estructura de directorios, con archivos almacenados en subdirectorios. Para denominar unívocamente un archivo se debe especificar el nombre completo de la ruta del archivo, por ejemplo /usuarios/avildb-book/capítul04.tex.

Al igual que en los primeros sistemas de archivos, los primeros sistemas de bases de datos tenían un único espacio de nombres para todas las relaciones. Los usuarios tenían que coordinarse para asegurarse de que no intentaban usar el mismo nombre para relaciones diferentes. Los sistemas de bases de datos actuales proporcionan una jerarquía de tres niveles para denominar a las relaciones. El nivel superior de la jerarquía consiste en catálogos, cada uno de los cuales puede contener esquemas. Los objetos SQL tales como las relaciones y las vistas están contenidos en un esquema.

Para realizar cualquier acción sobre una base de datos, un usuario (o un programa) debe en primer lugar conectarse a la base de datos. El usuario debe proporcionar el nombre de usuario y generalmente una contraseña secreta para

comprobar la identidad del usuario, como se vio en los ejemplos de ODBC y JDBC. Cada usuario tiene un catálogo y esquema predeterminados, y la combinación es única para el usuario. Cuando un usuario se conecta a un sistema de bases de datos, el catálogo y esquema predeterminados se configuran para la conexión; esto se corresponde con el directorio actual establecido para el directorio inicial del usuario cuando el usuario inicia la sesión en el sistema operativo.

Para identificar una relación unívocamente, se debe usar un nombre con tres partes, por ejemplo:

catálog05.esquema-banco.cuenta

Se puede omitir el componente catálogo y, en ese caso, la parte catálogo del nombre se considera el catálogo predeterminado para la conexión. Así, si catálogo5 es el catálogo predeterminado, se puede usar esquema-banco.cuenta para identificar la misma relación unívocamente. Además, también se puede omitir el nombre del esquema, y la parte esquema del nombre es de nuevo considerada como el esquema predeterminado de la conexión. Así, se puede usar tan solo cuenta si el catálogo predeterminado es catálogo5 y el esquema predeterminado es esquema-banco.

Con varios catálogos y esquemas disponibles pueden trabajar independientemente diferentes aplicaciones y usuarios sin preocuparse acerca de la coincidencia de nombres. Además, pueden ejecutarse varias versiones de una aplicación (una versión de producción y otras de test) en el mismo sistema de bases de datos.

El catálogo y esquema predeterminados son parte de un **entorno SQL** que se configura por cada conexión. El entorno también contiene el identificador de usuario (también conocido como identificador de autorización). Todas las consultas SQL habituales, incluyendo las instrucciones LDD y LMD operan en el contexto de un esquema. Los esquemas se pueden crear· o eliminar mediante las instrucciones **create schema** o **drop schema**. La creación y borrado de catálogos es dependiente de la implementación y no es parte de la norma SQL.

7.14.2. Extensiones procedimentales y procedimientos almacenados

SQL proporciona un lenguaje de módulos, que permite definir los procedimientos en SQL. Un módulo contiene normalmente varios procedimientos SQL. Cada procedimiento tiene un nombre, parámetros opcionales y una instrucción SQL. Una extensión del lenguaje estándar SQL-92 también permite constructoras procedimentales, tales como **for**, **while** e **if-then-else**, e instrucciones SQL compuestas (varias instrucciones SQL entre **begin** y **end**).

Los procedimientos se pueden almacenar en la base de datos y ejecutarse con la instrucción **call**. Estos procedimientos se denominan también procedimientos almacenados. Los procedimientos almacenados son particularmente útiles porque permiten que las operaciones de la base de datos se encuentren disponibles a aplicaciones externas, sin exponer ninguno de los detalles internos de la base de datos.

7.15 RESUMEN

Los sistemas de bases de datos comerciales no utilizan los lenguajes de consulta formales tales como el modelo relacional. El ampliamente usado lenguaje SQL, que se ha estudiado en este capítulo, está basado en el álgebra relacional formal, pero con mucho «azúcar sintáctico».

SQL incluye varias constructoras del lenguaje para las consultas sobre la base de datos. Todas las operaciones del álgebra relacional, incluyendo las operaciones del álgebra relacional extendida, se pueden expresar en SQL. SQL también permite la ordenación de los resultados de una consulta en términos de los atributos. Las relaciones de vistas se pueden definir como relaciones que contienen el resultado de consultas. Las vistas son útiles para ocultar información innecesaria y para recolectar información de más de una relación en una única vista.

