Argentina programa

Desarrollo de aplicaciones JAVA

Guía III

"Base de Datos"

BASE DE DATOS:

Un sistema gestor de bases de datos (SGBD) consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a dichos datos. La colección de datos, normalmente denominada **base de datos**, contiene información relevante para una empresa. El objetivo principal de un SGBD es proporcionar una forma de almacenar y recuperar la información de una base de datos de manera que sea tanto *práctica* como *eficiente*.

Los sistemas de bases de datos se diseñan para gestionar grandes cantidades de información. La gestión de los datos implica tanto la definición de estructuras para almacenar la información como la provisión de mecanismos para la manipulación de la información. Además, los sistemas de bases de datos deben proporcionar la fiabilidad de la información almacenada, a pesar de las caídas del sistema o los intentos de acceso sin autorización. Si los datos van a ser compartidos entre diversos usuarios, el sistema debe evitar posibles resultados anómalos.

Dado que la información es tan importante en la mayoría de las organizaciones, los científicos informáticos han desarrollado un amplio conjunto de conceptos y técnicas para la gestión de los datos.

APLICACIONES DE LOS SISTEMAS DE BASES DE DATOS

Las bases de datos son ampliamente usadas. Las siguientes son algunas de sus aplicaciones más representativas:

- **Banco**. Para información de los clientes, cuentas y préstamos, y transacciones bancarias.
- **Líneas aéreas**. Para reservas e información de planificación. Las líneas aéreas fueron de los primeros en usar las bases de datos de forma distribuida geográficamente (los terminales situados en todo el mundo accedían al sistema de bases de datos centralizado a través de las líneas telefónicas y otras redes de datos).
- **Universidades**. Para información de los estudiantes, matrículas de las asignaturas y cursos.
- **Transacciones de tarjetas de crédito**. Para compras con tarjeta de crédito y generación mensual de extractos.
- **Telecomunicaciones**. Para guardar un registro de las llamadas realizadas, generación mensual de facturas, manteniendo el saldo de las tarjetas telefónicas de prepago y para almacenar información sobre las redes de comunicaciones.
- **Finanzas**. Para almacenar información sobre grandes empresas, ventas y compras de documentos formales financieros, como bolsa y bonos.
- **Ventas**. Para información de clientes, productos y compras.
- **Producción**. Para la gestión de la cadena de producción y para el seguimiento de la producción de elementos en las factorías, inventarios de elementos en almacenes y pedidos de elementos.
- **Recursos humanos**. Para información sobre los empleados, salarios, impuestos y beneficios, y para la generación de las nóminas.

Como esta lista ilustra, las bases de datos forman una parte esencial de casi todas las empresas actuales.

A lo largo de las últimas cuatro décadas del siglo veinte, el uso de las bases de datos creció en todas las empresas. En los primeros días, muy pocas personas interactuaron directamente con los sistemas de bases de datos, aunque sin darse cuenta interactuaron con bases de datos indirectamente (con los informes impresos como extractos de tarjetas de crédito, o mediante agentes como cajeros de bancos y agentes de reserva de líneas aéreas). Después vinieron los cajeros automáticos y permitieron a los usuarios interactuar con las bases de datos. Las interfaces telefónicas con los computadores (sistemas de respuesta vocal interactiva) también permitieron a los usuarios manejar directamente las bases de datos. Un llamador podía marcar un número y pulsar teclas del teléfono para introducir información o para seleccionar opciones alternativas, para determinar las horas de llegada o salida, por ejemplo, o para matricularse de asignaturas en una universidad.

La revolución de Internet a finales de la década de 1990 aumentó significativamente el acceso directo del usuario a las bases de datos. Las organizaciones convirtieron muchas de sus interfaces telefónicas a las bases de datos en interfaces Web, y pusieron disponibles en línea muchos servicios. Por ejemplo, cuando se accede a una tienda de libros en línea y se busca un libro o una colección de música se está accediendo a datos almacenados en una base de datos. Cuando se solicita un pedido en línea, el pedido se almacena en una base de datos. Cuando se accede a un banco en un sitio Web y se consulta el estado de la cuenta y los movimientos, la información se recupera del sistema de bases de datos del banco. Cuando se accede a un sitio Web, la información personal puede ser recuperada de una base de datos para seleccionar los anuncios que se deberían mostrar. Más aún, los datos sobre los accesos Web pueden ser almacenados en una base de datos.

Así, aunque las interfaces de datos ocultan detalles del acceso a las bases de datos, y la mayoría de la gente ni siquiera es consciente de que están interactuando con una base de datos, el acceso a las bases de datos forma una parte esencial de la vida de casi todas las personas actualmente.

La importancia de los sistemas de bases de datos se puede juzgar de otra forma: actualmente, los vendedores de sistemas de bases de datos como Oracle están entre las mayores compañías software en el mundo, y los sistemas de bases de datos forman una parte importante de la línea de productos de compañías más diversificadas, como Microsoft e IBM.

