SISTEMAS GESTORES DE BASES DE DATOS

SQL

TEMA I

- **INTRODUCCION**
- **LA INFORMACION Y SU ALMACENAMIENTO**

1.2.1. Sistemas de información

1.2.2. Estructura básica de almacenamiento: el archivo

1.2.3. Modos de acceso a los registros de un archivo

1.2.4. Organizaciones físicas de archivos

1.2.5. Criterios de selección de organizaciones físicas

1.2.6. Gestión de archivos en soportes

Notas

- **INTRODUCCION**

Este manual pretende servir como manual para el módulo profesional *Sistemas Gestores de Bases de Datos* del ciclo formativo superior de Administración de Sistemas Informáticos.

El posible lector puede ser un estudiante de Formación Profesional, o un interesado en iniciarse en bases de datos, con preferencia por el modelo relacional y el lenguaje SQL.

En el manual se introduce la teoría del modelo relacional, mediante un ejemplo continuo, enlazar todos los capítulos de cara a la realización de un proyecto final de programación.

Si bien hasta hace unos años se han estado implantando sistemas informáticos con aplicaciones basadas en archivos dependientes del software y el hardware, la tendencia actual persigue independizar la información de las aplicaciones y agrupar todos los datos en una única entidad llamada *base de datos*, de forma que distintos procesos, en muchos casos de aplicaciones, e incluso, sistemas diferentes, utilicen y compartan la misma información.

- **LA INFORMACION Y SU ALMACENAMIENTO**

1.2.1. Sistemas de información

Un sistema de información es un conjunto de actividades que administran la información relevante en una entidad, generalmente, una empresa. Se encarga de la distribución de los datos según unos determinados requerimientos, de la correcta compartición de la información entre sus usuarios y de su almacenamiento en

soportes adecuados basados en ordenadores y en los avances de las telecomunicaciones.

Los grandes volúmenes de información manejados por un sistema se agrupan en conjuntos más pequeños para poder ser nombrados y representados en las aplicaciones que los utilizan.

La unidad más pequeña que permite representar información es el *bit*, que admite dos valores: 0 y 1. Un grupo de 8 bits forman el *byte*. Cada dato al que puede hacerse referencia en un sistema de información se denomina *campo* y está formado por un grupo de bytes. Los datos que forman parte de una entidad común se agrupan en un *registro*, que está formado por un conjunto de campos. El campo o grupo de campos que identifican unívocamente a cada registro de un archivo se denomina *campo clave*. Todos los registros del mismo tipo forman un *archivo*. El conjunto de archivos que representa la información de un sistema forma una *base de datos*.

1.2.2. Estructura básica de almacenamiento: el archivo

Para almacenar la información, ofrecen una estructura de datos de alto nivel llamada *archivo* que distribuye los datos en dispositivos externos de almacenamiento.

Normalmente los archivos de datos serán almacenados en registros de tipo lógico. Estos registros lógicos no son más que estructuras de datos formadas por uno o más elementos llamados *campos* que constituyen una unidad para un determinado proceso.

Cuando se necesita utilizar un archivo en un programa, define el tipo de registro que lo formará y, posteriormente, lo utiliza según las operaciones que facilite sobre él el lenguaje de programación.

Nombre del archivo: CLIENTES

Nombre del registro: R−CLIEN

Campo clave: NIF

Formato del registro:

Campo Nombre Tipo de datos Longitud

1 NIF Alfanumérico 10

2 APELLIDOS Alfanumérico 20

3 NOMBRE Alfanumérico 15

4 NACIMIENTO Fecha 8

.....

Las características más importantes de los archivos son:

- Residen en soportes externos (DD), por lo que su existencia no está limitada al tiempo de ejecución del programa que lo crea, sino que permanece cuando éste termina.
- Los datos pueden transportarse de un ordenador a otro.
- Tienen capacidad de almacenamiento ilimitada ya que, aunque un soporte tiene capacidad limitada, un archivo puede distribuirse en varios soportes.

Los archivos pueden clasificarse atendiendo a la función que realizan, se pueden dividir de la siguiente forma:

- Permanentes. Sus registros varían poco en el tiempo. También se conocen con el nombre de *archivos maestros.*

    - *Constantes*. Su información permanece prácticamente invariada, utilizándose como archivos de consulta.
    - *De situación*. Reflejan el estado o situación actual de una empresa o entidad. Estos archivos se actualizan periódicamente para adaptarlos a una nueva situación. Son los menos permanentes.
    - *Históricos*. Se obtienen de los anteriores cuando éstos dejan de utilizarse, y sirven para hacer estudios futuros estadísticos o de consulta.

- De movimientos. En ellos se almacena temporalmente la información que se utiliza para actualizar los archivos de situación.
- De maniobra. Son archivos temporales creados durante la ejecución de un programa y borrados habitualmente al terminar el mismo. Por ejemplo, archivos intermedios utilizados en procesos de ordenación.

Las operaciones que pueden realizarse sobre un archivo son:

- Crear la estructura del archivo. CREATE, establece la estructura y posición del archivo en el dispositivo de almacenamiento.
- Abrir un archivo creado para poder utilizarlo. Cuando el archivo ya existe, debe abrirse para trabajar con él, bien sea para consultarlo o para actualizarlo. OPEN.
- Leer un registro de un archivo. Transfiere la información del registro actual al área de datos del programa que solicita la lectura. READ.
- Escribir un registro en un archivo. Graba en el soporte de almacenamiento el contenido de un registro con los datos especificados en el área de datos del programa. WRITE.
- Cerrar un archivo cuando ya no va a utilizarse. Esta operación es obligatoria antes de concluir un programa. Si un archivo queda abierto, se corre el riesgo de perder información. CLOSE actualiza la situación real del archivo y elimina de memoria la tabla mantenida por el sistema para agilizar las operaciones de acceso al archivo.
- Eliminar un archivo. Cuando un archivo deja de tener validez, es conveniente borrarlo del dispositivo de almacenamiento para no desperdiciar espacio. DELETE.
- Renombrar un archivo. RENAME permite cambiar el nombre a un archivo.
- Copiar un archivo. Consiste en duplicar la información de un archivo en otro.

- Editar un archivo. Permite modificar el contenido de un archivo. Se utiliza, sobre todo, en archivos de texto.
- Indexar un archivo. Es una operación de alto nivel que permite dar a un archivo organización indexada. El acceso a los registros del archivo se hará a través de un archivo índice que estará ordenado por el campo que se indexó (clave).

1.2.3. Modos de acceso a los registros de un archivo

Independientemente de cómo estén organizados los datos en un archivo, para los programas, es importante la forma en que puede accederse a los registros. Existen básicamente tres modos de acceso:

- **Acceso secuencial**. Las operaciones de lectura o de escritura se hacen sobre el registro físicamente contiguo al último que se utilizó. Este modo de acceso es consecuencia de que los primeros dispositivos de almacenamiento que eran soportes secuenciales (tarjetas perforadas, cintas perforadas y cintas magnéticas).

- **Acceso directo**. Los registros pueden leerse y escribirse directamente en la posición física que ocupan en el archivo (tambores y discos magnéticos).
- **Acceso por índice**. Consiste en crear un índice ordenado con las claves del archivo. Para acceder a los registros, se busca secuencialmente la clave en el índice, que lleva asociada la dirección real del registro en el archivo, el cual se lee o escribe directamente.

1.2.4. Organizaciones físicas de archivo

Existen cuatro tipos de organización física de una archivo, secuencial, relativa, indexada e invertida.

- Organización secuencial: Con este tipo de organización, los registros se almacenan uno detrás de otro, sin dejar huecos y en el orden en que se van grabando. El único modo permitido de acceso a los registros es el secuencial. Las ventajas de este método radican en que no desaprovechan espacio de almacenamiento y en la elevada eficiencia de recuperación de registros cuando el porcentaje de consultados es suficientemente grande. Entre sus inconvenientes:

  - Para acceder a un registro, hay que leer todos los que hay delante de él, que de media son la mitad de los registros del archivo.

  - No se puede insertar un registro en medio de otros dos, ya que sólo está permitido añadir registros al final del archivo, lo que supone que si el archivo está ordenado, hay que reordenarlo.
  - No se puede borrar un registro porque se generaría un hueco.
  - Para actualizar un archivo es necesario un archivo de movimientos, trasladar el maestro actualizado a uno nuevo y, posteriormente, cambiar el nombre al nuevo por el del maestro.

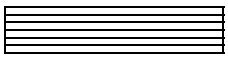
En la actualidad se utilizan poco los archivos con organización secuencial. Únicamente se aconseja esta organización para archivos históricos y constantes. Un ejemplo de archivo con organización secuencial puede ser el de abonados a una compañía telefónica, en el que se almacenan los datos que aparecen anualmente en las guías.

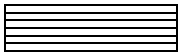
- Organización relativa: Un archivo con organización relativa es un conjunto de posiciones contiguas de memoria que tiene longitud fija. Cada posición se denomina *cubo*. El número de cubos que forman el archivo está predeterminado inicialmente. Los cubos están numerados de 1 a *n* (ó de 0 a n−1), siendo *n* el número de cubos del archivo. Cada cubo puede contener uno o varios registros. Por tanto, el número total de registros que puede contener el archivo es el número de cubos multiplicado por el número de registros que caben en cada cubo.

Es posible acceder a los registros en los modos secuencial y directo. Para el modo secuencial basta con utilizar una variable como contador de cubos e ir accediendo a los cubos cuya dirección coincida con la variable. Si los cubos tienen más de un registro, el acceso a los registros del mismo cubo se hace de forma secuencial.

Para acceder directamente, se debe conocer la dirección que ocupa el registro en el archivo. Se denomina dirección *lógica* al número de cubo en el que se encuentra un registro y dirección *física* a la que realmente ocupa el registro en el soporte donde se almacena el archivo.

La dirección del cubo en el que se encuentra un registro se calcula por su campo clave. El valor del campo clave se transforma en una de las direcciones lógicas válidas mediante un algoritmo de conversión. Esta técnica se denomina *hashing.* De esta forma, siempre que se desea acceder a los datos de un registro, antes hay que convertir el valor de su campo clave, aplicando el algoritmo correspondiente. Todos los algoritmos transforman el valor de la clave en un valor numérico y en algunos, el número resultante no está en el rango de direcciones posibles, por lo que hay que adaptarlo multiplicándolo por un factor de ajuste.





Existen multitud de algoritmos de conversión de claves como:

- Direccionamiento directo. Se utiliza cuando las claves tienen rangos de valores densos y cada valor tiene asociada una dirección en el archivo; por ejemplo, un archivo de habitaciones de un hotel con clave el número de habitación.

Dirección real = valor clave buscada − clave más pequeña + 1

- Conversión binario−octal. Consiste asignar a cada dígito un valor numérico que se corresponde con su orden alfabético. A la *A* se le asociaría 1, a la *B*, 2, etc. Una variante consiste en asignar el valor numérico ASCII correspondiente. Los valores de todos los dígitos que forman la clave se suman y el resultado se transforma en número octal. El número así obtenido se multiplica por el factor de ajuste.

- De restas sucesivas. Se utiliza para claves numéricas consecutivas que contienen muchos huecos de valores conocidos previamente; por ejemplo, el archivo de habitaciones de un hotel con un número fijo de habitaciones por planta y con el dígito del número de planta precediendo al número de habitación. El método consiste en restar a la clave la suma de todos los huecos anteriores. Así, en el caso de un hotel con 10 plantas y 30 habitaciones por planta sería de la siguiente forma:

Huecos: 1 a 100, 131 a 200, 231 a 300,..., 931 a 999

Habitación: 411

Huecos anteriores: 100 + 3 plantas * 70 huecos/planta = 310

Dirección asociada: 411−310 = 101

- Direccionamiento por *división:* Se divide el valor del campo clave por un número fijo, se toma el resto de la división entera como dirección lógica del registro y se le suma 1 si se desea adaptarlo al rango de direcciones 1,..., *n*. Para la elección del divisor existen varias opciones, entre las que destacan:

- Elegir el número de direcciones más uno. No es el mejor, porque si el número es divisible por números pequeños, generará demasiados sinónimos (se estudian más adelante).

- Elegir el número primo más cercano al número de direcciones físicas para minimizar el número de sinónimos. Si se elige el mayor de los primos menores que él, se corre el riesgo de dejar muchos huecos (número de direcciones − número primo). El concepto de hueco se estudia más adelante. Si se elige el menor de los primos mayores, hay que multiplicar por un factor de ajuste, y si el primo es bastante mayor que el número de direcciones, también se generarán muchos huecos.

- Una solución intermedia consiste en encontrar un número menor que el número de direcciones, pero que esté muy cercano, que sea primo o no tenga divisores menores de 10; por ejemplo:

Número de direcciones: 1000

Primo más cercano: 997

Clave a transformar: 5431

Resto de 5431 entre 997: 446

Dirección asociada: 446

- Direccionamiento por el *centro del cuadrado*. Consiste en eliminar del número obtenido al elevar la clave al cuadrado, las cifras extremas, es decir, las de la derecha e izquierda y tomar el mismo número de dígitos centrales como tenga el número de direcciones posibles como un valor decimal comprendido entre 0 y 1. A continuación, multiplicar este valor por el número de direcciones disponibles y despreciar los decimales resultantes.

- Direccionamiento por *plegamiento.* En este método, los dígitos exteriores en ambos extremos se desplazan hacia dentro de modo que se trasladan y queden dos números del orden de magnitud del número de direcciones. Se suman estos dos números y el resultado se multiplica por el factor de ajuste, despreciando los decimales.

La principal ventaja de esta organización es la rapidez con que se pueden recuperar de forma aleatoria. Gracias al acceso directo, se puede actualizar un archivo sin utilizar otro de movimientos. Las altas se realizan asignando al registro un cubo con algún registro vacío. Las modificaciones se hacen directamente sobre el registro elegido. Sus inconvenientes son los siguientes:

- Hay que hacer una estimación inicial del número de registros que podrá tener el archivo, corriendo el riesgo de desperdiciar espacio o llenarlo pronto y, por consiguiente, tener que trasladarlo a otro archivo de mayor capacidad.
- Se tiene que elegir un buen algoritmo de transformación de claves en direcciones.
- Todos los algoritmos, excepto el de direccionamiento directo, producen sinónimos. Dos registros son *sinónimos* si, teniendo claves diferentes, el algoritmo les asigna la misma dirección. En el ejemplo del algoritmo de direccionamiento por división, a la clave 4434 también le correspondería la dirección 446. Este problema puede solucionarse dando capacidad de más de un registro a cada cubo y guardando todos los sinónimos de una dirección en su cubo (lo que implica, seguramente, desaprovechamiento de espacio) o creando un archivo adicional con organización secuencial o relativa exclusivamente para los sinónimos. El algoritmo que menos sinónimos genera es el de división.
- Todos los algoritmos, excepto el de direccionamiento directo, producen huecos. Un *hueco* es un cubo que nunca será ocupado por ningún registro, ya que no existe ningún valor en el rango del campo clave que, al aplicar el algoritmo, genere la dirección de dicho cubo. En el ejemplo de direccionamiento por división los cubos 998, 999 y 1000 serán huecos.

- Organización indexada: La organización indexada nace para suplir las deficiencias de la secuencial (no hay acceso directo a los registros) y de la relativa (se accede directamente a los registros sólo por un campo). En muchos casos conviene recuperar la información accediendo por campos que no son clave. También puede ser necesario recuperar los registros en una secuencia distinta a la que ocupan físicamente.

Se denomina *archivo índice* a un archivo que contiene dos campos: uno con los valores del campo por el que se desea recuperar la información y otro con la dirección física (puntero) que ocupa el registro en el archivo de datos. El archivo índice se mantiene ordenado por el campo de datos, facilitando la búsqueda. Su función es análoga al índice de un libro.

- Un archivo de datos puede tener asociados tantos índices como sea necesario.
- Puede utilizarse una combinación de campos para formar el campo de un índice; por ejemplo, Apellidos + Nombre.

- Al actualizar el archivo de datos se debe reorganizar todos sus índices para mantener la coherencia.
- Los índices pueden ser varios niveles. En los índices de nivel mayor de uno se busca el primer elemento mayor o igual que el deseado. Este procedimiento facilita el acceso a los registros, sin embargo, complica los algoritmos de mantenimiento del índice.

Clave	Puntero
A	
B	
C	
D	
E	
F	
G	
H	
I	7
J	1
K	8
L	10
M	9
N	2
O	11
P	3
Q	5

Dirección	Código	Nombre
1	100	B
2	110	F
3	120	H
4	130	J
5	140	I
6	160	K
7	200	A
8	210	C
9	250	E
10	300	D
11	360	G
12	400	L
13	500	N
14	550	O
15	600	M
16	700	Q
17	999	P

- Organización invertida: Los sistemas de gestión de bases de datos utilizan accesos más complejos que los utilizados en sistemas de archivos. Por ello, establecen organizaciones de archivos de alto nivel basadas en las anteriores. Una organización de alto nivel es la invertida. Para un archivo de datos se crean dos índices:
  - *De nombre*. Contiene las claves por las que se desea acceder. Se almacenan en una columna los nombres de las claves de acceso y en la otra un puntero con la dirección de la clave en el índice.
  - *De valor*. Contiene todos los valores distintos de los campos clave y un puntero con un conjunto de valores del archivo de datos que cumplen la condición de la clave.

DATOS LECTORES

Dirección	Código	Nombre	Sexo	Tipo
1	111	Juan	V	Infantil

2	222	Rosa	M	Investigador
3	333	Pedro	V	Colaborador
4	444	Raúl	V	Infantil
5	555	Carlos	V	Nomal
6	666	Sofía	M	Trabajador
7	777	Ana	M	Infantil
8	888	Iván	V	Colaborador

ÍNDICE VALOR

Dirección	Valor−clave	Puntero
1		
2		
3		
4	V	1,3,4,5,6
5	M	2,6,7
6	Colaborador	3,8
7	Infantil	1,4,7
	Investigador	2
	Normal	5
	Trabajador	6
	1	
	3	

ÍNDICE NOMBRE

Nombre−Clave	Puntero
Sexo	
Tipo	

1.2.5. Criterios de selección de organizaciones físicas

Para decidir el tipo de organización que tendrá un archivo se deben considerar las operaciones que se van a realizar con más frecuencia sobre el mismo. Existen dos conceptos que miden el grado de utilización de los registros de un archivo y que pueden servir de punto de partida para decidir la organización que debe tener un archivo.

- *Indice de volatilidad.* Es el porcentaje de registros que se añaden o suprimen respecto al número medio total de registros del archivo en un período de tiempo fijo. Un archivo es *estático* cuando tiene un bajo porcentaje de adiciones o supresiones y es *volátil* en caso contrario.
- *Indice de actividad.* Es el porcentaje de registros utilizados para modificación o consulta respecto al número medio total de registros del archivo en un período de tiempo fijo. Puede emplearse para decidir si la organización de un archivo debe ser secuencial o relativa comparando los tiempos

empleados en acceder a los registros en ambos modos. Se utilizan dos factores:

A = NRV /NRT

B = LR / (TA * VLS)

Donde:

NRV = número medio de registros utilizados en tiempo fijo.

NRT = número medio total de registros del archivo.

LR = la longitud en bytes de registro.

TA = el tiempo medio de acceso directo a un registro en segundos.

VLS = velocidad de lectura secuencial en bytes por segundo.

Si A < B, deberá utilizarse la organización relativa y, en caso contrario, organización secuencial.

1.2.6. Gestión de archivos en soportes

La gestión de archivos es una de las tareas principales de los sistemas operativos. El trabajo es distinto según el tipo de soporte que se utilice. Existen dos tipos de soporte:

- *Secuenciales.* Los datos se graban a continuación de otros, de tal forma que el acceso a un dato se hace pasando sobre todos los datos que le preceden en el soporte. El ejemplo clásico es la cinta magnética.
- *Direccionables.* El espacio de almacenamiento se divide en espacios parciales direccionables individualmente, pudiendo acceder a un dato por la dirección en que está almacenado, sin que sea necesario pasar por los datos almacenados en direcciones físicas anteriores. Ejemplos pueden ser los disquetes, los discos duros, los CD−ROM, etc.

Actualmente, la información se almacena en soportes direccionables, dejando los secuenciales para realizar grandes copias de seguridad. Por ello, se explica cómo se distribuyen los archivos en discos.

El disco se divide en unidades mínimas de E/S llamadas *clusters o bloques*. A cada archivo se le asignan bloques según su organización y el método de asignación de bloques que utilice el sistema operativo. Existen tres formas teóricas de asignar bloques:

- La *asignación contigua* de bloques requiere que todos los de un archivo ocupen posiciones contiguas de disco para una eficiente utilización. Todos los bloques de un archivo están contiguos en el disco. El acceso secuencial es relativamente sencillo, porque basta con ir leyendo bloques en la secuencia en que están en el disco. El acceso directo tampoco es complicado; sabiendo la dirección del primer bloque del archivo, para acceder al *i* basta con sumar *i* al bloque inicial. No obstante, este método de asignación tiene el problema de la predeterminación de los bloques que ocupará un archivo. También es posible que en el disco haya suficiente número de bloques para almacenar el archivo, pero no ocupen posiciones contiguas. Este problema se conoce con el nombre de *fragmentación externa.*

- La *asignación enlazada* fragmenta el archivo en bloques que pueden estar distribuidos aleatoriamente por el disco. Para saber la secuencia, el sistema operativo guarda la dirección del primer bloque, y cada bloque almacena en sus últimos bytes la dirección del siguiente bloque en secuencia, o, dicho de otro modo, los últimos bytes son un *puntero* al siguiente bloque. La dirección del último bloque es un

valor prefijado de antemano (nil). El acceso secuencial es muy sencillo, pero el directo es imposible.