Las vistas temporales definidas con la cláusula **with** también son útiles para descomponer consultas complejas en partes más pequeñas y fáciles de entender. SQL incluye constructoras para insertar, actualizar y borrar información. Una transacción consiste en una secuencia de operaciones que deben ser atómicas. Es decir, todas las operaciones se realizan con éxito o ninguna. En la práctica, si una transacción no se puede completar con éxito, todas las acciones parciales realizadas se deshacen.

Las modificaciones sobre la base de datos pueden conducir a la generación de valores nulos en las tuplas. Se estudió cómo se podían introducir los valores nulos y la forma en que SQL maneja las consultas sobre las relaciones que contienen estos valores.

El lenguaje de definición de datos SQL se usa para crear relaciones con los esquemas especificados. El LDD de SQL soporta varios tipos incluyendo **date** y **time.**

Las consultas SQL se pueden llamar desde lenguajes anfitriones mediante SQL incorporado y dinámico. Las normas ODBC y JDBC definen interfaces para programas de aplicación para acceder a bases de datos SQL desde los programas en lenguaje C y Java. Los programadores usan cada vez más estas API para acceder a bases de datos.

También se vio una visión general de algunas características avanzadas de SQL, tales como las extensiones procedimentales, los catálogos, los esquemas y los procedimientos almacenados.

REFERENCIA BIBLIOGRÁFICA

ANÁLISIS Y DISEÑO DE BASES DE DATOS
I. T. HAWRYSKIEWYCZ
MEGABYTE NORIEGA EDITORES
MEXICO, 2000

ANÁLISIS Y DISEÑO DE SISTEMAS DE INFORMACIÓN
JAMES A. SEEN
MCGRAW HILL
MÉXICO, 1992

BASES DE DATOS
GEORGES GARDARIN
PARANINFO
ESPAÑA, 1987

CONCEPCIÓN Y DISEÑO DE BASES DE DATOS: DEL MODELO E/R AL MODELO RELACIONAL
ADORACIÓN DE MIGUEL CASTAÑO, MARIO GERARDO PIATTINI VELTHUIS
ADDISON-WESLEY IBEROAMERICANA, RA-MA
ESPAÑA, 1993

DISEÑO DE BASES DE DATOS PROBLEMAS RESUELTOS
ADORACIÓN DE MIGUEL CASTAÑO, et. al.
ALFAOMEGA RA-MA
COLOMBIA, 2001

DISEÑO Y ADMINISTRACION DE BASES DE DATOS, SEGUNDA EDICION
GARY W. HANSEN, JAMES V. HANSEN
PRENTICE HALL
MÉXICO, 1997

FUNDAMENTOS DE BASES DE DATOS
ABRAHAM SILVERSCHATZ, HENRY F. KORT
MCGRAW HILL
MÉXICO, 1988

FUNDAMENTOS DE BASES DE DATOS, CUARTA EDICIÓN
ABRAHAM SILVERSCHATZ, HENRY F. KORT, S. SUDARSHAN
MCGRAW HILL
ESPAÑA, 2002

ORGANIZACIÓN DE LAS BASES DE DATOS
JAMES MARTIN
PRENTICE HALL
MEXICO 1997

SISTEMAS DE BASE DE DATOS ADMINISTRACIÓN Y USO
ALICE Y. H. TSAI
PRENTICE-HALL HISPANOAMERICA, S.A.
MEXICO, 1998

REFERENCIA ELECTRÓNICA

http://www.rincondelbago.com

http://www.monografias.com

http://es.wikipedia.org

http://www.abcdatos.com

http://www.unav.es/SI/servicios/manuales.html

http://www.unav.es/SI/servicios/manuales/Access2003.pdf

http://elies.rediris.es/elies9/index-4.htm

http://www.tramullas.com/documatica/2-3.html

http://www.programatium.com/sql.htm

http://www.foxserv.net/

http://mysql.conclase.net

http://www.mysql.com/

http://dev.mysql.com/downloads/mysql/4.1.html

http://dev.mysql.com/downloads/mysql/6.0.html

http://sistemas.itlp.edu.mx/tutoriales/basedat1/tema1_9.htm

http://iteso.mx/~jpgonzal/bd2d.html