Es importante que leas el Anexo ya que encontrarás allí información acerca de la herramienta que utilizaremos para poner en práctica lo que veamos en esta guía utilizando como gestor de base de datos MySQL.

Modelo relacional

En el modelo relacional se utiliza un grupo de tablas para representar los datos y las relaciones entre ellos.

Cada tabla está compuesta por varias columnas, y cada columna tiene un nombre único. En la Figura 1.3 se presenta un ejemplo de base de datos relacional consistente en tres tablas: la primera

muestra los clientes de un banco, la segunda, las cuentas, y la tercera, las cuentas que pertenecen a cada cliente.

id-cliente	*nombre-cliente*	*calle-cliente*	*ciudad-cliente*
19.283.746	González	Arenal	La Granja
01.928.374	Gómez	Carretas	Cerceda
67.789.901	López	Mayor	Peguerinos
18.273.609	Abril	Preciados	Valsaín
32.112.312	Santos	Mayor	Peguerinos
33.666.999	Rupérez	Ramblas	León
01.928.374	Gómez	Carretas	Cerceda

(a) La tabla *cliente*

número-cuenta	*saldo*
C-101	500
C-215	700
C-102	400
C-305	350
C-201	900
C-217	750
C-222	700

(b) La tabla *cuenta*

id-cliente	*número-cuenta*
19.283.746	C-101
19.283.746	C-201
01.928.374	C-215
67.789.901	C-102
18.273.609	C-305
32.112.312	C-217
33.666.999	C-222
01.928.374	C-201

(b) La tabla *impositor*

FIGURA 1.3. Ejemplo de base de datos relacional.

La primera tabla, la tabla cliente, muestra, por ejemplo, que el cliente cuyo identificador es 19.283.746 se llama González y vive en la calle Arenal sita en La Granja. La segunda tabla, cuenta, muestra que las cuentas C-101 tienen un saldo de 500 € y la C-201 un saldo de 900 € respectivamente.

La tercera tabla muestra las cuentas que pertenecen a cada cliente. Por ejemplo, la cuenta C-101 pertenece al cliente cuyo identificador es 19.283.746 (González), y los clientes 19.283.746 (González) y 01.928.374 (Gómez) comparten el número de cuenta C-201 (pueden compartir un negocio).

El modelo relacional es un ejemplo de un modelo basado en registros. Los modelos basados en registros se denominan así porque la base de datos se estructura en registros de formato fijo de varios tipos. Cada tabla contiene registros de un tipo particular. Cada tipo de registro define un

número fijo de campos, o atributos. Las columnas de la tabla corresponden a los atributos del tipo de registro.

No es difícil ver cómo se pueden almacenar las tablas en archivos. Por ejemplo, un carácter especial (como una coma) se puede usar para delimitar los diferentes atributos de un registro, y otro carácter especial (como un carácter de nueva línea) se puede usar para delimitar registros. El modelo relacional oculta tales detalles de implementación de bajo nivel a los desarrolladores de bases de datos y usuarios.

El modelo de datos relacional es el modelo de datos más ampliamente usado, y una amplia mayoría de sistemas de bases de datos actuales se basan en el modelo relacional.

El modelo relacional se encuentra a un nivel de abstracción inferior al modelo de datos E-R. Los diseños de bases de datos a menudo se realizan en el modelo E-R (*ilustración 1)*, y después se traducen al modelo relacional (*ilustración 2)*. Por ejemplo, es fácil ver que las tablas cliente y cuenta corresponden a los conjuntos de entidades del mismo nombre, mientras que la tabla impositor corresponde al conjunto de relaciones impositor.

Nótese también que es posible crear esquemas en el modelo relacional que tengan problemas tales como información duplicada innecesariamente. Por ejemplo, supongamos que se almacena número-cuenta como un atributo del registro cliente. Entonces, para representar el hecho de que las cuentas C-101 y C-201 pertenecen ambas al cliente González (con identificador de cliente 19.283.746) sería necesario almacenar dos filas en la tabla cliente. Los valores de nombre-cliente, calle-cliente y ciudad-cliente de González estarían innecesariamente duplicados en las dos filas.

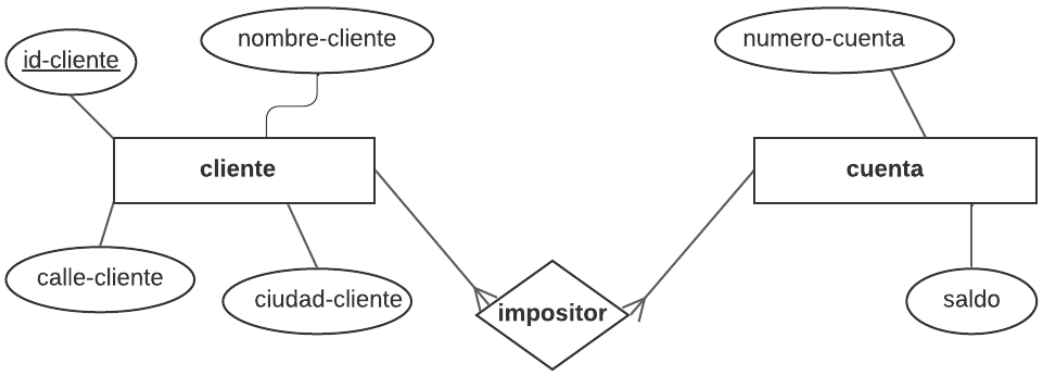

Ilustración 1-Modelo E-R

[1.1]