- La *asignación indexada* suple en gran medida los problemas de las anteriores. Consiste en reunir todos los punteros en un bloque llamado *índice.* El *bloque índice* es una tabla de entradas, donde la entrada i−ésima contiene la dirección del bloque i−ésimo del archivo. El acceso secuencial se realiza recorriendo las entradas del bloque índice y el acceso directo a un bloque *b* se hace yendo directamente a la dirección que indica la entrada *b* del bloque índice. También facilita el acceso por índices.

Compactación

Se conoce como *compactación* la resolución del problema de la fragmentación externa cuando el método de bloques de disco es la contigua. Este proceso consiste en reubicar todos los archivos, de forma que los bloques libres dispersos en el disco formen un único hueco.

La *fragmentación interna* es un problema inherente a la distribución del espacio en bloques y radica en el espacio desaprovechado por los bytes libres que quedan en el último bloque de cada archivo.

Existen unos programas llamados *compresores* que, además de utilizar técnicas que reducen el número de bytes que ocupa un archivo, unen un número indeterminado de archivos en uno sólo, reduciendo la fragmentación interna a un bloque como máximo. Por ejemplo, 100 archivos de 100 bytes de tamaño distribuidos en bloques de 1.024 bytes desaprovecharían 92.400 bytes, mientras que comprimidos formarían un archivo de 10.000 bytes, que ocuparían 9 bloques completos, y el décimo desaprovecharía 784 bytes.

CUESTIONES:

- Explicar en qué consisten las fragmentaciones interna y externa.
- Calcular la dirección que correspondería a la clave 5555 en un archivo con organización relativa de 2.500 cubos utilizando los algoritmos de división, plegamiento y centro del cuadrado.
- Razonar si sería más conveniente la organización secuencial o directa para un archivo en el que mensualmente se utilizan 60.000 veces alguno de los 20.000 registros del archivo. Tener en cuenta que el registro ocupa 500 bytes, el tiempo de acceso medio es de 10 milisegundos y la velocidad de lectura secuencial es de 16 Kbytes por segundo.
- Completar la siguiente tabla marcando con una X las casillas en las que el modo de acceso y la organización sean compatibles.

	ACCESO SECUENCIAL	~~ACCESO DIRECTO~~	ACCESO POR ÍNDICE
Organización secuencial			
Organización relativa			
Organización indexada			

- Escribir un algoritmo en pseudocódigo que implemente la operación de copia de un archivo origen en otro destino utilizando operaciones básicas.
- Con el siguiente ejemplo de archivo de datos, dibujar cómo quedarían los índices para una organización invertida con claves de acceso en los campos Provincia e Hijos.

NIF	~~APELLIDOS~~	~~DIRECCIÓN~~	~~PROVINCIA~~	~~HIJOS~~
11111111A	BBB CCC	C/ pisa, 1	Albacete	2

22222222B	DDD EEE	C/ roma, 1	Madrid	3
33333333C	EEE EEE	C/parís, 1	Murcia	0
44444444D	AAA BBB	C/ Madrid, 3	Toledo	1
55555555E	BBB DDD	C/ Mónaco,2	Albacete	1
66666666F	JJJ AAA	C/Pisa, 9	Toledo	0
77777777G	AAA JJJ	C/ Mónaco, 11	Albacete	2
88888888H	RRR SSS	C/ París, 21	Madrid	1
99999999I	TTT BBB	C/ Madrid, 10	Toledo	3

- Con el archivo de datos anterior, crear un índice de dos niveles con el primer nivel dividido en cubos con capacidad para tres claves. El campo clave será apellidos.

TEMA II

- **SISTEMAS GESTORES DE BASES DE DATOS**
- **Introducción**
- **Arquitectura de una base de datos**
- **Sistemas gestores de bases de datos**
- **Componentes**
- **Modelos de sistemas gestores de bases de datos**
- **EL MODELO ENTIDAD−RELACION**
- **Principales elementos del modelo**
- **Grado y cardinalidad**
- **Factores que determinan la calidad de un esquema**
- **FUNDAMENTOS DEL MODELO RELACIONAL**
- **Arquitectura**
- **Reglas de integridad**
- **Paso del modelo entidad−relación al modelo relacional**

Notas

- **SISTEMAS GESTORES DE BASE DE DATOS**

2.1.1. Introducción

Una base de datos es un conjunto de datos almacenados de forma organizada y estructurada en un soporte de información que es manejado por un ordenador. La información se guarda en archivos independientes integrados en la base, y puede ser compartida por distintos usuarios que la utilicen para fines diferentes en instantes de tiempo que pueden coincidir.

Sus componentes más importantes son el hardware, los datos, los usuarios y los procedimientos encargados del uso correcto de la información. En estos sistemas, los programadores y usuarios no tienen que saber cómo está distribuida y organizada la información. De esto se encarga un conjunto de procedimientos que forma parte del sistema gestor de la base de datos.

2.1.2. Arquitectura de una base de datos

Uno de los objetivos de un sistema de base de datos es proporcionar una visión lo más abstracta posible de la información, es decir, ocultar detalles referentes a la forma en que los datos están organizados y almacenados.

La arquitectura más estándar y, por tanto, la más utilizada, es la que hace una división en niveles de la base de datos. Se consideran tres niveles según la perspectiva desde la que sea vista la información. El nivel interno es el nivel más bajo de abstracción, donde se describe la información en función del sistema en que se implantará la base de datos. Por encima, se encuentra el conceptual, que representa a alto nivel toda la información de la base de datos independientemente de la máquina en que vaya a utilizarse. El más abstracto de los niveles es el externo, que gestiona la información desde el punto de vista individual de cada usuario, grupo de usuarios, programador o grupo de programadores.

- Nivel interno. Se especifican los archivos que contienen la información, su organización, la forma de acceder a los registros, el tipo y longitud del registro, los campos que lo componen, los campos clave, etc.
- Nivel conceptual. Se definen todos los datos que intervendrán en el sistema. Se obtiene a partir de los requerimientos de los usuarios potenciales del sistema de base de datos a implantar, sin importar la forma ni el lugar en el que se almacenarán y recuperarán los datos. Contiene los datos elementales (campos), los datos compuestos (registros), las relaciones existentes entre campos elementales y compuestos, las reglas que rigen el funcionamiento de la empresa, etc.
- Nivel externo. Es el conjunto de percepciones individuales de la base de datos. Cada visión individual se denomina *subesquema* o *vista*. Un subesquema podrá ser compartido por varios usuarios, y cada usuario tendrá la posibilidad de acceder a distintos subesquemas. Al crear un subesquema, es posible mezclar campos de distintos registros, omitir campos, cambiar el orden de los campos, añadir campos que puedan ser calculados a partir de los descritos en el esquema conceptual, etc.

Para una base de datos específica, hay un único esquema interno y conceptual, pero puede haber varios esquemas externos, cada uno definido para uno o varios usuarios.

ESQUEMA EXTERNO

Subesquema 1:

JUGADORES CON UNA ALTURA DETERMINADA: JU_ARB, JU_ALT, JU_EQ

Subesquema 2:

PARTIDOS GANADOS POR UN EQUIPO : EQ_LOCAL, EQ_VISIT, RESULTADO

Subesquema 3:

EQUIPOS CON AL MENOS TRES JUGADORES DE 2,05: EQ_NOM, NÚMERO

ESQUEMA CONCEPTUAL

EQUIPOS : EQ_ARB, EQ_NOM, EQ_DEL, ...

JUGADORES : JU_ARB, JU_NOM, JU_ALT, ...

PARTIDOS : EQ_LOCAL, EQ_VISIT, FECHA, ...

Reglas :

- Un jugador sólo pertence a un equipo.
- Los equipos no juegan más de un partido diario.
- .....

ESQUEMA INTERNO

ARCHIV ORGANIZ. CLAVE LONG.REG. CAMPOS TIPO DATOS

Equipos Indexada EQ_ARB 70 EQ_ARB X(4)

EQ_NOM X(15)

EQ_DEL X(20)

.....

Jugadores Indexada JU_ARB 100 JU_ARB X(4)

JU_NOM X(20)

JU_ALT. 9,99

.....

Partidos Indexada EQ_LICAL+FECHA 50 EQ_LOCAL X(4)
EQ_VISIT. X(4)

FECHA DD/MM/AA .....

2.1.3. Sistemas gestores de bases de datos

Un sistema gestor de base de datos (SGBD) es un conjunto de programas que permiten la administración y gestión de la información de una base de datos. Una de sus funciones es proporcionar diferentes niveles de abstracción de la información, dependiendo del tipo de usuario que la maneja. Para la mayoría, se ocultan los detalles de la forma y el lugar en que están almacenados los datos, así como los procedimientos de recuperación y actualización de la información.

También se encarga de conectar e implementar los distintos niveles de la arquitectura de la base de datos. El *sistema de control de base de datos*, es el encargado de transformar los datos requeridos por los programas de aplicación en registros físicos a leer, hacer la petición al sistema operativo y, cuando la información está disponible, transferible al área de trabajo del programa que la solicitó.

El *administrador del sistema* organiza y controla los recursos del sistema. Sus principales funciones son:

- Definir el esquema conceptual con el lenguaje de definición del SGBD.
- Controlar el acceso a la base de datos, concediendo permisos a los usuarios.
- Definir estrategias de recuperación frente a posibles fallos.

Los objetivos que debe cumplir una base de datos para ser lo más rápida, eficaz y polivalente son los siguientes:

- Debe independizar los datos de las aplicaciones que los utilizan. Es lo que se conoce como *independencia física* (se puede modificar el esquema físico sin que afecte a los superiores) e *independencia lógica* (si se modifica el esquema conceptual no es necesario modificar los programas de aplicación).
- Conseguir que los datos repetidos innecesariamente en la base sean los mínimos posibles (redundancia mínima de información).
- Suministrar mecanismos de seguimiento de las operaciones realizadas en la base de datos. Esto se consigue mediante procesos espías que mantienen archivos en los que se almacena la fecha y hora de conexión de los usuarios, las operaciones realizadas en cada sesión, los datos modificados, etc.
- Proporcionar versatilidad en las posibilidades de búsqueda de información facilitando al usuario varios criterios.
- Asegurar la protección de los datos contra accesos no autorizados o malintencionados de los usuarios.
- Controlar la integridad de la información en la base de datos. Para ello, debe dar respuesta a posibles fallos de hardware, defectos en el código de los programas de aplicación, actualizaciones incompletas, inserción de datos incorrectos o no válidos, etc.
- Proporcionar mecanismos para realizar copias de seguridad de la información.

- Conseguir un tiempo de respuesta suficientemente pequeño para evitar que el usuario se desespere. El *tiempo respuesta* se define como el tiempo que transcurre desde que el usuario termina de realizar una petición al sistema hasta que empieza a recibir repuesta.

- Solucionar los problemas planteados por la concurrencia. Actualmente es posible que una base de datos esté compartida por varios usuarios situados en distintos terminales. Por ello, puede darse el caso de que dos o más usuarios distintos intenten actualizar de forma concurrente (en el mismo instante de tiempo) un registro determinado. Fundamentalmente, estos problemas son la actualización incorrecta y el bloqueo mutuo.

CAJAS DE SUPERMERCADO		
Instantes de tiempo	Cliente caja 1	Cliente caja 2
T1		
T2	El cliente adquiere 5 unidades del producto X. Para actualizar, se lee el stock y quedan 100.	El cliente adquiere 10 unidades del producto X. Para actualizar, se lee el stock y quedan 100.
T3	Se hacen las operaciones de cálculo.	Se hacen las operaciones de cálculo.
T4	Se anota que quedan 95 (100 -5) unidades,	Se anota que quedan 19 (100 -10) unidades cuando realmente quedan 85.

El primer problema se soluciona mediante una técnica llamada *cierre*, que consiste en poner un semáforo que se encuentre cerrado cuando el registro está siendo actualizado y abierto en otro caso.

CAJAS DE UN SUPERMERCADO		
Instantes de tiempo	Cliente en caja 1	Cliente en caja 2
T1	El cliente adquiere 5 unidades del producto X. Para actualizar,	El cliente adquiere 10 unidades del producto X. Para actualizar,
T2	se cierra el semáforo del registro y se lee el stock Quedan 100.	como el semáforo está cerrado, se pasa a la cola de espera.
T3	Se hacen las operaciones de	El proceso sale de la cola, se
T4	cálculo.	cierra el semáforo y se lee el

T5	Se anota que quedan 95 unidades. Se abre el semáforo.	registro. Quedan 95 unidades.
T6		Se hacen las operaciones de cálculo. Se anota que quedan 85 unidades y se abre el semáforo.

Las transacciones que se encuentran con el semáforo cerrado se guardan en una cola de espera y se procesan cuando el semáforo cambia de estado.

El segundo problema es consecuencia de la solución al primero. Se produce *bloqueo mutuo* cuando dos o más transacciones se encuentran en una espera circular indefinida. Para resolver el problema se puede impedir que suceda, algo difícil que afectaría al rendimiento del SGBD. También puede detectarse y dar marcha atrás para corregirlo.

CAJAS DE UN SUPERMERCADO		
Instantes de tiempo	~~Cliente en caja 1~~	~~Cliente en caja 2~~
T1	El proceso accede al archivo ARTICULOS cerrando su semáforo.	
T2		El proceso accede al archivo CLIENTES y cierra su semáforo.
T3	Se realizan las operaciones de cálculo. Se necesita acceder al archivo CLIENTES, pero como el semáforo está cerrado se pasa a la cola de espera.	Se necesita acceder al archivo ARTICULOS, pero como el semáforo está cerrado, se pasa a la cola de espera.

2.1.4. Componentes

El SGBD está dividido en módulos que llevan a cabo sus funciones asociadas. Se compone del núcleo, lenguaje, utilidades y diccionario de datos.

- **Núcleo:** Es el conjunto de programas que coordinan y controlan el funcionamiento del SGBD. Son programas transparentes al usuario.
    - Controlan la integridad y seguridad.
    - Implementan las funciones de comunicación entre niveles.
    - Facilitan la independencia de los datos.
    - Gestionan el diccionario de datos.
    - Proporcionan el soporte necesario para los programas de utilidad y los lenguajes.
- **Lenguajes:** El SGBD proporciona lenguajes que permiten la definición y el manejo de los datos de la base. Cada SGBD tiene una estructura y organización particular, pero todos tienen la característica común de ofrecer al administrador y a los usuarios dos lenguajes:
    - *El lenguaje de descripción de datos* (DDL) se utiliza para definir el esquema conceptual y los distintos subesquemas externos de la base de datos. Se ofrece establecer parte de la seguridad de la base de datos, permitiendo asignar derechos sobre las operaciones que pueden realizar usuarios.

- *El lenguaje de manipulación de datos* (DML) es el encargado de gestionar la información de la base de datos. Permite añadir, eliminar y modificar registros, y recuperar información de forma estructurada de la base de datos.

Los SGBD son capaces de procesar peticiones del DML que se han formulado desde programas escritos en otros lenguajes de programación.

- **Utilidades:** Son aplicaciones que facilitan el trabajo a los usuarios y programadores. Tiene la característica común de tener un interfaz fácil de entender. Se basan en menús que guían al usuario para conseguir el objetivo final.

    - *Asistentes*
    - *Generador de menús.* Diseña el interfaz de usuario de una aplicación.
    - *Generador de informes.* Presentan datos en pantalla o impresora con un formato predefinido o fácil de definir sin conocer lenguajes de base de datos ni de programación.
    - *Generador de formularios.* Genera pantalla de diálogos que presentan ítems y permiten la introducción de información; bien por teclado, bien por botones.

- **Diccionario de datos:** Es un almacén integrado en el que se almacena toda la información referente a la descripción, gestión e implantación de la base de datos. También se conoce como *catálogo del sistema.* Contiene la descripción de los esquemas interno, conceptual y externo, tablas de usuarios con sus permisos respectivos, los programas de aplicación que utilizan, las operaciones que realizan dichos usuarios en el sistema, otros recursos implicados en el mismo, etc.

Los diccionarios de datos están estructurados entres capas o niveles. Si la complejidad del sistema de base de datos lo requiere, las capas pueden subdividirse recursivamente. Las tres capas son:

- La capa de mayor nivel de abstracción se llama *global*, donde se reproduce la información común a todos los usuarios, incluido el administrador de la base de datos.
- A continuación, existe una capa *intermedia* que organiza la relación entre las capas globales y local. Puede existir varias capas y, normalmente, se representan mediante *vistas*, que son percepciones individuales que tienen los usuarios de la base de datos.
- Finalmente, al más bajo nivel de abstracción, se encuentra la capa *local*, en la que se representan los datos como grupos de información específica.

2.1.5. Modelos de sistemas gestores de bases de datos

- **Modelo relacional:** Los tipos de registro conceptuales se denominan *relaciones* y se representan mediante tablas bidimensionales en las que las filas son las ocurrencias de registros y las columnas, los campos correspondientes. El nivel externo se describe mediante vistas y la base de datos se almacena en archivos indexados.

ACTOR

NIF	NOMBRE	TELEFONO	.....
1111111A	Juan	123456	
5555555E	Pedro	232323	
3333333C	Rosa	344444	

PELÍCULA

CÓDIGO	TITULO	DÍAS	.....
1111	Aladdin	180	
3333	Pocahontas	210	
8888	El rey león	99	

PROTAGONISTA

NIF	CÓDIGO	PAPEL	....
111111ª	1111	Genio	
333333E	1111	Pobre	
333333E	3333	India	
333333E	8888	Reina	

- **Modelo orientado a objetos:** Los sistemas de gestión de bases de datos orientados a objetos (SGBDOO) son polivalentes, ya que facilitan las funciones de los SGBD, algunos de los lenguajes de programación y otras propias del desarrollo de sistemas orientados a objetos. Los SGBDOO están siendo objeto de estudios, y mediante su evolución se espera que desplacen a los sistemas relacionales. El principal obstáculo con el que se han encontrado es el aumento de complejidad del modelo, así como el hecho de proporcionar un mayor número de funciones.

    - *Objetos e identidad.* Cada entidad del mundo real es modelada como un objeto. Cada objeto tiene un identificador único, y está asociado a un *estado* que se representa por los valores de sus atributos y un *comportamiento* que se define por procedimientos llamados *métodos* que actúan sobre él.
    - *Objetos complejos.* Los valores de los atributos de un objeto pueden ser objetos.
    - *Encapsulamiento.* Cada objeto tiene definidos en su interior los métodos y la interfaz para tener acceso y capacidad de manipulación de dicho objeto.

    - *Clases.* Todos los objetos definidos por los mismos atributos y métodos forman una clase. Cada objeto que pertenece a una clase se dice *instancia* de la clase.

    - *Herencia.* Una clase (subclase) se puede definir como una especialización de otras clases (superclases) existentes, por lo que heredará sus atributos y métodos.

    - *Sobrecarga.* Una operación puede tener asociados distintos métodos. El sistema decide qué método realiza la operación. Por ejemplo, un objeto puede incluir la operación RESTA (que toma dos parámetros, hace su diferencia y retorna el resultado) y tener varios métodos para implementar la operación (uno para parámetros enteros, otro para restar fechas, otro para restar cadenas, etc.).

La arquitectura de los SGBDOO está basada en un enfoque cliente−servidor donde el servidor soporta las funciones del SGBD. El modelo de datos se apoya en los conceptos de clase, objeto y función. No hay distinción entre atributos y métodos, ya que ambos se tratan como funciones. Las funciones heredadas pueden ser redefinidas, de modo que dos clases pueden tener una función con el mismo nombre, pero con definiciones diferentes. Los dos elementos principales de la arquitectura son, por un lado, el administrador de objetos que implementa el modelo orientado a objetos y facilita el soporte para definir esquemas y realizar consultas; y, por otro, el administrador de almacenamiento externo que actualmente se implementa sobre un relacional.

DOCUMENTO PERSONA

ATRIBUTOS	Código:9(6) Titulo : x(15) Especialidad : x(15) Departamento: x(15)
METODOS	

ATRIBUTOS	**DNI : x(10)** **Nombre: x(20)** **Fecha_nacimi: fecha**
METODOS	**@Edad: Fecha**

ATRIBUTOS	**Nombre: x(15)** **Tema: DOCUMENTO** **Alumno: INVESTIGADOR** **Tutor: PERSONA** **Fecha_inicio: Fecha** **Fecha_final : Fecha**
METODOS	**@Duración: 9(3)**

ATRIBUTOS	**Datos: PERSONA** **NOTA 1,** **NOTA 2,** **NOTA3 : 99V99**
METODOS	**@NOTA_FINAL: 99V99**

CUESTIONES:

- **Enumerar las funciones de un SGBD.**
- **Ejemplificar el problema de la concurrencia sin utilizar el empleado en la descripción de dicho problema.**
- **Explicar la diferencia entre DML y DDL.**
- **Representar gráficamente como quedaría la información en los modelos lógicos de datos estudiados para una base de datos con los siguientes tipos de registros:**

SOCIO: NIF, Nombre, Dirección

LIBRO: ISBN, Título, Autor

PRÉSTAMO: NIF_socio, ISBN_libro, Fecha_préstamos y Fecha_devolución.