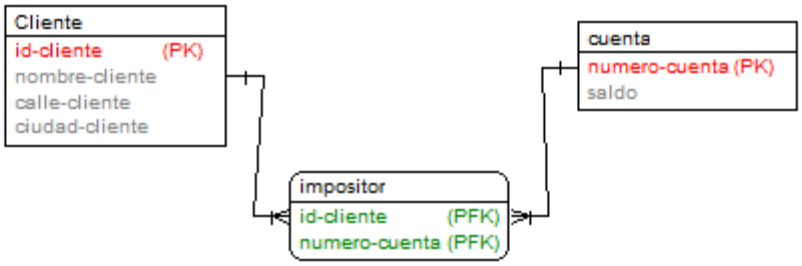

Ilustración 2-Modelo Relacional

LENGUAJES DE BASES DE DATOS

Un sistema de bases de datos proporciona un lenguaje de definición de datos para especificar el esquema de la base de datos y un lenguaje de manipulación de datos para expresar las consultas a la base de datos y las modificaciones. En la práctica, los lenguajes de definición y manipulación de datos no son dos lenguajes separados; en su lugar simplemente forman partes de un único lenguaje de bases de datos, tal como SQL, ampliamente usado.

Lenguaje de definición de datos

Un esquema de base de datos se especifica mediante un conjunto de definiciones expresadas mediante un lenguaje especial llamado lenguaje de definición de datos (LDD).

Por ejemplo, la siguiente instrucción en el lenguaje SQL define la tabla cuenta:

create table ***cuenta***

(número-cuenta char(10), saldo integer)

La ejecución de la instrucción LDD anterior crea la tabla cuenta. Además, actualiza un conjunto especial de tablas denominado diccionario de datos o directorio de datos.

Un diccionario de datos contiene metadatos, es decir, datos acerca de los datos. El esquema de una tabla es un ejemplo de metadatos. Un sistema de base de datos consulta el diccionario de datos antes de leer o modificar los datos reales.

Especificamos el almacenamiento y los métodos de acceso usados por el sistema de bases de datos por un conjunto de instrucciones en un tipo especial de LDD denominado lenguaje de

almacenamiento y definición de datos. Estas instrucciones definen los detalles de implementación de los esquemas de base de datos, que se ocultan usualmente a los usuarios.

Los valores de datos almacenados en la base de datos deben satisfacer ciertas restricciones de consistencia. Por ejemplo, supóngase que el saldo de una cuenta no debe caer por debajo de 100 €. El LDD proporciona facilidades para especificar tales restricciones. Los sistemas de bases de datos comprueban estas restricciones cada vez que se actualiza la base de datos.

Lenguaje de manipulación de datos

La manipulación de datos es:

• La recuperación de información almacenada en la base de datos.

• La inserción de información nueva en la base de datos.

• El borrado de información de la base de datos.

• La modificación de información almacenada en la base de datos.

Un lenguaje de manipulación de datos (LMD) es un lenguaje que permite a los usuarios acceder o manipular los datos organizados mediante el modelo de datos apropiado. Hay dos tipos básicamente:

• LMDs procedimentales. Requieren que el usuario especifique qué datos se necesitan y cómo obtener esos datos.

• LMDs declarativos (también conocidos como LMDs no procedimentales). Requieren que el usuario especifique qué datos se necesitan sin especificar cómo obtener esos datos.

Los LMDs declarativos son más fáciles de aprender y usar que los LMDs procedimentales. Sin embargo, como el usuario no especifica cómo conseguir los datos, el sistema de bases de datos tiene que determinar un medio eficiente de acceder a los datos. El componente LMD del lenguaje SQL es no procedimental.

Una consulta es una instrucción de solicitud para recuperar información. La parte de un LMD que implica recuperación de información se llama *lenguaje de consultas*. Aunque técnicamente sea incorrecto, en la práctica se usan los términos lenguaje de consultas y lenguaje de manipulación de datos como sinónimos.

Esta consulta en el lenguaje SQL encuentra el nombre del cliente cuyo identificador de cliente es 19.283.746:

select *cliente.nombre-cliente*

from *cliente*

where *cliente.id-cliente = 19283746*

La consulta especifica que las filas de (from) la tabla cliente donde (where) el id-cliente es 19 283 746 se debe recuperar, y que se debe mostrar el atributo nombre-cliente de estas filas. Si se ejecutase la consulta con la tabla de la Figura 1.3, se mostraría el nombre González.

Las consultas pueden involucrar información de más de una tabla. Por ejemplo, la siguiente consulta encuentra el saldo de todas las cuentas pertenecientes al cliente cuyo identificador de cliente es 19 283 746.

select *cuenta.saldo*

from *impositor, cuenta*

where *impositor.id-cliente = 19283746*

and *impositor.número-cuenta = cuenta.númerocuenta*

Si la consulta anterior se ejecutase con las tablas de la *Figura 1.3*, el sistema encontraría que las dos cuentas denominadas C-101 y C-201 pertenecen al cliente 19 283 746 e imprimiría los saldos de las dos cuentas, es decir, 500 y 900 €.

Hay varios lenguajes de consulta de bases de datos en uso, ya sea comercialmente o experimentalmente. Se estudiará el lenguaje de consultas más ampliamente usado, SQL.

Los niveles de abstracción que se discutieron en el Apartado 1.3 se aplican no solo a la definición o estructuración de datos, sino también a la manipulación de datos. En el nivel físico se deben definir algoritmos que permitan un acceso eficiente a los datos. En los niveles superiores de abstracción se enfatiza la facilidad de uso. El objetivo es proporcionar una interacción humana eficiente con el sistema. El componente procesador de consultas del sistema de bases de datos traduce las consultas LMD en secuencias de acciones en el nivel físico del sistema de bases de datos.

Estructura de un Sistema de Base de Datos:

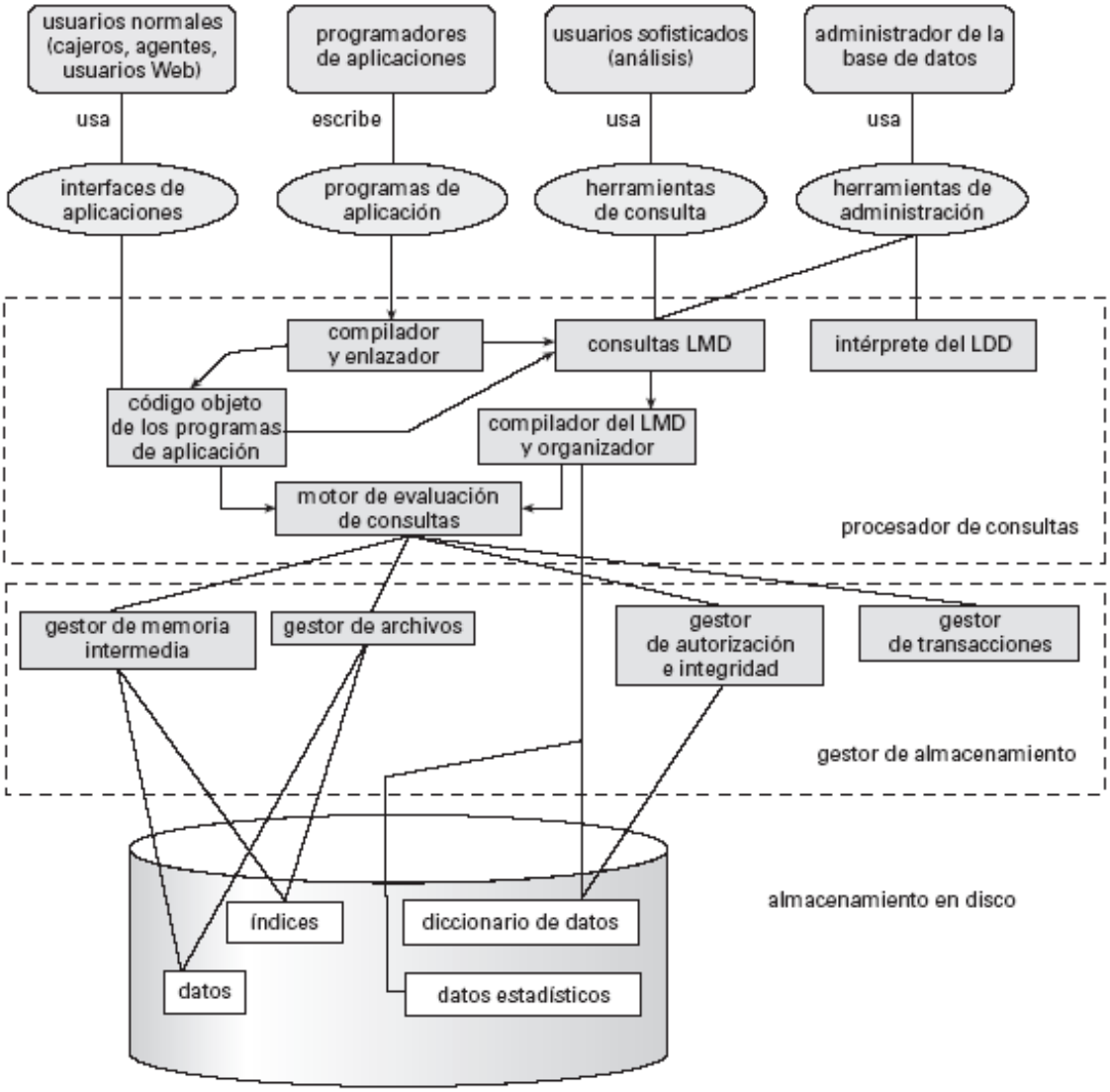

FIGURA 1.4. Estructura del sistema.