- **EL MODELO ENTIDAD−RELACION**

El modelo conceptual de datos más ampliamente conocido es el de entidades y relaciones. Permite realizar el diseño conceptual de una base de datos. Como se ha apuntado, este paso, en la fase de diseño, es previo a la elección del modelo lógico de datos y, por supuesto, al sistema gestor de base de datos.

El modelo entidad−relación es el que incorporan la mayoría de las herramientas de software para el diseño de sistemas. Es una representación gráfica y lingüística de los objetos que forman parte del mundo real. Describe los datos que son importantes en un entorno determinado. Proporciona una visión abstracta de la realidad, sin hacer alusión a formas de almacenamiento, tiempos de ejecución, sistemas operativos, sistemas gestores de bases de datos, etc.

Probablemente, su éxito es fruto del equilibrio conseguido en las cualidades deseables en un modelo de representación de datos: expresividad, simplicidad, minimalidad, formalidad y riqueza en su representación. Es expresivo porque ofrece varios mecanismos de abstracción de datos. Es rico en la descripción de conceptos, lo que pone de manifiesto su simplicidad. No obstante, determinados conceptos que complican el modelo, facilitan enormemente el paso al modelo relacional, jerárquico o de redes, lo que compensa el esfuerzo por entender y acostumbrarse a utilizar dichos conceptos. El modelo es mínimo, ya que ninguno de los elementos que intervienen pueden ser sustituidos por la combinación de otros. Está definido formalmente −Peter Chen lo hizo a finales de los setenta−. Es fácilmente legible y los diagramas son completos. La legibilidad de un diagrama entidad−relación se ve perjudicada cuando se pretenden incluir todos los elementos del modelo en un único diagrama. Por ello, es conveniente realizar varios esquemas detallados diferentes.

2.2.1. Principales elementos del modelo

- **Entidad.** Una entidad es una clase de objetos del mundo real.

Suelen ser los sustantivos empleados al describir las actividades de una empresa o institución: PERSONA, COCHE, ARTÍCULO, PROYECTO, ALUMNO etc. Cada objeto que pertenece a la clase se denomina *ocurrencia de la entidad*. Por ejemplo, los datos referentes al empleado Sánchez. Se distinguen dos clases de entidad:

- Fuertes. Son independientes no necesitan la existencia de otras entidades. Son la mayoría de ellas. Por ejemplo, EMPLEADOS.
- Débiles. Su existencia está ligada a otra entidad. Por ejemplo, HIJO_DE_EMPLEADO.

- **Relación.** Una relación es una asociación entre dos o más entidades.

Suelen identificarse por verbos que unen las entidades en la descripción lingüística de los datos: Por ejemplo, VENDE para EMPLEADO y COCHE, IMPARTE para PROFESOR Y ASIGNATURA etc.

Una relación es binaria cuando establece correspondencia entre dos entidades. Son las más comunes. Se denomina *anillo* o relación *recursiva* a la relación binaria que conecta a la misma clase de objetos; Por ejemplo, EQUPO SE_ENFRENTA_A_EQUIPO, EMPLEADO SUPERVIS_A_EMPLEADO, etc.

- **Atributo.** Un atributo es una entidad mínima de información que expresa propiedades de las entidades y relaciones.

Por ejemplo : Nombre, Edad, Fecha_nacimiento, Color, Modelo, pvp, etc. Se corresponde con el término campo utilizado en el ámbito de los sistemas de gestión de archivos.

Existen atributos que expresan propiedades de una entidad y atributos que lo hacen de una relación entre entidades. Por ejemplo, para la relación ACTOR TRABAJA_EN PELÍCULA, Nombre_actor, Edad y Teléfono pueden ser atributos de la entidad ACTOR; Título, Director y Fecha_de_estreno pueden ser atributos de la entidad PELÍCULA y Papel, Honorarios y Tiempo_en_escena podrán ser atributos de la relación TRABAJA_EN.

Se denominan ***ocurrencias de atributo*** a los valores válidos que pueden tomar los atributos; por ejemplo, para Color, rojo; para Teléfono, 6987877, etc.

Se dice que un ***atributo es compuesto*** si puede dividirse en atributos independientes. Por ejemplo, Fecha_de_nacimiento puede dividrse en dia, mes y año; Dirección puede dividirse en Calle, Número y Código_postal , etc.

Un atributo se dice que es ***obligatorio*** si todas las ocurrencias de atributo son valores no nulos y no vacíos, mientras que es ***opcional*** si puede estar vacío o contener el valor nulo.

Existe un caso especial llamado ***multivalor***, que significa que el atributo puede tener un conjunto de valores. Por ejemplo, el atributo Titulación para la entidad EMPLEADO es multivalor porque un empleado puede no tener titulación, tener alguna, una, dos etc. Las ocurrencias de atributos multivalores se encierran entre llaves y se separan con comas. Una ocurrencia de EMPLEADO con atributos NIF, Nombre, Titulación, edad, etc. Podría se : 1234567A, Pérez, {Bachiller, FPII ADM, Graduado social}, 30, etc.

- **Clave.** Se denomina clave a un grupo de atributos que determinan unívocamente todas las ocurrencias de una entidad (clave de la entidad) o de una relación (clave de la relación).

De todas las claves posibles para una entidad o relación, la elegida al realizar el diseño se denomina clave ***primaria o principal*** y, el resto, claves ***alternativas o secundarias.***

Una clave se dice *simple* si el número de atributos que la forman es uno, y *compuesta*, si tienen más de un atributo.

Un grupo de atributos es clave foránea de una entidad si el conjunto de atributos que la forman es clave primaria en otra entidad.

Por ejemplo, para la entidad LIBRO con atributos ISBN, Título, NIF_del_autor, y Número_de_ejemplares, las claves candidatas son ISBN Y Título (considerando que no existen libros con títulos idénticos), ISBN es clave primaria simple y Título es clave alternativa. NFI_del_autor es clave foránea porque es clave primaria en la

entidad Autor.

Cuando existen varias claves candidatas, puede resultar conflictivo elegir una de ellas como primaria. La decisión afectará a la rapidez de acceso físico a las ocurrencias de la entidad.

Para facilitar esta tarea pueden tenerse en cuenta los siguientes criterios:

- Elegir la clave que sirva para acceder directamente a las ocurrencias en el mayor número de operaciones.
- Es preferible escoger una clave simple antes que una compuesta.
- Se debe evitar que las claves foráneas formen parte de la clave primaria.

El concepto de clave puede ayudar a diferenciar entidades fuertes y débiles. Una entidad con claves candidatas es fuerte, mientras que si sólo tiene claves foráneas es débil.

2.2.2. Grado y cardinalidad

El *grado* de una relación refleja la participación de cada una de las entidades afectadas.

- *Uno a uno*, y se representa 1:1, si a cada ocurrencia de A le corresponde como máximo una ocurrencia de B, y viceversa
- *Uno a muchos*, y se representa 1:M, si a cada ocurrencia de A le pueden corresponder varias de B, pero a cada ocurrencia de B sólo le corresponde una de A como máximo. Si la asociación se entendiera de B con A, la relación sería M:1.
- *Muchos a muchos* y se representa M:M, si a cada ocurrencia de A le pueden corresponder varias de B, y viceversa.

La *cardinalidad* mide la obligatoriedad de correspondencia entre las ocurrencias de dos entidades en una relación. Se dice que una entidad A tiene un tipo de participación **obligatoria** en una ralación R con otra entidad B, si a cada ocurrencia de A le corresponde al menos una de B. En cambio, A tiene participación **opcional**, si pueden existir ocurrencias de A que no tengan correspondencia en B. Según esto, las relaciones binarias también se clasifican en:

- Obligatoria−Obligatoria. Todas las ocurrencias de cada entidad tienen correspondencia con como mínimo una ocurrencia de la otra. Por ejemplo, ALUMNO CURSA ASIGNATURA, todos los alumnos cursan al menos una asignatura, y cada asignatura es cursada por al menos un alumno.
- Obligatoria−Opcional. Cada ocurrencia de la primera entidad tiene asociada al menos una ocurrencia de la segunda; sin embargo, puede haber ocurrencias de la segunda entidad que no tengan asociadas ninguna en la primera entidad. Por ejemplo, en la relación CLIENTE OCUPA HABITACIÓN, suponiendo que sólo interesan los clientes alojados en el hotel, todos los clientes ocupan una habitación, pero puede haber habitaciones vacías y, por tanto, no ocupadas por ningún cliente. Si la relación hubiera sido enunciada a la inversa. HABITACIÓN ES−OCUPADA−POR CLIENTE, la relación sería opcional−obligatoria.
- Opcional−Opcional. Alguna ocurrencia de ambas entidades puede no tener correspondencia con ninguna ocurrencia de la otra entidad. Por ejemplo, en la relación LECTOR SACA−PRESTADO LIBRO, un lector socio de una biblioteca puede no haber sacado nunca un libro prestado y también es posible que ciertos libros nunca hayan sido prestados a lectores.

Diagramas entidad−relación

ENTIDADES

Fuerte y Obligatoria Débil Opcional

RELACIONES

1 1

1 M

M M

ATRIBUTOS

a1 a11

a2 a12

an a1n

Simple Compuesto

CLAVES

a1 a1

a2

an an an

a1 primaria y simple a1+ a2 primaria y compuesta ai foránea de E1 en E2 a1 ai

2.2.3. Factores que determinan la calidad de un esquema

A la hora de elaborar un esquema conceptual de base de datos utilizando el modelo entidad−relación, se debe intentar que el diagrama resultante sea:

- *Completo.* Se reflejan todas las características de la realidad que pretende representar.
- *Correcto.* Utiliza correctamente todos los elementos necesarios del modelo entidad−relación.
- *No redundante.* No existen propiedades repetidas en el esquema.
- *Legible.* Una persona que no ha participado en la elaboración del esquema es capaz de comprenderlo.
- *Extensible.* Se adapta a futuros cambios que no afecten totalmente a las estructuras del diagrama.
- *Normalizado.* El esquema está en la forma normal de Boyce−Codd, esta propiedad puede posponerse hasta que el diagrama se transforme al modelo lógico de datos elegido (relacional, jerárquico o de redes). Se opta por esta posibilidad dado que el proceso de normalización lleva asociados conceptos inherentes al modelo relacional de datos.

- **FUNDAMENTOS DEL MODELO RELACIONAL**

2.3.1. Arquitectura

- *Esquema conceptual.* Se define mediante relaciones que se representan con tablas. Cada entidad y relación se corresponde con una tabla bidimensional. Las columnas son los atributos y las filas las ocurrencias.

- *Esquema externo.* Se describe mediante *vistas* que se corresponden con los subesquemas de la

arquitectura estándar. Una vista es una tabla virtual que se forma a partir de las tablas del esquema conceptual, pero que no tiene correspondencia en el nivel interno.

- *Esquema interno.* Cada tabla del esquema conceptual se almacena en un archivo. Para cada clave candidata se crea un índice para el posterior acceso directo a los datos del archivo.

El siguiente gráfico resume la arquitectura de un sistema gestor de base de datos relacional.

.....

Tabla 1 Tabla n

.....

Indice 1 Indice i Indice 1 Indice j

....

.....

Archivo 1 Archivo n

De las tablas se derivan los siguientes conceptos:

- *Tupla.* Es una fila de la tabla que se corresponde con cada ocurrencia de la relación.
- *Atributo.* Es cada una de las columnas de la tabla. Es equivalente al concepto de atributo del modelo entidad−relación.
- *Cardinalidad.* Es el número de filas de la tabla.
- *Grado.* Es el número de atributos de la tabla. Todas las tuplas tienen el mismo número de atributos.
- *Dominio de un atributo.* Es el conjunto de valores que puede tomar dicho atributo. Pueden ser:

- *Generales.* Los valores están comprendidos entre un mínimo y un máximo. Por ejemplo, el dominio del atributo Garantía es un número comprendido entre 3 y 18 que representa el número de meses mínimo y máximo que ofrece el concesionario como garantía del coche.
- *Restringidos.* Los valores pertenecen a un conjunto discreto. Por ejemplo, Sexo puede tomar los valores masculino y femenino, Color admitirá los colores válidos para el entorno de la relación, etc.

- *Clave.* Este concepto está definido de forma similar al expuesto en el modelo entidad−relación. Es un conjunto de atributos que identifican unívocamente cada tupla de la relación. Todas las claves de una relación son *candidatas*; la utilizada se denomina *primaria* y el resto son *alternativas*. Si un conjunto de atributos es clave en otra tabla distinta, se considera clave *foránea* o *externa*.

En general, una tabla debe reunir los siguientes requisitos para ser consideradas como una relación:

- Tener un número fijo de atributos para todas las tuplas.
- Cada atributo tiene un único dominio.
- Las tuplas no tienen por qué tener una secuencia determinada.
- El orden de los atributos no importa.
- No pueden existir tuplas ni atributos repetidos.
- Cada intersección fila−columna debe contener un valor único perteneciendo al dominio de la columna correspondiente.

<u>Vistas</u>: Los conceptos que se han expuesto hasta el momento hacían referencia al esquema conceptual del modelo relacional. Para describir el esquema externo se utilizan las vistas. Una vista es una tabla que el usuario puede crear y manejar. Es una tabla virtual que no tienen que corresponderse con ningún archivo del nivel interno. Las tuplas que pertenecen a una vista se obtienen como resultado de consultas a las tablas del nivel conceptual.

Las vistas pueden formarse eliminando atributos de una tabla, uniendo tablas por atributos comunes y de ambas formas. También pueden definirse a partir de otras vistas. Al crearlas se deben de tener en cuenta las restricciones impuestas por el administrador de la base de datos para cada usuario.

Usuario1 usuario2

Vista3

Nombre	Cuota

Vista1 Vista 2

Nombre	dirección	Población
DNI	Tipo	Cuota

DNI	Nombre	Dirección	Cod_postal	Tipo

Tabla1 Tabla2 Tabla3

Tipo	Cuota

Cod_postal	Población	Provincia

2.3.2. Reglas de Integridad

Al enumerar los objetivos que debe cumplir un sistema gestor de bases de datos se citaba la integridad. Parte de este objetivo se cumple si se consigue mantener la coherencia y la veracidad de la información en la base de datos. Las operaciones que pueden afectar a la integridad son la inserción, la modificación y el borrado de registros en la base de datos.

Existen algunas restricciones que deben cumplirse para mantener la base de datos íntegra, aunque no todos los sistemas gestores de bases de datos relacionales facilitan mecanismos de especificación de tales restricciones. Las tres restricciones principales que forman parte del modelo relacional de datos son las siguientes:

- *Integridad de entidad.* Ningún valor de la clave primaria de una relación puede ser nulo o desconocido. Tampoco se podrá desconocer parte de dicha clave primaria si está estuviera formada por varios atributos. Es razonable que si, por definición, la clave primaria permite distinguir las tuplas de una relación, las tuplas con valor nulo en la clave primaria no estarán identificadas. Esta restricción se consigue comprobando en cada inserción y modificación que ningún valor de los introducidos para la clave primaria es nulo o no forma parte de su dominio.
- *Integridad de clave.* Los valores de claves candidatas en una relación deben ser únicos para cada tupla. El razonamiento es análogo al de la restricción anterior. Para evitar violar esa regla, se comprobará en todas las inserciones y modificaciones que el valor de todas las claves candidatas no existen en alguna tupla de esa relación.

- *Integridad referencial.* Una tupla de una relación R1 que haga referencia a otra relación R2 debe referirse a una tupla existente en R2. Formalmente, cada valor de un atributo A que forma parte de una clave foránea en una relación R2, R3, etc., en la que A forma parte de la clave primaria. Esta regla puede verse afectada en la inserción, modificación y borrado de tuplas. Para las dos primeras operaciones, el sistema gestor de base de datos relacional debe asegurarse de que al introducir un valor de una clave foránea, ese valor sea nulo o exista en todas las relaciones en las que la clave foránea sea clave primaria. En cuanto a la eliminación, no se permitirá borrar una tupla de una relación cuyo valor de clave primaria exista como valor de la clave foránea de otra relación.

PROFESORES

NOMBRE	EDAD	TELEFONO
Juan		
José		
Rosa	27	1111
Javi	31	2222
Luis	31	3433
	33	3232
	21	3456

ASIGNATURAS

Nombre	Horas	Tipo
Matem.		
Idioma		
F y Q	5	Obl
Lengua	3	Opt
Método	5	Obl
	4	Opt
	5	Opt

IMPARTE

Profesor	Asignatura
Juan	
José	Matem.
Luis	Lengua
	Idioma

Además de estas restricciones formales, es conveniente que los sistemas gestores de bases de datos relacionales, tengan en cuenta lo siguiente:

- No se permitirá introducir valores de atributos que no pertenezcan a sus dominios o sean de distintos tipo.
- Se podrán definir atributos obligatorios y opcionales. Los obligatorios, sean claves o no, deberán contener siempre valores válidos.

- Se podrán formular reglas de gestión de la base de datos (*reglas semánticas*), tales como la fecha de devolución será mayor que la de préstamo, el mínimo de unidades suministradas por un proveedor será tres, todos los valores numéricos deberán ser positivos, un árbitro no podrá participar en dos partidos consecutivos de un mismo equipo, etc.

2.3.3. Paso del modelo entidad−relación al modelo relacional

Recuérdese que en el proceso de diseño de una base de datos el primer paso consistía en realizar el diseño conceptual para, posteriormente, abordar el diseño lógico que generaba el esquema de la base de datos en el modelo lógico elegido (relacional, jerárquico, de redes u orientado a objetos). Este apartado presenta un procedimiento sistemático para transformar el esquema conceptual resultante de aplicar el modelo entidad−relación al problema de diseño de una base de datos en un esquema relacional.

Para poder llevar a cabo esta transformación, es necesario realizar previamente determinadas conversiones que eliminen elementos del modelo entidad−relación no representables en el modelo relacional.

Antes de pasar al modelo relacional, es conveniente analizar el diagrama entidad−relación y comprobar que no existen elementos que no puedan ser representados directamente en el modelo relacional. Para ello pueden seguirse los siguientes pasos:

- *Eliminación de jerarquías de generalización.* Al hacerlo hay que tener en cuenta que no debe existir pérdida de información. Existen tres posibilidades para realizar la conversión:

  - Integrar todas las entidades en una única. Esta nueva entidad contendrá todos los atributos de la entidad genérica, los de las subentidades y un atributo discriminativo para distinguir a qué subentidad pertenecen las tuplas. Los atributos de las subentidades son tratados como opcionales. Todas las relaciones que tuvieran subentidades se mantienen ligadas a la nueva entidad recién creada, reconsiderando la cardinalidad de cada relación. Esta alternativa presenta el inconveniente de generar demasiados valores nulos en los atributos opcionales. También ralentiza el proceso de búsqueda al tener en cuenta todas las tuplas en vez de las que pertenecen a la subentidad deseada. Su única ventaja es que permite modelar todas las jerarquías (totales o parciales y superpuestas o exclusivas).
  - Considerar cada subentidad como entidad. Para ello, se añaden los atributos de la entidad genérica a la subentidad; y la clave primaria de la genérica pasa a serlo de las nuevas entidades creadas. Sus inconvenientes son varios:

  - Se pierde el concepto de la entidad genérica.
  - Los accesos a la entidad genérica deben convertirse en accesos a todas las subentidades.
  - Los atributos de la entidad genérica son repetidos en cada subentidad.
  - Sólo es válida para jerarquías totales y exclusivas.
  - Si la entidad genérica tiene alguna relación, ésta debe propagarse a cada subentidad.

En consecuencia, esta alternativa es válida cuando la jerarquía es total o exclusiva, no importa el concepto de la entidad genérica en las operaciones y no hay relación entre la entidad genérica y otras entidades.

Modelo inicial:

NIF M M Marca

Modelo

Color

Nombre

Stock Matrícula M 1

Transformación de jerarquía total y exclusiva:

a)

Nombre

1

M

Marca Tipo

Modelo Matricula

Color Stock

M

M

NIF

B) Marca Marca

Modelo Modelo

Color Color

Stock Stock

M M

M

M M 1

NIF Nombre

C)

Marca

Modelo M M Color NIF

1 1

Stock 1 Matrícula 1 M 1 Nombre

- Insertar una relación entre la entidad genérica y cada subentidad. Los atributos se mantienen y a las subentidades se las puede identificar con la clave foránea de la entidad genérica. Su inconveniente principal es que el esquema resultante es bastante complejo e, incluso puede resultar redundante. En cambio, permite representar todos los tipos de jerarquías.

No existe un criterio general para utilizar una alternativa u otra. En cada caso hay que plantearse las ventajas e inconvenientes que plantea cada una. En general, para tomar la decisión se debe tener en cuenta el tipo de jerarquía, los atributos de la entidad genérica, a qué entidades afectan las operaciones a realizar y la complejidad del esquema resultante.

- *Eliminar los atributos compuestos.* El modelo relacional sólo permite representar atributo simples, por lo que deben convertirse los atributos compuestos. Existen dos alternativas:

  - Considerar como atributo simple cada componente del compuesto; por ejemplo, el atributo Fecha−de−nacimiento compuesto de Día, Mes y Año convertirlo en tres atributos Día−de−nacimiento, Mes−de−nacimiento y Año−de−nacimiento.
  - Considerar el atributo compuesto como simple. En este caso es responsabilidad de la aplicación identificar cada parte del atributo. Para el atributo Fecha−de−nacimiento, pasaría a ser simple y, en vez de tener tres partes de dos dígitos, se representaría con seis dígitos manteniendo el significado de cada dígito antes de la conversión.