EL MODELO RELACIONAL

El modelo relacional se ha establecido actualmente como el principal modelo de datos para las aplicaciones de procesamiento de datos. Ha conseguido la posición principal debido a su simplicidad, que facilita el trabajo del programador en comparación con otros modelos anteriores como el de red y el jerárquico.

LA ESTRUCTURA DE LAS BASES DE DATOS RELACIONALES

Una base de datos relacional consiste en un conjunto de **tablas**, a cada una de las cuales se le asigna un nombre exclusivo. Cada fila de la tabla representa una *relación* entre un conjunto de valores. Dado que cada tabla es un conjunto de dichas relaciones, hay una fuerte correspondencia entre el concepto de *tabla* y el concepto matemático de *relación*, del que toma su nombre el modelo de datos relacional. A continuación se introduce el concepto de relación.

En esta unidad se utilizarán varias relaciones diferentes para ilustrar los conceptos subyacentes al modelo de datos relacional. Estas relaciones representan parte de una entidad bancaria.

Estructura básica

Considérese la tabla cuenta de la Figura 3.1. Tiene tres cabeceras de columna: número-cuenta, nombre-sucursal y saldo. Se puede hacer referencia a estas cabeceras como atributos. Para cada atributo hay un conjunto de valores permitidos, llamado dominio de ese atributo. Para el atributo nombre-sucursal, por ejemplo, el dominio es el conjunto de los nombres de las sucursales.

número-cuenta	*nombre-sucursal*	*saldo*
C-101	Centro	500
C-102	Navacerrada	400
C-201	Galapagar	900
C-215	Becerril	700
C-217	Galapagar	750
C-222	Moralzarzal	700
C-305	Collado Mediano	350

FIGURA 3.1. La relación *cuenta*.

Como las tablas son esencialmente relaciones, se utilizarán los términos matemáticos relación y tupla en lugar de los términos tabla y fila.

En la relación cuenta de la Figura 3.1 hay siete tuplas.

El orden en que aparecen las tuplas es irrelevante, dado que una relación es un conjunto de tuplas. Así, si las tuplas de una relación se muestran ordenadas como en la Figura 3.1, o desordenadas, como en la Figura 3.2, no importa; las relaciones de estas figuras son las mismas, ya que ambas contienen el mismo conjunto de tuplas.

número-cuenta	*nombre-sucursal*	*saldo*
C-101	Centro	500
C-215	Becerril	700
C-102	Navacerrada	400
C-305	Collado Mediano	350
C-201	Galapagar	900
C-222	Moralzarzal	700
C-217	Galapagar	750

FIGURA 3.2. La relación *cuenta* con las tuplas desordenadas.

Esquema de la base de datos

Cuando se habla de bases de datos se debe diferenciar entre el **esquema de la base de datos**, o diseño lógico de la misma, y el **ejemplar de la base de datos**, que es una instantánea de los datos de la misma en un momento dado.

El concepto de relación se corresponde con el concepto de variable de los lenguajes de programación. El concepto de **esquema de la relación** se corresponde con el concepto de definición de tipos de los lenguajes de programación.

Resulta conveniente dar un nombre a los esquemas de las relaciones, igual que se dan nombres a las definiciones de tipos en los lenguajes de programación. Se adopta el convenio de utilizar nombres en minúsculas para las relaciones y nombres que comiencen por una letra mayúscula para los esquemas de las relaciones. Siguiendo esta notación se utilizará *Esquema-cuenta* para denotar el esquema de la relación de la relación *cuenta*. Por tanto, *Esquema-cuenta* = (*número-cuenta*, *nombre-sucursal*, *saldo*)

Se denota el hecho de que *cuenta* es una relación de *Esquema-cuenta* mediante *cuenta* (*Esquema-cuenta*)

En general, los esquemas de las relaciones incluyen una lista de los atributos y de sus dominios correspondientes.

El concepto de **ejemplar de relación** se corresponde con el concepto de valor de una variable en los lenguajes de programación. El valor de una variable dada puede cambiar con el tiempo; de manera parecida, el contenido del ejemplar de una relación puede cambiar con el tiempo cuando la relación se actualiza. Sin embargo, se suele decir simplemente «relación» cuando realmente se quiere decir «ejemplar de la relación».

nombre de la sucursal	*ciudad de la sucursal*	*activos*
Galapagar	Arganzuela	7.500
Centro	Arganzuela	9.000.000
Becerril	Aluche	2.000
Segovia	Cerceda	3.700.000
Navacerrada	Aluche	1.700.000
Navas de la Asunción	Alcalá de Henares	1.500
Moralzarzal	La Granja	2.500
Collado Mediano	Aluche	8.000.000

FIGURA 3.3. La relación *sucursal*.