- *Eliminar atributos multivalor*. Pueden darse dos casos:

  - El atributo multivalor pertenece a una entidad. En este caso, se crea una nueva entidad que contiene el atributo multivalor como monovalor y la clave primaria de la entidad original. La clave primaria de la nueva entidad es la suma de sus atributos. Por ejemplo, el atributo Méritos en una entidad Candidato con atributos DNI..., Méritos se dividirá en dos entidades: Candidato con todos sus atributos, excepto Méritos, y la entidad Currículum con los atributos DNI y Méritos.
  - El atributo multivalor pertenece a una relación R entre dos entidades E1 y E2. También se crea una nueva entidad que contiene el atributo multivalor como simple y:

  - La clave primaria de una de las entidades participantes en la relación si es 1:1.
  - La clave primaria de la entidad del lado muchos si la relación es 1:M.
  - Las claves primarias de ambas relaciones si la relación es M:M.

METODOLOGÍA PARA REALIZAR LA CONVERSIÓN

- **Entidades**. Este elemento es equivalente en los dos modelos. Por lo tanto, basta con cambiar el nombre de la entidad por el de la relación, que será representada por una tabla. La clave primaria de la entidad también lo es de la relación.

DNI CLIENTE

DNI	Nombre	Dirección

Nombre

Dirección

- **Relaciones recursivas.** Una relación R de una entidad E consigo misma se convierte en dos relaciones del modelo relacional: una tabla para la entidad E y otra para R. La nueva tabla incluirá dos veces la clave primaria de E (una vez por cada papel de la relación) y los atributos de R. La clave primaria será una de

ellas si la relación es 1:M o la combinación de las dos si es M:M. Si la relación fuera 1:1 se repetiría la clave primaria con el segundo papel y no necesitaría la segunda tabla.

Nombre Empleado

Nombre	Dirección

Dirección

Nombre_dire	Nombre_dirigido

1 M DIRECTOR

Debe aclararse que siendo la relación 1:M, es posible utilizar una única relación con la clave primaria duplicada en los dos papeles. En este caso, el ejemplo se transformaría en la siguiente tabla:

EMPLEADO

Nombre_empleado	Teléfono	Nombre−director

Sin embargo, esta alternativa no es aconsejable cuando la relación no es obligatoria−obligatoria, ya que podrían generarse valores nulos (los directores no tienen por qué estar dirigidos).

- **Relaciones binarias.** La forma de tratarlas depende de su grado y cardinalidad. En general, todos los caso se refieren a una relación R entre dos entidades E1 y E2.
    - *Relación 1:1.* En principio puede modelarse con una única relación que incluya los atributos de E1,E2 y R. Sin embargo, pueden generarse demasiados valores nulos cuando alguna de las participaciones no sea obligatoria. Por ello, se deben tratar por separado.

DNI

Nombre

1

1

Num_proyecto

Denominación

♦ Obligatoria−obligatoria. Se integran las dos entidades en una tabla que contiene los atributos y la clave primaria es cualquiera de las de E1 y E2. Si coincidiera, sólo se incluiría una vez en la nueva relación. El ejemplo quedaría del siguiente modo:

ESTUDIANTE

DNI	Nombre	Núm_proyecto	Denominación

♦ Obligatoria−opcional. Cada entidad se convierte en una relación representada por una tabla, y a la que tiene participación obligatoria se añadiría la clave primaria de la opcional. Las claves

primarias de ambas relaciones se mantienen. Suponiendo ESTUDIANTE opcional, el ejemplo resultaría:

ESTUDIANTE

DNI	Nombre

PROYECTO

Núm_proyecto	Denominación	DNI

- Opcional−opcional. En este caso, se generarán tres relaciones: una para cada entidad y otra para la correspondencia entre ambas. Las entidades no sufriría cambios y la nueva relación incluirá las claves primarias de E1 y E2, así como los atributos de R, si los hubiera. La clave primaria de esta relación recién creada será cualquiera de las de E1 o E2. El ejemplo, sería:

ESTUDIANTE

DNI	Nombre

PROYECTO

Núm_proyecto	Denominación

PROYECTO−DE−ESTUDIANTE

DNI	Núm_proyecto

- *Relación 1:M.* En general, se modelan añadiendo la clave primaria de la entidad del lado uno a la del lado muchos. Sin embargo, al aparecer alguna entidad con participación opcional es conveniente elegir otra solución. Ejemplo :

Código

Nombre

1

Descuento

M

ISBN

Título

- Obligatoria−obligatoria. Cada entidad se convierte en una relación con su clave primaria, y la clave primaria de la entidad del lado uno se añade a la del lado lado muchos. Los atributos de la relación R, si hubieran, se incluyen en la del lado muchos. El ejemplo quedaría de esta forma:

EDITORIAL

Código	Nombre

LIBRO

ISBN	Título	Código	Descuento

- Obligatoria−opcional. Se necesitan tres relaciones, una para cada entidad y otra para la correspondencia. Las entidades se transforman convirtiéndolas en tablas, y la nueva relación debe contener las claves primarias de las dos entidades y los posibles atributos de la correspondencia R. La clave primaria de la relación creada será la combinación de las claves primarias de E1 y E2; en el ejemplo:

EDITORIAL

Código	Nombre

LIBRO

ISBN	Título

SUMINISTRA

Código	ISBN	Descuento

- Opcional−opcional: Este caso se reduce al de relación obligatoria−opcional.

- *Relación M:M*. Para estas relaciones no importa la cardinalidad. Siempre se transforman en tres relaciones, una para entidad y otra para la correspondencia. Las entidades se convierten en tablas y la relación de la correspondencia incluye las claves primarias de las entidades y los atributos de la asociación, siendo su clave primaria la combinación de las claves primarias de las entidades. Como ejemplo, véase la siguiente relación M:M que muestra la figura:

DNI

Nom_alum

M

Nota_final

M

Nom_asig

Horas

Lo que quedaría:

ALUMNO

DNI	Nom_alum

ASIGNATURA

Nom_asig	Horas

NOTA

DNI	Nom_asig	Nota_final

- **Relaciones n−arias**. Tienen el mismo tratamiento que las relaciones M:M. Cada entidad se transforma en tabla y se añade una tabla para la asociación que incluya las claves primarias de las entidades participantes en la correspondencia y los atributos de la asociación. La composición de las claves primarias de las entidades es la clave primaria de la nueva relación. Por ejemplo, la siguiente relación ternaria quedaría del modo que muestra la figura:

DNI_CLI

Teléfono

Matricula

Pvp Forma_pago

Kms

DNI_Emp

Nombre

Lo que quedaría:

CLIENTE

DNI_CLI	Teléfono

COCHE

Matrícula	Pvp	Km

EMPLEADO

DNI_Emp	Nombre	Dirección

VENTA

DNI_cli	DNI_Emp	Matrícula	Forma_pago

TEMA III

PANORAMICA GENERAL DE SQL

- **EL LENGUAJE SQL**

- **OBJETIVOS DE SQL**
- **CARACTERISTICAS Y VENTAJAS DE SQL**
- **TAREAS DE USUARIO Y DE ADMISNISTRADOR**
- **TIPOS DE ORDENES DEL LENGUAJE SQL**
- **FORMAS DE TRABAJAR DE SQL**
- **ENTORNO DE TRABAJO**
- **SQL Y LA INTERCONEXION POR LA RED**
- **RESUMEN**
- **EJERCICIOS**

Notas

3.1. EL LENGUAJE SQL

SQL es una herramienta para organizar, gestionar y recuperar datos almacenados en una base de datos informática. SQl es un *lenguaje* informático que se utiliza para interactuar con una base de datos. De hecho, SQL funciona con un tipo específico de base de datos, llamado *base de datos relacional*.

En la actualidad es el lenguaje de uso y programación de bases de datos relacionales más extendidos.

El programa que controla la base de datos se llama *Sistema de Gestión de Base de Datos, o DBMS.*

Cuando es necesario recuperar datos de una base de datos, la petición se realiza utilizando SQL. El DBMS procesa la petición SQL, recoge los datos solicitados y los devuelve a quien los solicitó. Este proceso de petición de datos de la base de datos y posterior recepción de resultados se llama consulta (*query*); de aquí el nombre lenguaje estructurado de *consultas*.

Este es un lenguaje para el manejo de la base de datos a lo largo de todo su ciclo de vida.

Este lenguaje trabaja de modo declarativo. Cuando un usuario quiere realizar una operación con los datos, no debe describirla paso a paso. Basta con especificar el resultado que se desea mediante cláusulas y predicados. El gestor de la base de datos se ocupará de realizar las tareas necesarias para hacer efectiva la petición.

SQL se utiliza para controlar todas las funciones que suministra un DBMS a sus usuarios, incluyendo:

- *Definición de datos.* SQL permite que un usuario defina l estructura y la organización de los datos almacenados, así como las relaciones existentes entre ellos.

- *Recuperación de datos.* SQL permite a un usuario o a un programa recuperar y utilizar los datos almacenados en una base de datos.

- *Manipulación de datos.* SQL permite a un usuario o a un programa actualizar la base de datos añadiendo datos nuevos, borrando los viejos y modificando los almacenados previamente.

- *Control de acceso.* SQL puede ser utilizado para restringir la capacidad de un usuario para recuperar, añadir y modificar datos, protegiendo los datos almacenados contra accesos no autorizados.

- *Compartición de información.* SQL es utilizado para coordinar la compartición de datos entre usuarios concurrentes, asegurando que no haya interferencias entre ellos.

- *Integridad de datos.* SQL define restricciones de integridad en la base de datos, protegiéndola de alteraciones debidas a actualizaciones inconsistentes o fallos del sistema.

En segundo lugar, SQL no es realmente un lenguaje completo como COBOL, FORTRAN o C. SQL no contiene sentencias IF para probar condiciones, ni sentencias GOTO de salto, Ni siquiera sentencias DO o FOR para realizar bucles. En vez de eso, SQL es un sublenguaje de base de datos que consiste en alrededor de treinta sentencias especializadas en tareas de gestión de base de datos. Estas sentencias SQL están embebidas en otro lenguaje, como FORTRAN, COBOL o C, que las utiliza para acceder a las bases de datos.

Finalmente, SQL no es un lenguaje particularmente estructurado, especialmente cuando se compara con lenguajes altamente estructurados como C o Pascal. Hay bastantes inconsistencias en el lenguaje SQL, y además existen reglas especiales que evitan la construcción de sentencias SQL que parecen perfectamente legales, pero que no tienen sentido.

3.2. OBJETIVOS DE SQL

El *motor de la base de datos* es el corazón del DBMS, responsable de la estructuración, almacenamiento y recuperación de los datos del disco. Acepta peticiones SQL de otros componentes del DBMS, tales como facilidades de formularios, generadores de informes o facilidades de consultas interactivas, de programas escritos por los usuarios e incluso de otros sistemas informáticos.

- SQL es un *lenguaje interactivo de consultas.* Los usuarios escriben órdenes SQL en un programa SQL interactivo para recuperar datos y presentarlos en la pantalla.

- SQL es un *lenguaje de programación de bases de datos.* Los programadores introducen órdenes SQL en sus programas de acceder a los datos de las bases de datos.

- SQL es un *lenguaje de administración de base de datos.* El administrador responsable de la gestión de la base de una minicomputadora o sistema basado en una computadora central utiliza SQL para definir la estructura de la base de datos y el control de acceso a los datos almacenados.

- SQL es un *lenguaje cliente/servidor.* Los programas de las computadoras personales utilizan SQL para comunicarse, a través de una red de área local, con los servidores de bases de datos que almacenan los datos compartidos.

- SQL es un *lenguaje de bases de datos distribuidos.* Los sistemas de gestión de bases de datos distribuidas utilizan SQL para ayudar a distribuir los datos a través de muchos sistemas informáticos conectados.

3.3. CARACTERISTICAS Y VENTAJAS DE SQL

A continuación se muestran las principales características de SQL y las tendencias del mercado que le han hecho conseguir el éxito.

- Independencia de los proveedores
- Portabilidad entre sistemas informáticas
- Acuerdos con Microsoft (ODBC)
- Fundamento relacional
- Múltiples vistas de los datos
- Lenguaje completo de base de datos
- Definición dinámica de base de datos
- Arquitectura cliente/servidor

Independencia de los proveedores

Ningún nuevo producto de base de datos puede tener éxito si no ofrecen SQL. Una base de datos basada en SQL, y los programas que la usan, se pueden transferir del DBMS de un proveedor a otro con un esfuerzo mínimo de conversión y un pequeño reciclaje personal.

El SQL es tan popular en el entorno de las bases de datos, que muchos lenguajes de programación incluyen sentencias SQL como parte de su repertorio de instrucciones. Tal es el caso del lenguaje Visual Basic que, cuando se utiliza para acceder a las bases de datos creadas con cualquier producto relacional, permite describir un subnconjunto de los registros mediante una sentencia SQL.

Portabilidad entre sistemas informáticos

Los proveedores de DBMS basados en SQL ofrecen sus productos para un rango de computadoras que incluye las computadoras personales y estaciones de trabajo en redes de área local, las minicomputadoras y los sistemas basados en una computadora central. Las aplicaciones basadas en SQL que comienzan en sistemas monousuarios pueden ser transferidas a minicomputadoras mayores o a sistemas basados en una computadora central, según crecen.

El lenguaje SQL ha sido utilizado con diferentes sistemas comerciales, como lenguaje nativo o como una incorporación posterior a un sistema relacional preexistente, como INGRES.

Tras convertirse en un lenguaje estándar por ISO, se han desarrollado multitud de productos comerciales SQL. Existen Intérpretes de SQL en micros, minis y grandes ordenadores, sobre diferentes sistemas operativos.

Acuerdos con Microsoft (ODBC)

Microsoft considera el acceso a las bases de datos como un elemento clave de la arquitectura software de Windows para computadoras personales. El estándar de Microsoft que proporciona acceso a bases de datos es ODBC (conexión entre base de datos abiertas, Open Database Connectivity), una facilidad basada en SQL. Las principales aplicaciones Windows, tanto de Microsoft como de otros proveedores líderes de aplicaciones Windows son compatibles con ODBC, y el acceso ODBC está disponible o anunciado en las principales bases de datos SQL.

Fundamento relacional

La estructura tabular fila/columna de las bases de datos relacionales es intuitiva para los usuarios, manteniendo el lenguaje sencillo y fácil de entender. El modelo relacional también tiene un fuerte fundamento teórico que ha guiado la evolución y la implementación de las bases de datos relacionales.

Múltiples vistas de los datos

Utilizando SQL, el creador de una base de datos puede dar vistas diferentes de su estructura y contenidos a diferentes usuarios de la base de datos. Por ejemplo, la base de datos puede ser construida de modo que cada usuario sólo vea datos de su propio departamento o de su propia región de ventas. Además, los datos procedentes de diferentes partes de la base de datos pueden combinarse y presentarse al usuario como una simple fila/columna de una tabla. Las vistas se SQL pueden ser utilizadas de este modo para mejorar la seguridad de una base de datos y para acomodarla a las necesidades particulares de los usuarios individuales.

Lenguaje completo de base de datos

SQL fue inicialmente desarrollado como un lenguaje de consulta *ad hoc*, pero su potencia va ahora más allá de la recuperación de datos. SQL proporciona un lenguaje completo y consistente para crear una base de datos, gestionar su seguridad, actualizar sus contenidos, recuperar los datos y compartirlos entre muchos

usuarios concurrentes.

Definición dinámica de datos

Utilizando SQL, la estructura de una base de datos puede ser modificada y ampliada dinámicamente, incluso mientras los usuarios están accediendo a los contenidos de la base de datos. Este es un avance importante sobre los lenguajes de definición de datos estáticos.

Arquitectura cliente/servidor

SQL es un vehículo natural para implementar aplicaciones utilizando una arquitectura cliente/servidor distribuida.

3.4. TAREAS DE USUARIO Y DE ADMINISTRADOR

La existencia de una base de datos de tamaño medio requiere un mantenimiento bastante costoso. Por ello, se ponen en marcha bases de datos cuando hay varias personas interesadas en su aprovechamiento. Un grupo de personas que intenta utilizar unos datos genera problemas ya conocidos, como las violaciones de integridad y los posibles errores por acceso simultáneo.

Para resolver estos y otros conflictos aparece la figura del administrador de la base de datos (ABD ODBA, *Data Base Administrator).* Esta persona posee todos los privilegios o permisos de acceso sobre los datos, y se ocupa de otorgar algunos de ellos a los usuarios para que éstos realicen su trabajo.

Las personas (o programas) que acceden a la base de datos reciben el nombre genérico de *usuarios.* Pero no todos realizan las mismas tareas. Se pueden clasificar en programadores y usuarios finales:

- *Programadores.* Se trata de personas que no están interesadas propiamente en los datos, sino en desarrollar un programa que trabaje sobre los datos. Este programa será utilizado por un usuario final cuando esté acabado. Para acceder a los datos, su programa dialoga con el sistema gestor de la base de datos (SGBD). Los programadores acceden a los datos al menos para probar la aplicación que están desarrollando.
- *Usuarios finales.* Son aquellos a los que interesan los datos almacenados en la base de datos, para modificarlos (como en el caso del empleado de un banco) o simplemente consultarlos (como en el acceso a los fondos de una biblioteca). Estos usuarios finales tienen dos opciones para trabajar: utilizar un lenguaje conversacional propio del SGBD para el diálogo, o bien aprovechar una aplicación o programa hecho a medida por un programador.
- **Tareas del administrador**

    - La creación y destrucción de los objetos de la base de datos (tablas, vistas, usuarios, grupos). El administrador diseña la organización de la base de datos y la pone en marcha para que los usuarios obtengan buen servicio de ella. Creará los esquemas de la base de datos y los dotará de contenido.
    - La autorización de acceso a los objetos. Tanto un usuario que preguntan por un valor como un usuario que ejecuta un programa necesitan permiso para ver los datos. Estos permisos son otorgados por el administrador, de acuerdo con el tipo de usuario de que se trate. Por ejemplo, en una biblioteca no disponen de iguales permisos el encargado que registra los préstamos (y puede hacer anotaciones) que el lector que examina los títulos.
    - La gestión del almacenamiento físico y del espacio en disco.

    - La política de copias de seguridad y la restauración de la base de datos en caso de caída.

Un usuario puede crear objetos particulares y permitir a otros trabajar con ellos. En este caso, será un usuario quien realice las tareas de administración.

3.5. TIPOS DE ORDENES DEL LENGUAJE SQL

Una base de datos es un sistema de hardware, software y datos al servicio de una organización (empresa, institución, centro de estudios, etc.). Existen dos fases en la vida de una base de datos: la etapa de preparación y puesta en marcha, y la etapa de explotación. Esta última es el objetivo de todo el sistema, y las tareas preparatorias se realizan antes de entrar en esa fase de utilidad para la organización.

Un lenguaje para manejar bases de datos se compone de un conjunto o repertorio de instrucciones, al igual que en un lenguaje de programación habitual. Debido a la clasificación de las tareas que se realizan sobre la base de datos en dos grupos, se suelen distinguir dos sublenguajes, cada uno de los cuales sirve para un tipo de actividad diferente. En particular, en los lenguajes sobre bases de datos relacionales se distinguen dos partes:

- DDL (Data Description Language, lenguaje de descripción de datos). Es el lenguaje utilizado para la creación (y mantenimiento) de la estructura de la base de datos. Con este lenguaje se definen y modifican los esquemas de las relaciones, se crean o destruyen índices y se eliminan relaciones. Además, pueden definirse vistas y permisos de acceso para los usuarios.
- DML (Data Manipulation Language, lenguaje de manipulación de datos). Incluye las instrucciones para consultar la base de datos, así como para insertar o eliminar tuplas y modificar valores de datos. Este lenguaje es el utilizado para la fase de explotación o de trabajo útil de la base de datos, y es empleado por los usuarios finales.

Además existen tareas en el trabajo sobre una base de datos que no son propiamente de descripción ni de manipulación de datos. Por ejemplo, entran en el tercer grupo de instrucciones SQL la asignación de privilegios a un usuario (sentencia GRANT).

3.6. FORMAS DE TRABAJAR CON SQL

El lenguaje SQL se utiliza para actuar sobre una base de datos de alguno de estos modos:

- De modo interactivo. Desde un terminal se establece una conversación entre el usuario y el programa gestor de la base de datos (SGBD). Es similar al modo en que se mantiene un diálogo con un sistema operativo. Cualquier orden SQL, sea DDL, DML u otra, puede introducirse por este método, y no hay restricciones en el orden entre ellas (pueden mezclarse consultas con creaciones de tablas, por ejemplo). Responde al esquema de trabajo descrito en la siguiente figura.

Usuario Base de datos

- Desde un programa. Un usuario ejecuta sobre el sistema operativo una aplicación. Esta puede ser de dos tipos:
- Un programa escrito íntegramente en SQL. Existen extensiones al SQL que lo dotan de las estructuras de programación imperativa usuales (bucles, selección). Se trata de un lenguaje procedural que permite desarrollar programas imperativos que finalmente acceden a la base de datos vía SQL.
- Un programa escrito en un lenguaje convencional de programación parte de cuyo texto está escrito en SQL. A esta variante de utilización del lenguaje se le llama SQL *inmerso o* embebido, y al lenguaje que contiene el texto se le conoce como lenguaje huésped (host). Las instrucciones del lenguaje de programación se ejecutan por el procedimiento usual, y las instrucciones SQL se le pasan a un módulo especial de ejecución del SGBD. Puede verse un esquema en el siguiente gráfico. Una implementación de SQL embebido establece las relaciones que deben mantener los objetos de la base de datos con los objetos del programa huésped, así como ciertas restricciones de funcionamiento.

Base de datos

Programa

de usuario

El modo de trabajo con SQL programado aún admite dos variantes:

- SQL estático. El programa no admite cambios durante la ejecución. Éste es el método usado en la mayoría de las aplicaciones.
- SQL dinámico. Si durante la ejecución varía algún parámetro o parte del programa se necesita utilizar SQL dinámico. Es menos eficiente que un programa en SQL estático y utiliza técnicas dinámicas de manejo de variables, lo que hace su uso más difícil para el programador.

3.7. ENTORNO DE TRABAJO

Para el uso del lenguaje SQL es necesario un cierto entorno de trabajo. Éste se compone de varios elementos que se describen a continuación. Un presunto usuario de una base de datos bajo SQL debe disponer de:

- Sistema operativo. La máquina en que el usuario realiza su trabajo se denomina *local*. Lo usual es que una base de datos SQL se active sobre una plataforma multiusuario con varios puestos de trabajo. El ordenador local no suele tener instalado todo el SGBD, sino que estará distribuido entre varias máquinas. Por tanto, el usuario deberá tener acceso al sistema operativo o red que se utilice.
- Aplicación para conexión con la base de datos. Una vez que el usuario se ha conectado al sistema operativo, debe conectarse a la base de datos.
- Acceso a la base de datos. Debe existir como usuario autorizado en la base de datos.
- Utilidades. Se trata de programas diseñados para realizar operaciones sobre la base de datos. La utilidad más básica es un terminal de texto en el que introducir órdenes SQL.
- Útiles de administración del sistema. El ABD utilizará para su trabajo algunas herramientas que están vedadas al resto de usuarios, como programas que evalúen el rendimiento del sistema o aplicaciones para la gestión de usuarios.
- Compiladores. El usuario puede trabajar con SQL sólo en modo interpretado, para lo cual le basta con los útiles enumerados anteriormente. Pero si desarrolla o simplemente utiliza un programa ya desarrollado por un programador puede que necesite compilarlo.

3.8. SQL Y LA INTERCONEXION POR LA RED

La creciente popularidad de la interconexión de computadoras por red durante los últimos años ha tenido un fuerte impacto en la gestión de base de datos y ha dado a SQL una nueva importancia. En esta arquitectura, tanto el DBMS como los datos físicos residen en un mainframe o minicomputadora central, junto con el programa de aplicación que acepta entradas desde el terminal de usuario y muestra los datos en la pantalla del usuario.

Supongamos que el usuario teclea una consulta que requiere una búsqueda secuencial en una base de datos, como por ejemplo la petición para hallar la cantidad media de mercancías de todos los pedidos. El DBMS recibe la consulta, explora la base de datos para acceder a cada uno de los registros de datos del disco, calcula el promedio y muestra el resultado en la pantalla terminal. Tanto el procesamiento de la aplicación como el procesamiento de la base de datos se producen en la computadora central, y como el sistema es compartido por muchos usuarios, cada usuario experimenta una degradación del rendimiento cuando el sistema está más cargado.

Arquitectura servidora de archivos

En esta arquitectura, una aplicación que se ejecuta en una computadora personal puede acceder de forma

transparente a los datos localizados en un servidor de archivos, que almacena los archivos compartidos. Cuando la aplicación de PC solicita los datos de un archivo compartido, el software de red recupera automáticamente el bloque solicitado del archivo en el servidor.

Esta arquitectura proporciona un rendimiento excelente para consultas típicas, ya que cada usuario dispone de la potencia completa de una computadora personal ejecutando su propia copia del DBMS.

Arquitectura cliente/servidor

Muestra la emergente arquitectura *cliente/servidor* de gestión de base de datos. En esta arquitectura las computadoras personales están combinadas en una red de área local junto con un *servidor de base de datos* que almacena las bases de datos compartidas. Las funciones del DBMS están divididas en dos partes. Los frontales (*fronts−ends*) de base de datos, tales como herramientas de consulta interactiva, generadores de informe y programas de aplicación, se ejecutan en la computadora personal. El motor de soporte (*back−end*) de la base de datos que almacena y gestiona los datos se ejecuta en el servidor. SQL se ha convertido en el lenguaje de base de datos estándar para comunicación entre las herramientas frontales y el motor soporte de esta arquitectura.

En la arquitectura cliente/servidor, la consulta viaja a través de la red hasta el servidor de base de datos como una petición SQL. El motor de base de datos el servidor procesa la petición y explora la base de datos, que también reside en el servidor. Cuando se calcula el resultado, el motor de la base de datos envía de vuelta, a través de la red, una única contestación a la petición inicial, y la aplicación frontal la muestra en la pantalla del PC.

La arquitectura cliente/servidor reduce el tráfico de red y divide la carga de trabajo de la base de datos. Las funciones íntimamente relacionadas con el usuario, tales como el manejo de entradas y la visualización de datos, se concentran en el PC. Las funciones de intenso procesamiento de datos, tales como la entrada/salida de archivos y el procesamiento de consultas, se concentran en el servidor de la base de datos. Lo que es más importante, el lenguaje SQL proporciona un interfaz bien definido entre los sistemas frontales y de soporte, comunicando las peticiones de acceso a la base de datos de una manera eficiente.

Las implementaciones de SQL Server, Oracle, Informix e Ingres para LAN de PC y SQLBase de Gupta Technologies utilizan este enfoque.

3.9. RESUMEN

- SQL es una herramienta para organizar, gestionar y recuperar datos.
- El programa que controla la base de datos se llama sistema de gestión de base de datos (DBMS).
- SQL puede ser usado para restringir la capacidad de un usuario para recuperar, añadir y modificar datos.
- SQL define restricciones de integridad en la base de datos, protegiéndola de alteraciones debidas a actualizaciones inconsistentes o fallos del sistema.
- SQL es un lenguaje cliente/servidor.
- SQL es un lenguaje de bases de datos distribuidos. Los datos de una base de datos pueden estar distribuidos entre muchos sistemas informáticos conectados.
- Los proveedores de DBMS basados en SQL ofrecen sus productos para un rango de computadoras de

distintos sistemas operativos.

- En los lenguajes sobre bases de datos relacionales se distinguen dos partes: DDL y DML.
- Con SQL se puede trabajar en modo interactivo y desde un programa.

3.10. EJERCICIOS

- Localizar al administrador de la base de datos que se tenga disponible. Averiguar que tareas realiza durante las fases de puesta en marcha de una base de datos y durante el mantenimiento posterior.
- Poner ejemplos de tareas que puedan desarrollar sobre una base de datos un usuario final y un programador.
- De entre las operaciones sobre tablas ya conocidas, ¿cuáles corresponden al DML y cuáles DDL?
- Poner ejemplos de usos de una base de datos en que convenga el uso interpretado del lenguaje SQL y otros en que convenga el uso desde un programa.

TEMA IV

GRAMATICA DE SQL

- **SENTENCIAS**
- **NOMBRES**
- **TIPOS DE DATOS**
- **CONSTANTES**
- **EXPRESIONES**
- **FUNCIONES INTERNAS**
- **RESUMEN**

Notas

4.1. SENTENCIAS

El lenguaje SQL consta de unas treinta sentencias, que a continuación se resumen. Cada sentencia demanda una acción específica por parte del DBMS, tal como la creación de una nueva tabla, la recuperación de datos o la inserción de nuevos datos en la base de datos.

Sentencia	**Descripción**
Manipulación de datos	
SELECT	Recupera datos de la BD
INSERT	Añade nuevas filas de datos de la BD
DELETE	Suprime filas de la BD
UPDATE	Modifica datos existentes en la BD
Definición de datos	
CREATE TABLE	Añade una base de datos a la BD
DROP TABLE	Suprime una tabla de la BD
ALTER TABLE	Modifica la estructura de una tabla existente
CREATE VIEW	Añade una nueva vista a la BD
DROP VIEW	Suprime una lista de la BD
CREATE INDEX	Construye un índice para una columna
DROP INDEX	Suprime el índice para una columna

CREATE SYNONUM	Define un alias para un nombre de tabla
DROP SYNONUM	Suprime un alias para un nombre de tabla
COMMENT	Define comentarios para una tabla
LABEL	Define el título de una columna
Control de acceso	
GRANT	Concede privilegios de acceso a usuarios
REVOKE	Suprime privilegios de acceso a usuarios
Control de transacciones	
COMMIT	Finaliza la transacción actual
ROLLBACK	Aborta la transacción actual
SQL programático	
DECLARE	Define un cursor para una consulta
EXPLAIN	Describe el plan de acceso a datos para una consulta
OPEN	Abre un cursor para recuperar resultados de consulta
FETCH	Recupera una fila de resultados de consulta
CLOSE	Cierra un cursor
PREPARE	Prepara una sentencia SQL para ejecución dinámica
EXECUTE	Ejecuta dinámicamente una sentencia SQL
DESCRIBE	Describe una consulta preparada

Todas las sentencias SQL tienen la misma forma básica ilustrada en la siguiente figura:

Verbo Nombre de tabla Cláusulas

DELETE FROM INFVENTAS

Palabras clave WHERE VENTAS < 2000.00

Nombres de columna Constante

Todas las sentencias SQL comienzan con un verbo, una palabra clave que describe lo que la sentencia hace. *CREATE, INSERT, DELETE* y *COMMIT* son verbos típicos. La sentencia continúa con una o más cláusulas. Una cláusula puede especificar los datos sobre los que debe actuar la sentencia, o proporcionar más detalles acerca de lo que la sentencia se supone que hace. Todas las cláusulas comienzan también con una palabra clave, tal como *WHERE, FROM, INTO* y *HAVING*. Algunas cláusulas son opcionales; otras son necesarias. La estructura y contenido específicos varían de una cláusula a otra. Muchas cláusulas contienen nombres de tablas o columnas; algunas pueden contener palabras claves adicionales, constantes o expresiones.Generalmente es buena idea evitar palabras clave al nombrar tablas y columnas.

La siguiente tabla lista las palabras clave incluidas en el estándar SQL.

ADA	CURRENT	GO	OF	SOME
ALL	CURSOR	GOTO	ON	SQL
AND	DEC	GRANT	OPEN	SQLCODE

ANY	DECIMAL	GROUP	OPTION	SQLERROR
AS	DECLARE	HAVING	OR	SUM
ASC	DEFAULT	IN	ORDER	TABLE
AUTHORIZATION	DELETE	INDICATOR	PASCAL	TO
AVG	DESC	INSERT	PLI	UNION
BEGIN	DISTINCT	INT	PRESICION	UNIQUE
BETWEEN	DOUBLE	IINTEGER	PRIMARY	UPDATE
BY	END	INTO	PRIVILEGES	USER
C	ESCAPE	IS	PROCEDURE	VALUES
CHAR	EXEC	KEY	PUBLIC	VIEW
CHARACTER	EXISTS	LANGUAGE	REAL	WHENEVER
CHECK	FETCH	LIKE	REFERENCES	WHERE
CLOSE	FLOAT	MAX	ROLLBACK	WITH
COBOL	FOR	MIN	SCHEMA	WORK
COMMIT	FOREIGN	MODULE	SECTION	

A lo largo de todo este manual, las formas aceptables de una sentencia SQL se ilustran mediante un diagrama sintáctico, como el que se muestra en el siguiente gráfico.

DELETE FROM nombre de tabla

WHERE condición de búsqueda

Una sentencia SQL válida o una cláusula se construye siguiendo la línea a través del diagrama. Las palabras claves del diagrama sintáctico y de los ejemplos (tales como DELETE y FROM) se muestran siempre en mayúsculas, pero casi todas las implementaciones SQL aceptan las palabras clave tanto en mayúsculas como en minúsculas, y con frecuencia es más conveniente declararlas efectivamente en minúsculas.

Los ítems variables de una sentencia SQL (tales como el nombre de una tabla y la condición de búsqueda en el gráfico anterior) se muestran en *cursiva minúscula.* Es cuestión del usuario especificar el ítem adecuado cada vez que utilice la sentencia. Las cláusulas y palabras clave opcionales, tales como la cláusula WHERE del gráfico anterior se indican mediante caminos alternativos dentro del diagrama sintáctico. Cuando se ofrece una elección de palabras clave opcionales, la opción por omisión (es decir, el comportamiento de la sentencia si no se especifica palabra clave) está subrayada.

- **NOMBRES**

Los objetos de una base de datos basada en SQL se identifican asignándoles nombres únicos. Los nombres se utilizan en las sentencias SQL para identificar el objeto de la base de datos sobre la que la sentencia debe actuar. El estándar SQL especifica nombres de tabla (que identifican tablas), nombres de columna (que identifican columnas) y nombres de usuario (que identifican usuarios de la base de datos). Muchas implementaciones SQL contemplan objetos nominados adicionalmente, tales como procedimientos almacenados (Sybase y SQL Server), relaciones clave primaria/clave ajena (DB2) y formularios de entrada de datos (Ingres).

El estándar SQL especifica que los nombres deben contener de 1 a 18 caracteres, comenzar con una letra, y que no pueden contener espacios o caracteres de puntuación especiales. En la práctica los nombres contemplados por los productos DBMS basados en SQL varían significativamente. **DB2**, por ejemplo, restringe los nombres de usuario a ocho caracteres, pero permite 18 caracteres en los nombres de tablas y

columnas. **ORACLE** permite nombre de tablas y columnas de 30 caracteres, y muchas otras implementaciones SQL también permiten nombres más largos. Los diferentes productos también se diferencian en los caracteres especiales que permiten en los nombres de tablas. Para conseguir portabilidad es mejor mantener los nombres relativamente breves y evitar el uso de caracteres especiales.

Nombres de tabla

Cuando se especifica un nombre de tabla en una sentencia SQL, se presupone que se refiere a una de las tablas propias (es decir una tabla ya creada). Con el permiso adecuado, también se puede hacer referencia a tablas propiedad de otros usuarios, utilizando un ***nombre de tabla cualificado***. Un nombre de tabla cualificado especifica el nombre del propietario de la tabla junto con el nombre de la tabla, separados por un punto (.). Por ejemplo, la tabla CUMPLEAÑOS, propiedad del usuario de nombre SAM, tiene el nombre de tabla cualificado.

SAM.CUMPLEAÑOS

Un nombre de tabla cualificado puede ser utilizado generalmente dentro de una sentencia SQL en cualquier lugar donde pueda aparecer un nombre de tabla.

Nombres de columna

Cuando se especifica un nombre de columna en una sentencia SQL, SQL puede determinar normalmente a qué columna se refiere a partir del contexto. Sin embargo, si la sentencia afecta a dos columnas con el mismo nombre correspondientes a dos tablas diferentes, debe utilizarse un *nombre de columna cualificado* para identificar sin ambigüedad la columna designada. Un nombre de columna cualificado especifica tanto el nombre de la tabla que contiene la columna como el nombre de la columna, separados por un punto (.). Por ejemplo, la columna de nombre VENTAS de la Tabla REPVENTAS tiene el nombre de columna cualificado:

REPVENTAS. VENTAS

Si la columna procede de una tabla propiedad de otro usuario, se utiliza un nombre de tabla cualificado en el nombre de columna cualificado. Por ejemplo, la columna DIA_MES de la Tabla CUMPLEAÑOS propiedad del usuario SAM se especifica mediante el nombre de columna cualificado completo:

SAM.CUMPLEAÑOS.DIA_MES

Los nombres de columna de cualificados se pueden utilizar generalmente en una sentencia SQL en cualquier lugar en el que pueda aparecer un nombre de columna simple (no cualificado); las excepciones se indican en las descripciones de las sentencias SQL individuales.

- **TIPOS DE DATOS**

SQL define una serie de tipos de datos que va a gestionar. El propósito de la definición de tipos es múltiple.

- Es necesario conocer el tamaño a reservar para cada uno de los datos nuevos que se van a introducir.
- Dividiendo los datos en diferentes tipos podemos implementar restricciones de integridad.

Ejemplo: Dato tipo fecha: 17/13/97. Mes entre 1 y 12.

- Una buena definición de los tipos de datos ayuda a los administradores y programadores en su tarea.

SQL define los siguientes tipos de datos:

- **Números exactos.**

    - *Números enteros*: INT ó INTEGER. Hace referencia a un tipo de dato entero de 32 bits (0 y 232).
    - *Números enteros cortos*: SMALLINT. Hace referencia a un tipo de dato entero de 16 bits (0 y 216).
    - TINYINT. Tiene un rango de números de 0 a 255. Ocupa 1 byte.

- **Números decimales.**

    - **NUMERIC (precision, escala)**
    - **DECIMAL (precision, escala)**

En ambos hay que especificar dos datos, *la precisión*, que es el número de dígitos en el número; *y la escala*, que es la cantidad de dígitos que están a la derecha del punto decimal. La diferencia entre ellos es que NUMERIC tiene una longitud (parte entera más decimal) fija, mientras que esto no ocurre con DECIMAL. Tanto en un tipo como en el otro, la **escala<=precisión**.

- **Números aproximados.**

    - *Números reales*; REAL. Define un número decimal de coma flotante con hasta 7 dígitos de mantisa. 4 Byte de almacenamiento.
    - *Números flotantes*: FLOAT (precision). Es necesario especificar la precisión del número.Ocupa 8 Byte de almacenamiento. Tiene 15 digitos de precisión.

- **Cadenas de caracteres.**

    - *Caracteres fijos*: CHAR (numero) ó CHARACTER (numero). Siempre se almacena el número de caracteres indicado en numero, aún cuando sea necesario rellenar los espacios sobrantes con caracteres en blanco porque la longitud de la cadena de caracteres sea inferior a numero.
    - *Caracteres variables*: VARYING CARÁCTER (VARCHAR). Sólo se almacenan los caracteres que se hayan introducido, hasta un máximo que viene dado por numero.

VARCHAR(numero)

Cadenas de bits.

- BIT (numero)

Teniendo numero el mismo significado que en el tipo anterior. Habitualmente las cadenas de bits se usan para almacenar flags (banderas) para el control de algún proceso.

- **Datos de fecha y hora.**

    - DATETIME: Resulta de la unión de un tipo DATE y de un tipo TIME. Está compuesto de 2 segmentos de 4 byte cada uno. `2:00 pm 1/1/97' se introducen entre comillas.

- **Otros tipos de datos.**

Los datos vistos en los puntos anteriores son soportados, por todos los sistemas gestores de bases de datos relacionales, muchos de ellos definen otros tipos de datos importantes de conocer en cada caso particular puesto que en muchos tipos de datos hay una restricción implícita que podemos aprovechar para colaborar en la integridad de los mismos. Los sistemas casi siempre incorporan la posibilidad de que el usuario puede definir sus propios tipos de datos.

Ejemplo: MONEY. Para almacenar unidades monetarias.Ocupa 8 byte de almacenamiento. SMALLMONEY.

Tipos de datos del sistema gestor de bases de datos relacionales SQL Server:

- *Cadenas de caracteres de longitud variable:* Casi todos los productos SQL incorporan los datos VARCHAR, que permiten que una columna almacene cadena de caracteres que varían de longitud de una fila a otra, hasta una longitud máxima. El estándar SQL Sólo especifica cadenas de longitud fija, que se rellenan por la derecha con caracteres en blanco adicionales.

- *Importes monetarios:* Muchos productos SQL incorporan un tipo MONEY, que se almacena generalmente como número decimal o en coma flotante. El tener un tipo monetario distinto permite al DBMS dar formato adecuadamente a los importes monetarios cuando son visualizados.

- *Fechas y horas:* Incorporar valores para fechas y horas es también habitual en los productos SQL, aunque los detalles varían ampliamente de un producto a otro. Generalmente se permiten variadas combinaciones de fechas, horas, instantes, intervalos de tiempo y aritmética de fecha y hora. El estándar SQL permite una elaborada especificación para los tipos de datos TIMESTAMP e INTERVAL, que incluye la gestión de zonas horarias y precisión en el tiempo (por ejemplo, decenas o centenas de segundos).

- *Texto extenso:* Varias bases de datos basadas en SQL permiten columnas que almacenan cadenas de texto extensas (típicamente de hasta 32000 o 65000 caracteres), y en algunos casos incluso más. El DBMS restringe generalmente el uso de estas columnas en consultas y búsquedas interactivas.

Text hasta 2147 millones de caracteres. **nText** hasta 2147 millones de caracteres unicode.

- *Flujo de bytes no estructurados:* Oracle y otros varios productos permiten almacenar y recuperar secuencias de byte de longitud variable sin estructurar. Las columnas que contienen estos datos se utilizan para almacenar imágenes de vídeo comprimidas, código ejecutable y otros tipos de datos sin estructurar.

- *Caracteres asiáticos:* DB2 permite cadenas de longitud fija y de longitud variable de caracteres de 16 bits utilizados para representar caracteres Kanji y otros caracteres asiáticos. Sin embargo, la búsqueda y ordenación de estos tipos GRAPHIC y VARGRAPHIC no está permitida.
- **CONSTANTES**

  En algunas sentencias SQL, un valor de datos numéricos, de caracteres o de fecha debe de ser expresado en forma textual. Por ejemplo, en esta sentencia INSERT, que añade un vendedor a la base de datos:

  INSERT INTO REPVENTAS (NUM_EMPL, NOMBRE, CUOTA, CONTRATO, VENTAS)

  VALUES (115, `Dennis Irving', 175000.000, `21−JUN−90', 0.00)

  El valor para cada columna en la fila recién insertada se especifica en la cláusula VALUES. Los valores de datos constantes también se utilizan en las expresiones, tal como por ejemplo en la sentencia SELECT:

  SELECT CIUDAD

  FROM OFICINA

```
WHERE OBJETIVO > (1.1 * VENTAS) + 10000.00
```

El estándar SQL ANSI/ISO especifica el formato de las constantes numéricas y de caracteres, o *literales*, que representan valores de datos específicos. Estos convenios son seguidos por la mayoría de las implementaciones SQL.