Como ejemplo de ejemplar de una relación, considérese la relación *sucursal* de la Figura 3.3. El esquema de esa relación es

Esquema-relación = (*nombre-sucursal, ciudad-sucursal, activos*)

Obsérvese que el atributo *nombre de la sucursal* aparece tanto en *Esquema-sucursal* como en *Esquema-cuenta*. Esta duplicidad no es una coincidencia. Más bien, utilizar atributos comunes en los esquemas de las relaciones es una manera de relacionar las tuplas de relaciones diferentes. Por ejemplo, supóngase que se desea obtener información sobre todas las cuentas abiertas en sucursales ubicadas en Arganzuela. Primero se busca en la relación *sucursal* para encontrar los nombres de todas las sucursales sitas en Arganzuela. Luego, para cada una de ellas, se mira en la relación *cuenta* para encontrar la información sobre las cuentas abiertas en esa sucursal. Esto no es sorprendente: recuérdese que los atributos que forma la clave primaria de un conjunto de entidades fuertes aparecen en la tabla creada para representar el conjunto de entidades, así como en las tablas creadas para crear relaciones en las que participan el conjunto de entidades.

nombre-cliente	*calle-cliente*	*ciudad-cliente*
Abril	Preciados	Valsaín
Amo	Embajadores	Arganzuela
Badorrey	Delicias	Valsaín
Fernández	Jazmín	León
Gómez	Carretas	Cerceda
González	Arenal	La Granja
López	Mayor	Peguerinos
Pérez	Carretas	Cerceda
Rodríguez	Yeserías	Cádiz
Rupérez	Ramblas	León
Santos	Mayor	Peguerinos
Valdivieso	Goya	Vigo

FIGURA 3.4. La relación *cliente*.

Continuemos con el ejemplo bancario. Se necesita una relación que describa la información sobre los clientes. El esquema de la relación es:

Esquema-cliente = (*nombre-cliente, calle-cliente, ciudad-cliente*)

En la Figura 3.4 se muestra un ejemplo de la relación *cliente* (*Esquema-cliente*). Obsérvese que se ha omitido el atributo *id-cliente*, porque se desea tener esquemas de relación más pequeños en este ejemplo. Se asume que el nombre de cliente identifica unívocamente un cliente; obviamente, esto no es cierto en el mundo real, pero las suposiciones hechas en estos ejemplos los hacen más sencillos de entender.

En una base de datos del mundo real, *id-cliente* (que podría ser el número de la seguridad social o un identificador generado por el banco) serviría para identificar unívocamente a los clientes.

También se necesita una relación que describa la asociación entre los clientes y las cuentas. El esquema de la relación que describe esta asociación es:

Esquema-impositor = (*nombre-cliente, número-cuenta*)

nombre cliente	*número cuenta*
Abril	C-102
Gómez	C-101
González	C-201
González	C-217
López	C-222
Rupérez	C-215
Santos	C-305

FIGURA 3.5. La relación *impositor*.

número-préstamo	*nombre-sucursal*	*importe*
P-11	Collado Mediano	900
P-14	Centro	1.500
P-15	Navacerrada	1.500
P-16	Navacerrada	1.300
P-17	Centro	1.000
P-23	Moralzarzal	2.000
P-93	Becerril	500

FIGURA 3.6. La relación *préstamo*.

En la Figura 3.5 se muestra un ejemplo de la relación *impositor* (*Esquema-impositor*).

Puede parecer que, para el presente ejemplo bancario, se podría tener sólo un esquema de relación, en vez de tener varios. Es decir, puede resultar más sencillo para el usuario pensar en términos de un esquema de relación, en lugar de en términos de varios esquemas.

Supóngase que sólo se utilizara una relación para el ejemplo, con el esquema (*nombre-sucursal, ciudad-sucursal, activos, nombre-cliente, calle-cliente, ciudad-cliente, número-cuenta, saldo*)

Obsérvese que si un cliente tiene varias cuentas hay que repetir su dirección una vez por cada cuenta. Es decir, hay que repetir varias veces parte de la información. Esta repetición supone un gasto inútil y se evita mediante el uso de varias relaciones, como en el ejemplo presente. Además, si una sucursal no tiene ninguna cuenta (por ejemplo, una sucursal recién creada que todavía no tiene clientes), no se puede construir una tupla completa en la relación única anterior, dado que no hay todavía ningún dato disponible referente a *cliente* ni a *cuenta*. Para representar las tuplas incompletas hay que utilizar valores *nulos* que indiquen que ese valor es desconocido o no existe. Por tanto, en el ejemplo presente, los valores de *nombre-cliente*, *calle-cliente*, etcétera, deben quedar nulos. Utilizando varias relaciones se puede representar la información de las sucursales de un banco sin clientes sin utilizar valores nulos. Sencillamente, se utiliza una tupla en *Esquema-sucursal* para representar la información de la sucursal y sólo crear tuplas en los otros esquemas cuando esté disponible la información adecuada.