Constantes numéricas

Las constantes enteras y decimales (también denominadas *literales numéricos exactos*) se escriben como números decimales ordinarios dentro de las sentencias SQL, com un signo más o menos opcional delante.

```
21 −375 2000.00 +497500.8778
```

No se debe poner una coma entre los dígitos de una constante numérica, y no todos los dialectos de SQL permiten el signo más inicial, por lo que es mejor evitarlo. Para los datos monetarios, la mayoría de las implementaciones SQL utilizan simplemente constantes enteras o decimales, aunque algunas permiten que la constante sea especificada con un símbolo de moneda.

```
$0.75 $5000.00 $−567.89
```

Las constantes en coma flotante (también llamadas *literales numéricos aproximados*) se especifican utilizando la notación E hallada comúnmente en lenguajes de programación tales como C y FORTRAN. He aquí algunas constantes SQL en coma flotante válidas:

```
1.5E3 −3.14159E1 2.5E−7 0.783926E21
```

La E se lee por diez elevado a, de modo que la primera constante es 1.5 por diez elevado a tres, ó 1.500.

Constantes de cadena

El estándar especifica que las constantes SQL de caracteres han de ir encerradas entre comillas simples o dobles , como en los siguientes ejemplos:

```
`Jones, John J.' `New York' `Oeste'
```

```
Jones, John J. New York Oeste
```

SQL permite nombres de columnas que contengan blancos y otros caracteres especiales. Cuando estos caracteres aparecen como nombres en una sentencia SQL, deben ir entre corchetes. Por ejemplo, si la columna NOMBRE de la tabla REPVENTAS fuera en realidad NOMBRE COMPLETO en una base de datos SQL, esta sentencia SELECT sería válida.

```
SELECT [NOMBRE COMPLETO], VENTAS, CUOTA

FROM REPVENTAS

WHERE [NOMBRE COMPLETO] = `Jones, John J.'
```

Constantes de fecha y hora

En productos SQL que incluyen datos fecha/hora, los valores constantes para las fechas, horas e intervalos de tiempo se especifican como constantes de cadenas de caracteres.

SQL Server también permite datos fecha/hora y acepta una variedad de formatos diferentes para las constantes de fecha y hora. El DBMS acepta automáticamente todas las formas alternativas, y se pueden entremezclar si así se quiere. He aquí algunos ejemplos de constantes de fecha legales en SQL Server.

March 15 1990, Mar 15 1990, 3/15/1990, 3−15−90, 1990 MAR 15

Y he aquí algunas constantes de tiempos legales:

15:30:25 3:30:25 PM 3:30:25 pm 3 PM

Constantes simbólicas

Además de las constantes suministradas por el usuario, el lenguaje SQL incluye constantes simbólicas especiales que devuelven valores de datos mantenidos por el propio DBMS.

El estándar especifica solamente una única constante simbólica, pero la mayoría de los productos SQL proporcionan mucha más. Las constantes simbólicas que se encuentran habitualmente en las implementaciones SQL están listadas en la siguiente tabla:

Constante	**Descripción**
USER	El nombre de usuario bajo el cual se está accediendo actualmente a la base de datos (DB2, SQL/DS, Oracle, VAX, SQL, SQLBase y también especificado en el estándar ANSI/ISO.
CURRENT_TIMESTAMP	La fecha y hora actuales (DB2, SQL/DS).

Generalmente, una constante simbólica puede aparecer en una sentencia SQL en cualquier lugar en el que pudiera aparecer una constante ordinaria del mismo tipo de dato. El estándar adoptó las constantes simbólicas más útiles de las implementaciones SQL2 actuales tales como, CURRENT_TIMESTAMP, USER, y SESSION_USER .

Algunos productos SQL, incluyendo SQL Server, proporcionan acceso a valores del sistema mediante funciones internas en lugar de hacerlo con constantes simbólicas. La versión SQL Server es:

SELECT NOMBRE, CONTRATO

FROM REPVENTAS

WHERE CONTRATO > GETDATE ()

- **EXPRESIONES**

Las expresiones se utilizan en el lenguaje SQL para calcular valores que se recuperan de una base de datos y para calcular valores utilizados en la búsqueda en la base de datos. Por ejemplo, esta consulta calcula las ventas de cada oficina como orcentaje de su objetivo:

SELECT CIUDAD, OBJETIVO, VENTAS, (VENTAS/OBJETIVO) * 100

FROM OFICINAS

y esta consulta lista las ciudades de las oficinas cuyas ventas son superiores a $50.000 por encima del objetivo:

SELECT CIUDAD

FROM OFICINAS

WHERE VENTAS > Objetivo + 50000.00

El estándar SQL ANSI/ISO especifica cuatro operaciones aritméticas que pueden ser utilizadas en expresiones: suma (X+Y), resta (X−Y), multiplicación (X*Y) y división (X/Y). También se pueden utilizar paréntesis para formar expresiones más complicadas, como la siguiente:

(VENTAS * 1.05) − (OBJETIVO * 0.95)

Estrictamente hablando, los paréntesis no son necesarios en esta consulta, ya que el estándar especifica que la multiplicación y la división tienen una precedencia superior a la suma y la resta. Sin embargo, deberían utilizarse siempre los paréntesis para lograr que las expresiones no sean ambiguas, ya que diferentes dialectos de SQL pueden utilizar reglas diferentes. Además los paréntesis incrementan la legibilidad de la sentencia y hacen que las sentencias de SQL programado sean más fáciles de mantener.

El estándar también especifica conversión automática de tipos de datos desde números enteros a decimales, y desde números decimales a números en coma flotante, según se precise.

Por tanto, estos tipos de datos se pueden mezclar en una expresión numérica. Muchas implementaciones SQL incluyen otros operadores y permiten operaciones sobre datos de caracteres y de fechas. SQL, por ejemplo, incluye un operador concatenación de cadenas, escrito con el símbolo +. Si dos columnas llamadas NOMBRE y APELLIDO contiene los valores Jim y Jackson, entonces esta expresión:

(`Mr./Mrs. `+ NOMBRE + APELLIDO)

produce la cadena Mr./Mrs. Jim Jackson. Como ya se ha mencionado, también permite la suma, y resta de datos DATE, TIME y TIMESTAMP, para aquellas ocasiones en las que estas operaciones tengan sentido.

- **FUNCIONES INTERNAS**

Es posible que en alguna ocasión estemos interesados en localizar un registro que tome un valor en uno o varios campos que desconocemos, pero que posee por algún motivo una propiedad interesante para nosotros. Un ejemplo clarísimo puede ser nuestro deseo de conocer cual es el artículo del que menos unidades tenemos, por ejemplo, para realizar el pedido con más urgencia o del que más unidades hay en el almacén para no realizarlo, o el número medio de unidades para cada artículo presentes en almacén.

SQL dispone de lo que se denominan funciones integradas o funciones de conjuntos para resolver estas cuestiones. Las más habituales son:

- ♦ MAX (máximo)

- ♦ MIN (mínimo)
- ♦ AVG (media)
- ♦ SUM (suma)
- ♦ COUNT (total)

Ejemplo: Hallar el producto, mostrando su referencia y descripción del que menos unidades tenemos en almacén.

SELECT cod_art, des_art, MIN (cantidad) FROM artículos.

Funciones integradas en SQL.

Función	**Devuelve**
CAST (valor AS tipo_dato)	El valor, convertido al tipo de dato especificado (por ejemplo, una fecha convertida a una cadena de caracteres)
CONVERT (cadena USING conv)	Cadena convertida según especifique la función de conversión indicada
CURRENT_TIMESTAMP ()	Fecha y hora actuales.
LOWER (cadena)	Convierte una cadena de tipo varchar a minúsculas.
SUBSTRING (fuente,n , lon)	Extrae de la cadena *fuente* una subcadena , comenzando en el carácter n−ésimo, con una longitud lon
UPPER (cadena)	Convierte una cadena de tipo varchar a mayúsculas.
LEFT(cadena,lon)	Devuelve de la cadena una subcadena comenzando por la izquierda y con una longitud lon.
LEN(cadena)	Devuelve la longitud de la cadena.
LTRIM(cadena)	Quita los blancos de la izquierda en la cadena.
RTRIM(cadena)	Quita los blancos de la derecha en la cadena.
STR(cadena)	Devuelve la cadena alineada a la derecha.
RIGHT(cadena, nº elem)	Devuelve el número de elementos de la cadena que están a la derecha. RIGHT(CASA,2) = SA
ASCII(cadena)	Devuelve el nº ASCII correspondiente a la derecha

- **RESUMEN**

Este capítulo ha descrito los elementos básicos del lenguaje SQL. La estructura básica del SQL se puede resumir tal como se expresa seguidamente:

- ♦ El lenguaje SQL incluye unas treinta sentencias, cada una formada por un verbo y una o más cláusulas. Cada sentencia efectúa una única función específica.
- ♦ Las bases de datos basadas en SQL pueden almacenar varios tipos de datos entre los que se incluyen texto, enteros, números, números decimales, números de coma flotante y, generalmente, varios tipos más específicos del vendedor.

- ♦ Las sentencias SQL pueden incluir expresiones que combinen nombres de columnas, constantes, y funciones internas, utilizando operadores aritméticos y otros operadores específicos del vendedor.
- ♦ Las variaciones de los tipos de datos, constantes y funciones internas hacen que la portabilidad de las sentencias SQL sea más difícil de lo que en un principio pudiera parecer.
- ♦ Los valores NULL proporcionan un modo sistemático de gestionar los datos que faltan o son implicados dentro del lenguaje SQL.

TEMA V

CREANDO BASES DE DATOS Y TABLAS

- **BASES DE DATOS DE SISTEMAS**
- **BASES DE DATOS SQL SERVER**
- **PÁGINAS, EXTENSIONES Y FICHEROS**
- **ENTRAR EN ANALIZADOR DE CONSULTAS**
- **CREACION DE BASES DE DATOS DESDE EL ANALIZADOR DE CONSULTAS**
- **MODIFICACION DE BASES DE DATOS**
- **SELECCIÓN DE BASES DE DATOS ACTIVAÇ**
- **ELIMINACION DE BASES DE DATOS**
- **PROCEDIMIENTOS ALMACENADOS ADICIONALES**
- **DEFINICION DE TABLAS (CREATE TABLE)**
- **ELIMINACION DE UNA TABLA (DROP TABLE)**
- **MODIFICACION DE UNA DEFINICION DE TABLA (ALTER TABLE)**
- **INTRODUCCION DE DATOS EN LA BASE DE DATOS**

5.13.1. Insertar una fila

5.13.2. Insertar varias filas

5.13.3. Carga masiva

- **SUPRESION DE DATOS DE LA BASE DE DATOS**
- **MODIFICACION DE DATOS DE LA BASE DE DATOS**
- **RESUMEN**
- **EJERCICIOS**
- **BASES DE DATOS DE SISTEMAS**

SQL Server es un buen ejemplo de un sistema autoportante: controla las tablas de datos según las reglas que ellos mismos establecen a través de las tablas de datos. Cuando instala el software de SQL Server, automáticamente se crean cuatro bases de datos conocidas como bases de datos del sistema. Normalmente se evitará su modificación, puesto que un erro puede hacer que deje de funcionar el servidor. De todas formas es conveniente que se sepa cómo funcionan estas bases de datos, cuyos nombres son **master, Model, tempdb y msdb**.

Master

Es la encargada de guardar la información que se utiliza con la mayoría de las operaciones básicas SQL Server. En esta base de datos se encuentra la información sobre las cuentas de los usuarios y la configuración del sistema. Asimismo, tiene información sobre dónde localizar las bases de datos que crean los clientes. Si le pasa algo a esta base de datos, se encontrará metido en problemas. **Conviene**

que se hagan periódicamente copias de seguridad.

Model

La base de datos Model es la única de las cuatro que se puede modificar con ciertos sucesos. Siempre que se crea una base de datos, SQL Server inicia una copia de Model. Si quiere que algún elemento aparezca en todas las bases de datos de su servidor (por ejemplo, cierto tipo de datos que se utilizan en su empresa), tendrán que añadirlo a la base de datos model para que cada vez que se haga una copia de ella, se incluya dicho elemento.

No debe borrarse esta base de datos, puesto que SQL Server la usa com plantilla para todas las bases de datos nuevas, y si no la encuentra, no funcionará.

Tempdb

La base de datos tempdb es una parte de SQL Server. En ella se almacenan todas las tablas temporales que se generan durante la ejecución de los proceso. La base de datos tempdb se crea automáticamente cada vez que se inicia SQL Server y no hay ninguna razón por la que deba tocarse.

Msdb

Esta base de datos se utiliza para que el agente SQL Server guarde la información que necesita para procesar trabajos y alertas. Nunca deberá modificarse MSDB directamente. Debe utilizarse la interfaz del usuario para crear, modificar o eliminar cualquier objeto del Agente SQL Server.

- **BASES DE DATOS DE SQL SERVER**

Una de las formas que tiene SQL Server para proteger los datos es a través de registros de transacciones, bases de datos especiales que se utilizan para registrar la actividad de un cliente.

REGISTRO DE TRANSACCIONES

Cuando se crea una base de datos, SQL Server crea automáticamente un registro de transacciones para ella. Este registro es un archivo especial en el que se guarda toda la actividad relacionada con dicha base de datos. Siempre que un usuario agregue, elimine o modifique la información de la base de datos, el registro de transacciones tomará nota de la acción. Por defecto, estos registros tienen el mismo nombre que la base de datos, pero utilizan la extensión .ldf, aunque a la hora de crear la base de datos puede darle cualquier nombre.

El registro de transacciones proporciona cierto nivel de seguridad a los datos de SQL Server. En caso de fallo, es posible aplicar las transacciones que se encuentran en este archivo en otra copia de la base de datos. Por ejemplo, supongamos que se hace una copia de seguridad de la base de datos de los clientes el lunes a las diez de la mañana. Si el dispositivo encargado del mantenimiento de la base de datos falla el viernes a las cinco de la tarde, se pueden aplicar los registros de la semana a la copia de la base de datos que se hizo el lunes, con lo que se recuperará el trabajo de la semana.

Para protegerse de los fallos del hardware, el registro de transacciones se ha de guardar en un dispositivo fisico distinto de aquel en el que se encuentra la base de datos. Para obtener una protección máxima, conviene hacer un mirror (una copia) de ambos dispositivos. Además, tiene que

hacer un backup de la base de datos principal con cierta regularidad para que el tamaño del registro de transacciones no se dispare.

Para mantener el archivo de registro, SQL Server escribe las últimas acciones al principio del fichero. Cuando se modifican los datos, SQL Server da los siguientes pasos:

- El cambio se guarda en el registro de transacciones.
- Las páginas de datos modificadas se guardan en la memoria caché de almacenamiento.
- Se efectúan los cambios en las páginas que se encuentran en la memoria caché.
- En el proceso de comprobación se guardan los cambios en el disco.

El proceso de comprobación es un procedimiento interno de SQL Server que escribe todos los cambios de la memoria caché en el disco duro en el que se encuentra la base de datos. Si lo desea, se puede ejecutar manualmente CHECKPOINT en Transact_SQL para obligar a que se efectúe el proceso de comprobación.

- **PÁGINAS, EXTENSIONES Y FICHEROS**

Tres son los términos relacionados con SQL Server con los que hay que familiarizarse:

- ♦ **Páginas.**
- ♦ **Extensiones.**
- ♦ **Ficheros.**

Una página es la unidad más pequeña de almacenamiento de SQL Server. Tiene 8K, o lo que es lo mismo, 8.192 bytes. Las páginas de una base de datos únicamente guardarán información de un solo objeto (por ejemplo, las filas de una tabla), aunque lo normal es que éstos ocupen varias páginas. Cada una de ellas tiene 96 bytes reservados para guardar información, como el objeto al que pertencen y la cantidad de espacio libre que les queda. Como una fila de datos no puede ocupar varias páginas, las filas de las tablas SQL Server tienen una limitación de tamaño de 8K.

Una extensión la forman ocho páginas consecutivas, o lo que es lo mismo, 64K. Ahora que los objetos ocupan una página, es posible que las páginas de una misma extensión tengan objetos distintos. Cuando una tabla o un índice crece hasta el punto en que ocupa ocho páginas de extensiones distintas, se reorganizan para que se encuentren todas juntas. SQL Server agrega las extensiones que sean necesarias hasta que se alcance el límite de espacio del dispositivos de almacenamiento con el que trabaja.

SQL Server guarda la información directamente en ficheros (o archivos) del sistema operativo. Por defecto, las extensiones de archivos que se utilizan son los siguientes:

- ◊ **Archivos con la extensión .mdf son los ficheros de datos principales.**
- ◊ **Archivos con la extensión .ndf son los ficheros de datos secundarios, encargados de los datos adicionales definidos en el archivo principal.**
- ◊ **Ficheros con la extensión .ldf son los archivos de registro.**

Los archivos de SQL Server crecen automáticamente según se les van añadiendo datos. Al crecer una base de datos se puede especificar el incremento y el tamaño máximo de dichos ficheros.

- **ENTRAR EN ANALIZADOR DE CONSULTAS**

Para hacer pruebas con estas sentencias **SELECT** utilizaremos una herramienta, que viene dentro del paquete de SQL Server y que se llama *analizador de consultas*.

Para poner en marcha el *analizador de consultas* deberemos acudir al menú Inicio de Windows NT 4, seleccionar Programas, marcar SQL Server y dentro de él analizador de consultas.

Nos mostrará la pantalla de conexión al servidor de SQL Server. Seleccionaremos el nombre del ordenador Server donde se encuentra instalado el programa SQL Server. Si se encuentra en la maquina donde nos encontramos entonces se puede seleccionar *local*. Nos pedirá el nombre de usuario y contraseña con la que se hará la autentificación, usaremos las mismas que tiene Windows NT en su base de datos del Dominio.

Tras ello aparecerá la ventana principal de *analizador de consultas*. Se muestra una ventana donde pone *Query e* introduciremos la instrucción en lenguaje SQL (concretamente *Transact* SQl, que es el dialecto de Microsoft SQL Server).

Por defecto nos aparecerá conectada a *Master*, aunque podemos seleccionar cualquier otra base de datos de nuestro servidor.

Tenemos dos formas de seleccionar la base de datos sobre la que queremos lanzar consultas.

Una de ellas, la más fácil, es desplegar el combo que contiene las bases de datos disponibles y que se encuentran marcado como DB encima de las pestañas de consulta.

La segunda, la más curiosa, consiste en enviar una consulta USE con el nombre de la base de datos. Por ejemplo USE PUBS.

Tras escribir USE PUBS pulsaremos con el ratón la flecha de ejecución de la instrucción, o bien pulsaremos la combinación de teclas F5.

Veremos cómo automáticamente queda seleccionada la pestaña *Results* con la siguiente frase: *comandos completados con éxito* indica que se ha ejecutado correctamente.

En la ventana puede volver a escribirse otra instrucción SQL. Pueden escribirse varias instrucciones SQL de forma consecutiva y ejecutarlas simultáneamente al pulsar la tecla F5.

Como saber quién está conectado al sistema

SQL Server dispone de instrucciones de administración, las cuáles siguen el principio de Codd de que la configuración de la propia base de datos debe ser accesible mediante consultas SQL.

Dado que estas consultas implican conocer las tablas de sistema y que éstas pueden variar dependiendo de la versión de SQL Server, Microsoft proporciona una serie de procedimientos almacenados o *Stored Procedures* que realizan por nosotros estas consultas.

Los procedimientos almacenados que proporciona Microsoft comienzan siempre por *sp_*.

En este caso, para conocer los usuarios conectados a nuestro sistema, utilizaremos la consulta **sp_who**.

De hecho lo que estamos obteniendo son los procesos que están vivos dentro de SQL Server, no sólo los usuarios reales. La información que nos aporta es :

Id de proceso,

Estado del proceso,

Nombre del registro de usuario,

Nombre del Host,

Nombre de la base de datos que el usuario usa,

Comando SQL Server recien ejecutado.

Cómo obtener el listado de las tablas de la base de datos

Para obtener el listado de las tablas también disponemos de otra *stored procedure* que se llama **sp_tables**:

Cómo obtener la descripción de los campos de una tabla

Para obtener la descripción de los campos de una tabla utilizaremos el procedimiento almacenado llamado **sp_columns** seguido del nombre de la tabla que queremos ver. Por ejemplo:

Sp_columns *authors*

- **CREACION DE BASES DE DATOS DESDE ANALIZADOR DE CONSULTAS**

Para crear una base de datos, hay que estar conectado como administrador del sistema `SA' (o tener el permiso de utilizar CREATE DATABASE), y estar en la base de datos de sistema master.

El objeto DATABASE debe crearse en primer lugar. Una base de datos contiene todos los demás objetos.

− El catálogo de base de datos (18 tablas de sistema).

− Los objetos de usuario (tablas, valores por defecto, vistas, reglas, desencadenadores, procedimientos).

− Los índices, los tipos de datos, las restricciones de integridad.

− El registro de transacciones.