Se incluyen dos relaciones más para describir los datos de los préstamos concedidos en las diferentes sucursales del banco:

Esquema-préstamo = (*número-préstamo, nombre-sucursal, importe*)

Esquema-prestatario = (*nombre-cliente, número-préstamo*)

Las relaciones de ejemplo *préstamo* (*Esquema-préstamo*) y *prestatario* (*Esquema-prestatario*) se muestran en las Figuras 3.6 y 3.7, respectivamente.

La entidad bancaria que se ha descrito se deriva del diagrama E-R mostrado en la Figura 3.8. Los esquemas de las relaciones se corresponden con el conjunto de tablas que se podrían generar utilizando el método esbozado en el Apartado 2.9. Obsérvese que las tablas para *cuenta-sucursal* y *préstamo-sucursal* se han combinado en las tablas de *cuenta* y *préstamo* respectivamente. Esta combinación es posible dado que las relaciones son de varios a uno desde *cuenta* y *préstamo*, respectivamente, a *sucursal* y, además, la participación de *cuenta* y *préstamo* en las relaciones correspondientes es total, como indican las líneas dobles en la figura. Finalmente, obsérvese que la relación *cliente* puede contener información sobre clientes que ni tengan cuenta ni un préstamo en el banco.

Cuando sea necesario, habrá que introducir más esquemas de relaciones para ilustrar casos concretos.

nombre cliente	*número préstamo*
Fernández	P-16
Gómez	P-93
Gómez	P-15
López	P-14
Pérez	P-17
Santos	P-11
Sotoca	P-23
Valdivieso	P-17

FIGURA 3.7. La relación *prestatario*.

LENGUAJES DE CONSULTA

Un **lenguaje de consulta** es un lenguaje en el que un usuario solicita información de la base de datos. Estos lenguajes suelen ser de un nivel superior que el de los lenguajes de programación habituales. Los lenguajes de consulta pueden clasificarse como procedimentales o no procedimentales. En los lenguajes procedimentales el usuario instruye al sistema para que lleve a cabo una serie de operaciones en la base de datos para calcular el resultado deseado. En los lenguajes no procedimentales el usuario describe la información deseada sin dar un procedimiento concreto para obtener esa información.

La mayor parte de los sistemas comerciales de bases de datos relacionales ofrecen un lenguaje de consulta que incluye elementos de los enfoques procedimental y no procedimental.

Aunque inicialmente sólo se estudiarán las consultas, un lenguaje de manipulación de datos completo no sólo incluye un lenguaje de consulta, sino también un lenguaje para la modificación de las bases de datos. Estos lenguajes incluyen órdenes para insertar y borrar tuplas, así como órdenes para modificar partes de las tuplas existentes.

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. Cada relación tiene una estructura similar a la presentada anteriormente. SQL permite el uso de valores nulos para indicar que el valor o bien es desconocido, no existe. Se fijan criterios que permiten al usuario especificar a qué atributos no se puede asignar valor nulo.

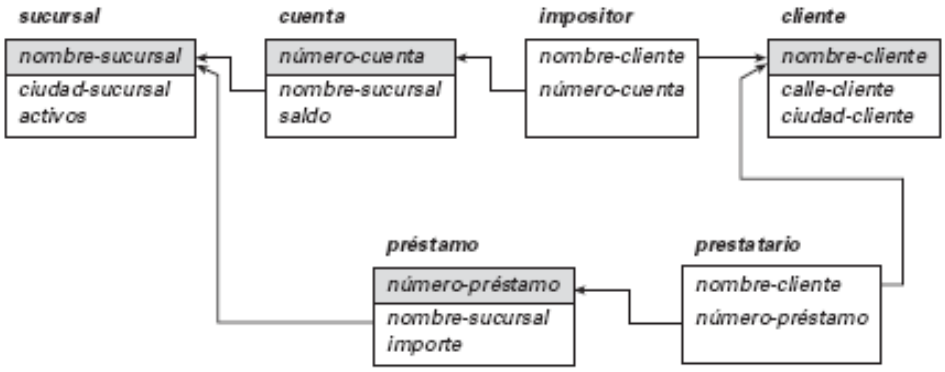

FIGURA 3.9. Diagrama de esquema para el banco.

La estructura básica de una expresión SQL consiste en tres cláusulas: **select**, **from** y **where**.

- La cláusula **select s**e usa para listar los atributos deseados del resultado de una consulta.
- La cláusula **from** lista las relaciones (tablas) que deben ser analizadas en la evaluación de la expresión.
- La cláusula **where** es un predicado que engloba a los atributos de las relaciones que aparecen en la cláusula **from**.

Una consulta típica en SQL tiene la forma

select $A_1, A_2, \ldots, A_n$

from $r_1, r_2, \ldots, r_m$

where P

Cada *Ai* representa un atributo, y cada *ri* una relación (tabla). *P* es un predicado.

Si se omite la cláusula **where**, el predicado P es **cierto** por lo tanto 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**.

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*

Sí 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.

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 $\wedge$, $\vee$ y $\neg$) 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**.

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.

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*, 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'

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*

Variables tupla

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. 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 de *sucursal* se refiere.

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 por ciento (%): El carácter % encaja con cualquier subcadena.
- Subrayado (_): El carácter _ encaja con cualquier carácter.

Los patrones son muy sensibles, esto es, los caracteres en 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 «||»), la extracción de subcadenas, el cálculo de la longitud de las cadenas, la conversión a mayúsculas y minúsculas, etc.

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 **desc** para orden descendente o **asc** 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* **desc**, *número-préstamo* **asc**

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.

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-cliente = cliente.nombre-cliente and
      ciudad-cliente = 'Madrid'
group by impositor.nombre-cliente
having count (distinct impositor.número-cuenta) >= 3
```

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.

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 sistemas 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 la 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).

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*, *númerocuenta*, *saldo*)
values ('Navacerrada', 'C-9732', 1200)

insert into *cuenta* (*número-cuenta*, *nombresucursal*, *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-sucursal = '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-401 es desconocido, no se puede determinar si es igual a «Navacerrada ».

Se puede prohibir la inserción de valores nulos utilizando el LDD de SQL

.

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 legal 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, como se vio anteriormente, 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 igual o muy poco inferior a 10.000 € recibiría un 11,3% de interés.

ANEXO:

Que es Xampp?

XAMPP es un acrónimo que significa "Apache, MySQL, PHP y Perl". Es un paquete de software gratuito y de código abierto que proporciona un entorno de desarrollo completo para la creación de sitios web dinámicos.

En términos sencillos, XAMPP es un programa que puedes instalar en tu computadora para convertirla en un servidor web local. Proporciona todo lo que necesitas para ejecutar aplicaciones web, como un servidor web (Apache), una base de datos (MySQL) y los lenguajes de programación PHP y Perl.

El propósito principal de XAMPP es facilitar la configuración y el uso de un servidor web en entornos de desarrollo. Permite a los desarrolladores crear y probar sitios web en su propia máquina sin necesidad de una conexión a Internet. Esto es útil para aprender a programar, desarrollar aplicaciones web, probar sitios web antes de publicarlos y experimentar con diferentes tecnologías web.

XAMPP es compatible con diferentes sistemas operativos, como Windows, macOS y Linux, lo que lo hace accesible para una amplia gama de usuarios. Además, incluye herramientas adicionales, como phpMyAdmin, que facilita la administración de la base de datos MySQL.

Es importante tener en cuenta que XAMPP está destinado principalmente a fines de desarrollo y no se recomienda su uso para alojar sitios web en producción en Internet, ya que no está diseñado para ser tan seguro o eficiente como los servidores web profesionales.

En resumen, XAMPP es un paquete de software que te permite convertir tu computadora en un servidor web local para desarrollar y probar sitios web en un entorno de desarrollo.

Pasos básicos para instalar XAMPP y utilizar únicamente Apache y MySQL:

Descarga XAMPP: Ve al sitio web oficial de XAMPP (https://www.apachefriends.org) y descarga la versión correspondiente a tu sistema operativo (Windows, macOS o Linux).

Ejecuta el instalador: Una vez que la descarga haya finalizado, ejecuta el archivo de instalación de XAMPP. Si estás en Windows, simplemente haz doble clic en el archivo descargado y sigue las instrucciones del instalador.

Selecciona los componentes: Durante el proceso de instalación, se te presentarán los componentes disponibles para instalar. Asegúrate de desmarcar aquellos que no deseas utilizar y mantén seleccionados únicamente Apache y MySQL.

Elige la carpeta de instalación: Se te pedirá que elijas la carpeta de instalación de XAMPP. Puedes dejar la ubicación predeterminada o seleccionar una carpeta diferente en tu disco duro.

Completa la instalación: Una vez que hayas seleccionado los componentes y la carpeta de instalación, haz clic en "Instalar" o "Siguiente" y espera a que se complete la instalación de XAMPP. Esto puede llevar algunos minutos.

Inicia los servicios de Apache y MySQL: Una vez instalado, inicia XAMPP. En Windows, puedes encontrar un acceso directo en el escritorio o en el menú de inicio. En macOS y Linux, puedes iniciar XAMPP desde el terminal o utilizando el administrador de aplicaciones.

Verifica el funcionamiento: Abre un navegador web y escribe "http://localhost" en la barra de direcciones. Si ves una página de inicio de XAMPP, significa que Apache se ha instalado correctamente. Para verificar MySQL, puedes abrir "http://localhost/phpmyadmin" en el navegador y asegurarte de que puedes acceder a la interfaz de administración de la base de datos.

Ahora tienes XAMPP instalado con Apache y MySQL. Puedes utilizar Apache para alojar y acceder a tus archivos web localmente, y MySQL para crear y administrar bases de datos.