SINTAXIS :

CREATE DATABASE nombre [ON [PRIMARY]

[([NAME = nombrelógico,]

FILENAME = 'nombreFisico'

[, SIZE = tamaño]

[, MAXSIZE = { tamMax l UNLIMITED }]

[, FILEGROWTH = valorIncremento]) [,...]]

[LOG 0N { <archivo> }]

[FOR LOAD | FOR ATTACH]

NAME Nombre lógico del archivo.

FILE LENAME Ubicación y nombre físico del archivo.

SIZE Tamaño inicial del archivo en megabytes (MB) o kilobyies (KB). El tamaño predeterminado es de 1 megabytes.

MAXSIZE Tamaño máximo del archivo indicado en kilobytes o megabytes (por defecto megabytes). Si no se indica ningún valor el tamaño del archivo estará limitado por el espacio libre en disco.

UNLIMITED (Sin tamaño máximo) el limite es el espacio libre en disco.

FILEGROWTH Esta parámetro se utiliza cuando se precisa aumentar o disminuir el tamaño de los archivos de forma automática o manual. Este paso puede precisarse en porcentaje o de forma estática en kilobytes o megabytes.Las extensiones poseen un tamaño de 64 KB, este es pues el valor mínimo del paso de incremento que se puede indicar. El fichero irá aumentando mientras sea necesario y no sobrepase su valor máximo.

LOG 0N Ubicación del registro de transacciones.El registro de transacciones guarda las modificaciones aportadas a los datos.Por cada INSERT, UPDATE o DELETE, se hace una escritura en el registro antes de la escritura en la base. La validaci¢n de las transacciones se consigna también en el registro. Este registro sirve para la recuperaci¢n de los datos en caso de fallo.

FOR LOAD Para crear una base que será recuperada a partir de una copia de seguridad.

FOR ATTACH Para crear una base utilizando archivos ya creados. Esta petición es útil cuando la base se crea con más de 16 archivos.

Ejemplo:

Creación de la base de datos. Gescom (6 MB) con el registro de transacciones (2MB).

```
CREATE DATABASE GESCOM

ON PRIMARY

(name=gescom_data,

filename="c:\mssql7\data\g_data.mdf",

size=6MB,

maxsize=15MB,

filegrowth=1MB)

LOG ON
```

(name=gescom_log,

filename="c:\mssql7\glog.ldf",

size=2MB,

maxsize=15MB,

filegrowth=1)

- **MODIFICACION DE BASES DE DATOS**

Vista la instrucción **CREATE DATABASE**, no nos será difícil entender comparativamente cómo funciona **ALTER DATABASE**:

ALTER DATABASE nombre

MODIFY FILE

(name=nombrelógico

[,size=tamaño]

[, maxsize=tammax]

[FILEGROWTH=valor incremento])

ADD FILE

(name=nombrelógico

, FILENAME=nombrefisico

[,size=tamaño]

[, maxsize=tammax]

[FILEGROWTH=valor incremento])

Es posible modificar manualmente el tamaño, el tamaño máximo y la tasa de aumento de un archivo de datos con la instrucción ALTER DATABASE.

Aumentar el tamaño de un archivo existente:

ALTER DATABASE GESCOM

MODIFY FILE

(NAME=GESCOM_DATA,

SIZE=7MB)

Añadir archivos a la base de datos gescom

ALTER DATABASE GESCOM

ADD FILE

(NAME=GESCOM_DATA2,

FILENAME=C:\MSSQL7\DATA\G_DATA2.NDF,

SIZE=3)

- **SELECCION DE BASES DE DATOS ACTIVA**

La instrucción **USE** *NombreBaseDatos* sirve para seleccionar la base de datos activa desde la interfaz analizador de consultas.

Desde el momento en que una base de datos está activa, se pueden enviar consultas SQL contra las tablas de esa base de datos sin necesidad de especificar con el nombre y un punto a qué base de datos pertenece la tabla.

- **ELIMINACION DE BASES DE DATOS**

La instrucción **DROP DATABASE** *NombreBaseDatos* permite borrar una base de datos existente con las siguientes excepciones:

  - ♦ No se puede borrar una base de datos cuando está siendo utilizada por otro usuario.
  - ♦ No se puede borrar la base de datos *Model*, ya que ésta es la que se toma como patrón para la creación de nuevas bases de datos.
  - ♦ No se puede borrar la base de datos master, ya que se toma como base de datos principal de SQL Server.
  - ♦ No se puede borrar la base de datos tempdb, ya que es el lugar donde se almacenan temporalmente algunos datos de sistema necesarios para el funcionamiento del SQL Server.

- **PROCEDIMIENTOS ALMACENADOS ADICIONALES**

Para simplificar las consultas a las tablas de sistema, Microsoft ofrece la posibilidad de trabajar con unos procedimientos almacenados que a continuación vamos a detallar:

sp_helpdb

Proporciona información sobre todas las bases de datos o sobre la base de datos que suministremos como parámetro, mostrando el tamaño en MB, el usuario propietario de la base de datos, el número identificador de la base de datos, el status y la fecha de creación.

sp_spaceused

Muestra el espacio consumido y el espacio disponible de la base de datos en uso.

Sp_rename

Permite modificar el nombre de cualquier objeto definido como tablas, columnas, indices o tipos definidos por el usuario. No puede cambiar los nombres de la mayoría de objetos del sistema y, de los

que puede cambiar, únicamente tendrá acceso a aquéllos de los que sea propietario. La sintaxis es :

Sp_rename {nombre_objeto} [, nombre_nuevo] [, tipo_objeto]

El parámetro tipo_objeto permite especificar exactamente qué objeto se está renombrando en los casos en los que una columna de una tabla tiene el mismo nombre que un índice de otra tabla.

Sp_rename pedidos, pedidos2000

La información sobre las vistas y los procedimientos se actualiza automáticamente en la tabla del sistema sysobjects cuando se modifica su nombre. Cuando se renombra una restricción PRIMARY KEY o UNIQUE, se actualiza tods los índices asociados. Si se cambia el nombre de uñn indice asociado a una clave principal (PRIMARY KEY), Esta también se renombrará.

- **DEFINICION DE TABLAS (CREATE TABLE)**

La estructura más importante de una base de datos relacional es la tabla. En una base de datos multiusuario, las tablas principales son típicamente creadas una vez por el administrador de la base de datos y utilizadas luego día tras día.

Creación de una tabla (CREATE TABLE)

La sentencia define una nueva tabla en la base de datos y la prepara para aceptar datos. Las diferentes cláusulas de la sentencia especifican los elementos de la definición de la tabla.

En la práctica, la creación de una nueva tabla es relativamente sencilla. Cuando se ejecuta una sentencia CREATE TABLE, uno se convierte en el propietario de la tabla recién creada, a la cual se le da el nombre especificado en la sentencia. El nombre de la tabla debe ser un nombre SQL legal y no debe entrar en conflicto con el nombre de alguna otra tabla ya existente. La tabla recién creada está vacía, pero el DBMS la prepara para aceptar datos añadidos con la sentecia INSERT.

Definiciones de columnas

Las columnas de la tabla recién creada se definen en el cuerpo de la sentencia CREATE TABLE. Las definiciones de columnas aparecen en una lista separada por comas e incluida entre paréntesis. El orden de las definiciones de las columnas determina el orden de izquierda a derecha de las columnas en la tabla. Cada definición especifica:

- El *nombre de la columna*, que se utiliza para referirse a la columna en sentencias SQL. Cada columna de la tabla debe tener un nombre único, pero los nombres pueden ser iguales a los de las columnas de otras tablas. El número máximo de columnas por tabla es de 1024. La longitud máxima de una línea (registro, es decir la suma de las longitudes de las columnas) es de 8060 byte (sin contar los datos de texto o imagen).

CREATE TABLE nombre_tabla

(nombrecolumna tipcolumna [restricciones]

[, nombrecolumna .....]

[, restricciones])

[ON grupoarchivo]

nombretabla Puede ser de la forma base.propietario.tabla

nombrecolumna Nombre de la columna que debe ser única en la tabla. Puede haber 250 columnas por tabla.

Tipocolumna Tipo de sistema o tipo definido por el usuario.

Restricciones Reglas de integridad (UNIQUE, IDENTITY, REFERENCES, NULL, DEFAULT ....).

Grupoarchivo Grupo de archivos sobre el cual se creará la tabla.

Define la Tabla *OFICINAS* y sus columnas.

CREATE TABLE OFICINAS

(OFICINA INTEGER NOT NULL,

CIUDAD VARCHAR (15) NOT NULL,

REGION VARCHAR (10) NOT NULL,

DIR INTEGER,

OBJETIVO MONEY,

VENTAS MONEY NOT NULL)

Define la Tabla *PEDIDOS* y sus columnas.

CREATE TABLE PEDIDOS

(NUM_PEDIDO INTEGER NOT NULL,

FECHA_PEDIDO DATETIME NOT NULL,

CLIE INTEGER NOT NULL,

REP INTEGER,

FAB CHAR (3) NOT NULL,

PRODUCTO CHAR (5) NOT NULL,

CANT INTEGER NOT NULL,

IMPORTE MONEY NOT NULL)

El estándar especifica que una columna puede contener valores NULL a menos que específicamente se declare NOT NULL.

Valores por omisión (DEFAULT)

Define la Tabla *OFICINAS* con valores por omisión

CREATE TABLE OFICINAS

(OFICINA INTEGER NOT NULL,

CIUDAD VARCHAR (15) NOT NULL,

REGION VARCHAR (10) NOT NULL DEFAULT 'Este',

DIR INTEGER DEFAULT 106,

OBJETIVO MONEY DEFAULT NULL,

VENTAS MONEY NOT NULL DEFAULT 0.00)

Con esta definición de tabla, sólo es necesario especificar el número de oficinas y la ciudad cuando se inserte una nueva oficina. La región por omisión es el Este, el director de la oficina es Sam Clark (106) , las ventas son cero y el objetivo es NULL. El campo objetivo tomaría el valor por omisión NULL incluso sin la especificación DEFAULT NULL.

Definiciones de clave primaria y ajena.

Además de la definición de las columnas de una tabla, la sentencia CREATE TABLE identifica la clave primaria de la tabla y las relaciones de la tabla con otras tablas de la base de datos.

La cláusula PRIMARY KEY especifica la columna o columnas que forman la clave primaria de la tabla. Esta columna (o combinación de columnas) sirve como identificador único para cada fila de la tabla. El DBMS requiere automáticamente que el valor de clave primaria sea único en cada fila de la tabla. Además, la definición de columna para todas las columnas que forman la clave primaria debe especificar que la columna es NOT NULL.

La cláusula FOREIGN KEY especifica una clave ajena en la tabla y la relación que crea con otra tabla (padre) de la base de datos. La cláusula especifica:

- La columna o columnas que forman la clave ajena, todas las cuáles son columnas de la tabla que está siendo creada.
- La tabla que es referenciada por la clave ajena. Esta es la tabla padre en la relación; la tabla que está siendo definida es la hija.
- Un nombre opcional para la relación. El nombre no se utiliza en ninguna sentencia SQL, pero puede aparecer en mensajes de error y es necesaria si se desea poder suprimir la clave ajena posteriormente.
- Cómo debe tratar el DBMS un valor NULL en una o más columnas de la clave ajena, cuando la compare con las filas de la tabla padre.
- Una restricción de comprobación opcional que restrinja los datos de la tabla para que sus filas encuentren una condición de búsqueda especificada.

En general la defínción de las restricciones se hace por las instrucciones CREATE y ALTER TABLE o por la interfaz Enterprise Manager. Se guarda en las tablas de sitema **syscomments**, **sysreferences** y **sysconstraints**. Se pueden obtener datos sobre las restricciones por los procedimientos almacenados, **sp_help** nombretabla y **sp_helpconstraint** nombretabla.

[CONSTRAINT nombreRestricción]{ [{ PRIMARY KEY | UNIQUE } [CLUSTERED |

NONCLUSTERED] { (columna[,...n]) } [WITH FILLFACTOR = factorRelleno] [ON {grupoArchivos | DEFAULT}]] | FOREIGN KEY [(columna[,...n])] REFERENCES tablaReferencia [(columnaReferencia[,...n])] [NOT FOR REPLICATION] | CHECK [NOT FOR REPLICATION] (condicionesBúsqueda)}

CONSTRAINT

Es una palabra clave que indica el principio de la definición de una restricción PRIMARY KEY, UNIQUE, FOREIGN KEY o CHECK. Las restricciones son propiedades especiales que exigen la integridad de los datos y crean tipos especiales de índices para la tabla y sus columnas.

nombreRestricción

Es el nombre de una restricción. Los nombres de restricción deben ser únicos en una base de datos.

PRIMARY KEY

Es una restricción que exige la integridad de entidad para una o varias columnas dadas a través de un índice único. Sólo se puede crear una restricción PRIMARY KEY por cada tabla.

UNIQUE

Es una restricción que proporciona la integridad de entidad para una o varias columnas dadas a través de un índice único. Una tabla puede tener varias restricciones UNIQUE.

CLUSTERED | NONCLUSTERED

Son palabras clave que indican que se ha creado un índice agrupado o no agrupado para la restricción PRIMARY KEY o UNIQUE. De forma predeterminada, el valor de las restricciones PRIMARY KEY es CLUSTERED, y el de las restricciones UNIQUE es NONCLUSTERED.

Sólo se puede especificar CLUSTERED para una única restricción de una instrucción CREATE TABLE. Si especifica CLUSTERED para una restricción UNIQUE y especifica también una restricción PRIMARY KEY, el valor predeterminado de PRIMARY KEY es NONCLUSTERED.

[WITH FILLFACTOR = factorRelleno]

Especifica cuánto se debe llenar cada página de índice de SQL Server utilizada para almacenar los datos de índice. Los valores de factorRelleno especificados por el usuario pueden estar entre 1 y 100; el valor predeterminado es 0. Un factor de relleno pequeño crea el índice con más espacio disponible para las nuevas entradas de índice sin tener que asignar nuevo espacio.

FOREIGN KEY...REFERENCES

Es una restricción que proporciona integridad referencial para los datos de la columna o columnas. Las restricciones FOREIGN KEY requieren que cada valor de la columna exista en la columna de referencia correspondiente de la tabla a la que se hace referencia. Las restricciones FOREIGN KEY pueden hacer referencia sólo a columnas que sean restricciones PRIMARY KEY o UNIQUE en la tabla de referencia.

tablaRef

Es el nombre de la tabla a la que hace referencia la restricción FOREIGN KEY.

(columnaRef[,...n])

Es una columna o lista de columnas de la tabla a la que hace referencia la restricción FOREIGN KEY.

CHECK

Es una restricción que exige la integridad del dominio al limitar los valores posibles que se pueden escribir en una o varias columnas.

NOT FOR REPLICATION

Palabras clave que se utilizan para impedir que se exija la restricción CHECK durante el proceso de distribución utilizado por la duplicación. La cláusula NOT FOR REPLICATION significa que la restricción se fuerza en las modificaciones de los usuarios, pero no en el proceso de duplicación.

La restricción NOT FOR REPLICATION CHECK se aplica tanto a la imagen anterior como posterior de un registro actualizado para impedir que se agreguen o eliminen registros del intervalo duplicado. Se comprueban todos los borrados e inserciones; si éstos se encuentran en el intervalo duplicado, se rechazan.

Cuando esta restricción se utiliza con una columna de identidad, SQL Server permite que la tabla no tenga que reinicializar los valores de columna de identidad cuando un usuario de duplicación la actualiza.

expresiónLógica

Es una expresión lógica que devuelve TRUE o FALSE.

columna

Es una columna o lista de columnas, entre paréntesis, que se utiliza en las restricciones de tabla para indicar las columnas que se están utilizando en la definición de la restricción.

Es un marcador de posición que indica que el elemento anterior se puede repetir n veces.

He aquí una sentencia CREATE TABLE ampliada para la Tabla PEDIDOS, que incluye la definición de su clave primaria y de las tres claves ajenas que contiene:

CREATE TABLE PEDIDOS1

(NUM_PEDIDO INTEGER NOT NULL CONSTRAINT clave PRIMARY KEY,

FECHA_PEDIDO DATETIME NOT NULL,

CLIE VARCHAR(4) NOT NULL CONSTRAINT **PEDIDOPOR** FOREIGN KEY REFERENCES CLIENTES,

REP VARCHAR(3),

FAB VARCHAR (3) NOT NULL,

PRODUCTO VARCHAR (5) NOT NULL,

CANT INTEGER NOT NULL,

IMPORTE MONEY NOT NULL,

CONSTRAINT **TOMADOPOR** FOREIGN KEY (REP)

REFERENCES REPVENTAS (NUM_EMPL),

CONSTRAINT **ESPOR** FOREIGN KEY (FAB, PRODUCTO)

REFERENCES PRODUCTOS (ID_FAB,ID_PRODUCTO))

La figura muestra las tres relaciones creadas por esta sentencia y los nombres que se les asigna . En general es buena idea asignar un nombre de relación, ya que ayuda a clarificar la relación creada por la clave ajena. Por ejemplo, cada pedido fue remitido por el cliente cuyo número aparece en la columna CLIE de la Tabla PEDIDOS. La relación creada por esta columna ha recibido el nombre de PEDIDOPOR.

Cuando el DBMS procesa la sentencia CREATE TABLE, compara cada definición de clave ajena con la definición de la tabla referenciada. El DBMS se asegura que la clave ajena y la clave primaria de la tabla referenciada concuerdan en el número de columnas que contienen y en sus tipos de datos. La tabla referenciada debe estar ya definida en la base de datos para que esta comparación tenga éxito.

Tabla CLIENTES Tabla REPVENTAS Tabla PRODUCTOS

NUM_CLIE	EMPRESA
21003	Acme Mfg.

NUM_CLIE	NOMBRE
105	Bill Adams

ID_FAB	ID_PRODUCTO	DESCRIPCION
ACI	41004	Art. Tipo 4

PEDIDOPOR TOMADOPOR ESPOR

tabla PEDIDOS

NUM_PEDIDO	FECHA_PEDIDO	CLIE	REP	FAB	PRODUCTO	CANT	IMPORTE
112963	12/17/1989	2103	105	ACI	41004	28	$3,276.00

Ejemplo:

Supongamos que queremos crear una tabla llamada CATEGORÍA cuya clave principal es cod_cat.

Create table CATEGORÍA

(cod_cat varchar(2) NOT NULL,

Etiqueta varchar(30) NULL,

CONSTRAINT pk_categ PRIMARY KEY CLUSTERED (cod_cat))

Ejemplo:

Se pretende que las columnas Designación y Precio debe ser única en la tabla ARTICULOS:

Create table ARTICULOS

(NUM_ART varchar(2) PRIMARY KEY,

DESIGNACIÓN_art varchar(3),

precio integer,

constraint pk_desig UNIQUE NONCLUSTERED,

constraint pk_precio UNIQUE NONCLUSTERED)

La propiedad identity

Esta propiedad puede ser asignada a una columna numérica (entera, tinyint, smallint, int, decimal(p,0) o numeric(p,0)), en la creación o en la modificación de la tabla y permite que el sistema genere valores para esta columna. Los valores serán generados en la creación de la línea, sucesivamente y partiendo del valor inicial especifiado (por defecto 1) y aumentando o disminuyendo línea tras línea en un incremento (por defecto 1).

¡Sólo puede haber una columna IDENTITY por tabla¡

CREATE TABLE PEDIDOS

(num_pedido int identity (1000,1),

numero_cli int,

fecha_pdo datetime,

estado_pdo varchar(2))

En las creaciones de línea (INSERT), no se precisa un valor para NÚMERO_PEDIDO. La primera inserción asignará el num_pedido 1000, a la segunda el num_pedido 1001, etc ...

Ejemplo:

Crear una tabla llamada PRODUCTOS cuyo precio por producto debe ser superior a cero.

CREATE TABLE PRODUCTOS

(ID_FAB VARCHAR(3),

ID_PRODUCTO VARCHAR(5),

DESCRIPCION VARCHAR(20),

PRECIO MONEY,

EXISTENCIAS SMALLINT,

CONSTRAINT CLAVE_PRODUCTOS PRIMARY KEY (ID_FAB,ID_PRODUCTO),

CONSTRAINT PK_PRECIO CHECK (precio>0))

- **ELIMINACION DE UNA TABLA (DROP TABLE)**

Con el tiempo la estructura de una base de datos crecerá y se modificará. Nuevas tablas serán creadas para representar nuevas entidades, y algunas tablas antiguas ya no serán necesarias. Se puede eliminar de la base de datos una tabla que ya no es necesaria con la sentencia DROP TABLE.

El nombre de la tabla en la sentencia identifica la tabla a eliminar. Normalmente se eliminará una de las tablas propias del usuario y se utilizará un nombre de tabla no cualificado. Con el permiso adecuado, también se puede eliminar una tabla propiedad de otro usuario especificando un nombre de tabla cualificado.

DROP TABLE CLIENTES

Sam te da permiso para eliminar su tabla, llamada CUMPLEAÑOS.

DROP TABLE SAM.CUMPLEAÑOS

Cuando la sentencia DROP TABLE suprime una tabla de la base de datos, su definición y todos sus contenidos se pierden. No hay manera de recuperar los datos, y habría que utilizar una nueva sentencia CREATE TABLE para volver a crear la definición de la tabla. Debido a sus serias consecuencias, debe utilizarse la sentencia DROP TABLE con mucho cuidado.

La supresión de una tabla suprimirá los datos y los índices asociados. La supresión no será posible si la tabla es referenciada por una clave externa.

- **MODIFICACION DE UNA DEFINICION DE UNA TABLA (ALTER TABLE)**

Después de que una tabla ha sido utilizada durante algún tiempo, los usuarios suelen descubrir que desean almacenar información adicional con respecto a las entidades representadas en la tabla. En la base de datos ejemplo, por ejemplo, se podría desear:

- ♦ Añadir el nombre y el número de teléfono de una persona de contacto a cada fila de la Tabla CLIENTES, que se utilizará para contactar con los clientes.
- ♦ Hacer de la columna REGION en la Tabla OFICINAS una clave ajena de la tabla REGIONES recién creada.
- ♦ Suprimir la definición de clave ajena que enlaza la columna CLIE en la Tabla PEDIDOS con la Tabla CLIENTES, sustituyéndola con dos definiciones de clave ajena que enlacen la columna CLIE con las Tablas recién creadas INFO_CLIE e INFO_CONTA.

Cada uno de estos cambios, y algunos otros, pueden ser realizados con la sentencia ALTER TABLE. Al igual que con la sentencia DROP TABLE, ALTER TABLE se utilizará normalmente sobre tablas propias. Con el permiso adecuado, sin embargo, se puede especificar un nombre de tabla cualificado y alterar la definición de la tabla de otro usuario. La sentencia ALTER TABLE puede:

- ♦ Añadir una definición de columna a una tabla.
- ♦ Cambiar el valor por omisión de una columna.
- ♦ Añadir o eliminar una clave primaria para una tabla.
- ♦ Añadir o eliminar una nueva clave ajena para una tabla.
- ♦ Añadir o eliminar una restricción de unicidad para una tabla.

♦ Añadir o eliminar una restricción de comprobación para una tabla.

ALTER TABLE *nombre−de−tabla* ADD *definición−de−columna*

ALTER *nombre−de−columna* DEFAULT *valor*

DROP DEFAULT

DROP *nombre−de−columna* CASCADE

RESTRICT

ADD CONSTRAINT *definición−clave−primaria*

definición−clave−ajena

. .. .definición−unicidad

... .restricción−comprobación

DROP CONSTRAINT *nombre−restricción*

Cada una de las cláusulas de la sentencia ALTER TABLE puede aparecer sólo una vez en la sentencia. Se puede añadir una columna y definir una clave ajena en una única sentencia ALTER TABLE, pero deben utilizarse dos sentencias ALTER TABLE para añadir dos columnas.

Añadir una columna.

El uso más común de la sentencia ALTER TABLE es añadir una columna a una tabla existente. La cláusula de definición de columna en la sentencia ALTER TABLE es prácticamente idéntica a la de la sentencia CREATE TABLE y funciona del mismo modo. La nueva columna se añade al final de las definiciones de la columna de la tabla y aparece como la columna más a la derecha en consultas posteriores. El DBMS asume normalmente un valor NULL para la columna recién añadida en todas las filas existentes de la tabla . Si la columna se declara NOT NULL con un valor por omisión , el DBMS supone que el valor por omisión es el del tipo de datos de la columna. Observa que no puede declararse simplemente la nueva columna NOT NULL, ya que el DBMS asumiría valores NULL para la columna en las filas existentes, violando inmediatamente la restricción.

Añade un nombre de contacto y un número telefónico a la Tabla CLIENTES.

ALTER TABLE CLIENTES

ADD NOMBRE_CONTACTO VARCHAR (30)

ALTER TABLE CLIENTES

ADD TELEF_CONTACTO CHAR (10)

Añade una columna de nivel inventario mínimo a la Tabla PRODUCTOS.

ALTER TABLE PRODUCTOS

ADD CANT_MIN INTEGER NOT NULL WITH DEFAULT

Modificación de claves primaria y ajena.

El otro uso habitual para la sentencia ALTER TABLE es cambiar o añadir definiciones de clave primaria y clave ajena a una tabla.

Las cláusulas que añaden definiciones de claves primaria y ajena son exactamente las mismas que las de la sentencia CREATE TABLE y funcionan del mismo modo. Sólo se puede eliminar una clave ajena si la relación que crea tuvo asignado un nombre originalmente. Si la relación no tiene nombre, no hay manera de especificarla en la sentencia ALTER TABLE. En este caso no se puede suprimir la clave ajena a menos que se elimine y vuelva a crear la tabla utilizando el procedimiento descrito para suprimir una columna.

Haz de la columna REGION en la Tabla OFICINAS una clave ajena para la Tabla REGIONES recién creada, cuya clave primaria es el nombre región.

ALTER TABLE OFICINAS

ADD CONSTRAINT **ENREGION** FOREIGN KEY (REGION)

REFERENCES REGIONES (REGION)

Nombre de campo clave en la tabla REGIONES.

He aquí un ejemplo de una sentencia ALTER TABLE que modifica una clave primaria, la clave ajena correspondiente a la clave primaria original debe ser suprimida, puesto que ya no es una clave ajena para la tabla alterada:

- ♦ Borra la clave ajena OPERAEN

ALTER TABLE REPVENTAS

DROP CONSTRAINT OPERAEN

5.13. INTRODUCCION DE DATOS EN LA BASE DE DATOS

Típicamente, una nueva fila de datos se añade a una base de datos relacional cuando una nueva entidad representada por la fila aparece en el mundo exterior. Por ejemplo, en la base de datos ejemplo:

- ♦ Cuando se contrata un nuevo vendedor, debe añadirse una nueva fila a la Tabla PROVEEDORES para almacenar los datos de ese vendedor.
- ♦ Cuando un vendedor firma con un nuevo cliente, debe añadirse una nueva fila a la Tabla CLIENTES que representa al nuevo cliente.

En cada caso, la nueva fila se añade para mantener la base de datos como un modelo preciso del mundo real. La unidad de datos más pequeña que puede añadirse a una base de datos relacional es una fila. En general, un DBMS basado en SQL proporciona tres maneras de añadir nuevas filas de datos a una base de datos:

- ♦ Una sentencia INSERT de *una fila* añade una única nueva fila de datos a una tabla.
- ♦ Una sentencia INSERT *multifila* extrae filas de datos de otra parte de la base de datos y las añade a una tabla. Se utiliza habitualmente en procesamiento de fin de mes o de fin de año cuando filas antiguas de una tabla se trasladan a una tabla inactiva.
- ♦ Una utilidad de *carga masiva* añade datos a una tabla desde un archivo externo a la base de

datos. Se utiliza habitualmente para cargar inicialmente la base de datos o para incorporar datos transferidos desde otro sistema informático o recolectado desde muchas localizaciones.

5.13.1. Insertar una fila

La sentencia INSERT de una fila, añade una nueva fila a una tabla. La cláusula INTO especifica la tabla que recibe la nueva fila (la tabla destino) y la cláusula VALUE especifica los valores de los datos que contendrá la nueva fila. La lista de columnas indica qué valor va a qué columna de la nueva fila.

Supongamos que se acaba de contratar un nuevo vendedor, Henry Jacobsen, con los siguientes datos personales:

Nombre: Henry Jacobsen

Edad: 36

Número de empleado: 111

Título: Director de ventas

Oficina: Atlanta (número de oficina 13)

Fecha de contrato: 25 de julio de 1990

Cuota: Aún no asignada

Ventas anuales hasta la fecha: $0.00

He aquí la sentencia INSERT que añade al Sr. Jacobsen a la base de datos:

INSERT INTO *nombre−de tabla*

(*nombre−de−columna*)

,

VALUES (*constante*)

NULL

,

Incluir a Henry Jacobsen como nuevo vendedor.

INSERT INTO

REPVENTAS (NOMBRE, EDAD, NUM_EMPL, VENTAS, TITULO, CONTRATO, OFICINA_REP)

VALUES (`Henry Jacobsen', 36, 111, 0.00, `Dir Ventas', `25−JUL−90', 13)

1 fila insertada.

La sentencia INSERT construye una fila de datos que se corresponde con la estructura en columnas de la tabla, la rellena con los datos de la cláusula VALUES y luego añade la nueva fila a la tabla. Las filas de una tabla no están ordenadas, por lo que no existe la noción de insertar la fila al comienzo o al final o entre dos filas de la tabla. Después de ejecutar la sentencia INSERT, la nueva fila es simplemente una parte de la tabla. Una consulta posterior de la tabla REPVENTAS incluirá la nueva fila, pero puede aparecer en cualquier punto entre las filas del resultado de la consulta.

Supongamos que el Sr. Jacobsen recibe ahora su primer pedido, de InterCorp, un nuevo cliente que tiene asignado el número de cliente 2.126. El pedido es para 20 Widgets ACI−41004, por un precio total de $2.340, y le ha sido asignado el número de pedidos 113.069. He aquí las sentencias INSERT que añaden el nuevo cliente y el pedido a la base de datos:

Inserta un nuevo cliente y nuevo pedido para el Sr. Jacobsen.

INSERT INTO

CLIENTES (EMPRESA, NUM_CLIE, LIMITE_CREDITO, REP_CLIE)

VALUES (`InterCorp', 2126, 15000.00,111)

1 fila insertada.

INSERT INTO

PEDIDOS (IMPORTE, FAB, PRODUCTO, CANT, FECHA_PEDIDO, NUM_PEDIDO, CLIE, REP)

VALUES (2340.00, `ACI', `41004', 20,GETDATE(), 113069, 2126, 111)

1 fila insertada.

Como muestra este ejemplo, la sentencia INSERT puede ser larga si hay muchas columnas de datos, pero su formato sigue siendo muy sencillo. La segunda sentencia INSERT utiliza la constante de sistema GETDATE() en la cláusula VALUES, haciendo que se inserte la fecha actual como fecha de pedido.

En la práctica, sin embargo, los datos referentes a un nuevo cliente, un nuevo pedido o un nuevo vendedor son caso siempre añadidos a una base de datos mediante un programa de entrada orientado a formularios. Cuando la entrada de datos está completa, el programa inserta la nueva fila de datos utilizando SQL programado. Sin embargo, independientemente de que se utilice SQL programado o interactivo, la sentencia INSERT es la misma.

El propósito de la lista de columnas en la sentencia INSERT es hacer corresponder los valores de datos en la cláusula VALUES con las columnas que van a recibirlos. La lista de valores y la lista de columnas deben contener el mismo número de elementos y el tipo de dato de cada valor debe ser compatible con el tipo de dato de la columna correspondiente, o en caso contrario se producirá un error.

Inserción de valores NULL.

Cuando SQL inserta una nueva fila de datos en una tabla, automáticamente asigna un valor NULL a cualquier columna cuyo nombre falte en la lista de columnas de la sentencia INSERT. En esta

sentencia INSERT, que añade al Sr. Jacobsen a la Tabla REPVENTAS, las columnas CUOTA y DIRECTOR están omitidas:

INSERT INTO

REPVENTAS (NOMBRE, EDAD, NUM_EMPL, VENTAS, TITULO, CONTRATO, OFICINA_REP)

VALUES (`Henry Jacobsen', 36, 111, 0.00, `Dir Ventas', `25−JUL−90, 13)

Como resultado, la fila recién añadida tiene un valor NULL, en las columnas CUOTA y DIRECTOR. Puede hacer más explícita la asignación del valor NULL incluyendo las columnas en la lista y especificando la palabra clave NULL en los valores.

Inserción de todas las columnas.

Por conveniencia, SQL permite omitir la lista de columnas de la sentencia INSERT. Cuando se omite la lista de columnas, SQL genera automáticamente una lista formada por todas las columnas de la tabla, una secuencia de izquierda a derecha. Esta es la misma secuencia de columnas generadas por SQL cuando se utiliza una consulta SELECT. Utilizando esta forma abreviada, la sentencia INSERT anterior podría escribirse de la siguiente forma:

INSERT INTO REOVENTAS

VALUES (111, `Henry Jacobsen', 36, 13, `Dir Ventas', `25−JUL−90', NULL, NULL, 0.00)

5.13.2. Insertar varias filas.

La segunda forma de la sentencia INSERT, añade múltiples filas de datos a su tabla destino. En esta forma de la sentencia INSERT, los valores de datos para las nuevas filas no son especificados explícitamente dentro del texto de la sentencia. En su lugar, la fuente de las nuevas filas es una consulta de base de datos especificadas en al sentencia.

La adición de filas cuyos valores provienen de la propia base de datos puede parecer extraña al principio, pero es muy útil en algunas situaciones especiales. Por ejemplo, supongamos que se desea copiar el número de pedido, la fecha y el importe de todos los pedidos remitidos con anterioridad al 1 de enero de 1990, desde la Tabla PEDIDOS en otra Tabla llamada ANTPEDIDOS. La sentencia INSERT multifila proporciona un modo eficiente

Y compacto de copiar los datos:

Copia pedidos antiguos en la Tabla ANTPEDIDOS.

INSERT INTO ANTPEDIDOS (NUM_PEDIDO, FECHA_PEDIDO, IMPORTE)

SELECT NUM_PEDIDO, FECHA_PEDIDO, IMPORTE

FROM PEDIDOS

WHERE FECHA_PEDIDO < `01−ENE−90'

9 FILAS INSERTADAS.

La sentencia INSERT parece complicada, pero realmente es muy simple. La sentencia identifica la tabla que va a recibir las nuevas filas (ANTPEDIDOS) y las columnas que reciben los datos, lo mismo que la sentencia INSERT de una sola fila. El resto de la sentencia es una consulta que recupera datos de la Tabla PEDIDOS.

INSERT INTO *nombre−de tabla consulta*

(*nombre−de−columna*)

,

Conceptualmente SQL efectúa primero la consulta sobre la Tabla PEDIDOS y luego inserta los resultados, fila a fila en la Tabla ANTPEDIDOS.

He aquí otra situación donde se podría utilizar la sentencia INSERT multifila. Supongamos que se desea analizar los patrones de compra de un cliente examinando que clientes y qué vendedores son responsables de los grandes pedidos. Las consultas que se realizarían combinarían datos de la Tablas CLIENTES; REPVENTAS y PEDIDOS. Estas consultas de tres tablas de efectuarían bastante rápidamente en nuestra pequeña base de datos ejemplo, pero en una base de datos corporativa real con muchos miles de filas, llevaría mucho tiempo. En lugar de ejecutar muchas consultas de tres tablas largas, se podría crear una nueva tabla llamada GRANPEDIDOS para que contuviera los datos requeridos. Esta tabla estaría definida del modo siguiente:

IMPORTE Importe del pedido (procedente de PEDIDOS)

EMPRESA Nombre del cliente (procedente de CLIENTES)

NOMBRE Nombre del vendedor (procedente de REPVENTAS)

REND Importe superior o inferior a la cuota (calculado de REPVENTAS)

FAB Id del fabricante (procedente de PEDIDOS)

PRODUCTO Id del producto (Procedente de PEDIDOS)

CANT Cantidad pedido (procedente de PEDIDOS)

Una vez que se ha creado la Tabla GRANPEDIDOS, se puede utilizar esta sentencia INSERT multifila para rellenarla:

Carga datos en la Tabla GRANPEDIDOS para análisis:

INSERT INTO GRANPEDIDOS (IMPORTE, EMPRESA, NOMBRE, REND, PRODUCTO, FAB, CANT)

SELECT IMPORTE, EMPRESA, NOMBRE, REND, PRODUCTO, FAB, CANT

FROM PEDIDOS, CLIENTES, REPVENTAS

WHERE CLIE = NUM_CLIE

AND REP = NUM_EMPL

AND IMPORTE >15000.00

6 filas insertadas.

En una base de datos de grandes dimensiones, esta sentencia INSERT puede tardar un buen rato en ejecutarse, ya que implica una consulta de tres tablas. Cuando la sentencia se complete, los datos de la Tabla GRANPEDIDOS contendrán información duplicada de otras tablas. Además, la Tabla GRANPEDIDOS no se mantendrá, automáticamente actualizada cuando se añadan nuevos pedidos a la base de datos, por lo que sus datos pueden quedar rápidamente obsoletos. Cada uno de estos factores parece desventaja. Sin embargo, las consultas de análisis subsiguientes que utiliza una tabla como ésta pueden expresarse de forma muy sencilla.

El estándar SQL especifica varias restricciones lógicas sobre la consulta que aparece dentro de la sentencia INSERT multifila:

- La consulta no puede contener una cláusula ORDER BY.
- El resultado de la consulta debe contener el mismo número de columnas que hay en la lista de columnas de la sentencia INSERT y los tipos de los datos deben ser compatibles columna a columna.
- La consulta no puede ser la UNION de varias sentencias SELECT diferentes. Sólo puede especificarse una única sentencia SELECT.
- La tabla destino de la sentencia INSERT no puede aparecer en la cláusula FROM de la consulta o de ninguna subconsulta que ésta contenga.

5.13.3. Carga masiva

Los datos a insertar en una base de datos son con frecuencia extraídos de otro sistema automático o recolectados de otros lugares y almacenados en un archivo secuencial. Para cargar los datos en una tabla, se podría escribir un programa con un bucle que leyera cada registro del archivo y utilizara la sentencia INSERT de una fila para añadir la fila a la tabla. Sin embargo, el recargo de hacer que el DBMS ejecute repetidamente sentencias INSERT de una fila puede ser bastante alto. Si insertar una sola fila tarda medio segundo para una carga de sistema típica, éste es un rendimiento probablemente aceptable para un programa interactivo. Pero este rendimiento rápidamente pasa a ser inaceptable cuando se aplica a la tarea de cargar 50.000 filas de datos de una vez. En este caso, la carga de los datos requeriría más de 6 horas.

Por esta razón, todos los productos DBMS comerciales incluyen una capacidad de carga masiva que carga los datos desde un archivo a una tabla a alta velocidad.

5.14. SUPRESION DE DATOS DE LA BASE DE DATOS

Típicamente, una fila de datos se suprime de una base de datos cuando la entidad representada por la fila desaparece del mundo exterior.

- Cuando un cliente cancela un pedido, la correspondiente fila de la Tabla PEDIDOS debe ser suprimida.
- Cuando un vendedor abandona la empresa, la fila correspondiente de la Tabla REPVENTAS debe ser eliminada.
- Cuando una oficina de ventas se cierra, la fila correspondiente de la Tabla OFICINAS debe ser cancelada. Si los vendedores de la oficina son despedidos, sus filas también deberían ser suprimidas de la Tabla REPVENTAS, Si son reasignados, sus columnas OFICINA_REP deben ser actualizadas.

La sentencia DELETE

La sentencia DELETE elimina filas seleccionadas de datos de una única tabla. La cláusula FROM especifica la tabla destino que contiene las filas. La cláusula WHERE especifica qué filas de la tabla van a ser suprimidas.

DELETE FROM *nombre−de tabla*

WHERE *condición de búsqueda*

Elimina a Henry Jacobsen de la base de datos.

DELETE FROM REPVENTAS

WHERE NOMBRE = `Henry Jacobsem'

1 fila suprimida.

La cláusula WHERE de este ejemplo identifica una sola fila de la Tabla REPVENTAS, que SQL elimina de la tabla.

Elimina todos los pedidos de InterCorp (número de cliente 2.126).

DELETE FROM PEDIDOS

WHERE CLIE = 2126

2 filas suprimidas.

En este caso, la cláusula WHERE selecciona varias filas de la Tabla PEDIDOS y SQL elimina todas las filas seleccionadas de la tabla. Conceptualmente, SQL aplica la cláusula WHERE a cada una de las filas de la Tabla PEDIDOS, suprimiendo aquéllas para las cuales la condición de búsqueda produce un resultado TRUE y manteniendo aquéllas para las cuales la condición de búsqueda produce un resultado FALSE o NULL.

Suprime todos los pedidos remitidos antes del 15 de noviembre de 1989.

DELETE FROM PEDIDOS

WHERE FECHA_PEDIDO < `15−NOV−89'

5 filas suprimidas.

Suprime todas las filas correspondientes a los clientes atendidos por Bill Adams, Mary Jones o Dan Roberts (números de empleados 105, 109 y 101).

DELETE FROM REPVENTAS

WHERE CONTRATO < `01−JUL−88'

AND CUOTA IS NULL

0 filas suprimidas

Supresión de todas las filas

La cláusula WHERE en una sentencia DELETE es opcional, pero casi siempre está presente. Si se omite la cláusula WHERE de una sentencia DELETE, se suprimen todas las filas de la tabla destino, como en este ejemplo:

Suprime *todos* los pedidos.

DELETE FROM PEDIDOS

30 filas suprimidas.

Aunque esta sentencia DELETE produce una tabla vacía, no borra la Tabla PEDIDOS de la base de datos. La definición de la Tabla PEDIDOS y sus columnas siguen estando almacenadas en la base de datos. La tabla aún existe y nuevas filas pueden ser insertadas en la Tabla PEDIDOS con la sentencia INSERT. Para eliminar la definición de la tabla de la base de datos, debe utilizarse la sentencia DROP TABLE.

Debido al daño potencial que puede producir una sentencia DELETE como ésta, es importante especificar siempre una condición de búsqueda y tener cuidado de que se seleccionan realmente filas que se desean. Cuando se utiliza SQL interactivo, es buena idea utilizar primero la cláusula WHERE en una sentencia SELECT para visualizar las filas seleccionadas, asegurarse de que son las que realmente se desea suprimir y sólo entonces utilizar la cláusula WHERE en una sentencia DELETE.

DELETE con subconsulta.

Las sentencias DELETE con condiciones de búsqueda simples, como las de los ejemplos anteriores, seleccionan las filas a suprimir basándose únicamente en los propios contenidos de las filas. A veces, la selección de las filas debe efectuarse en base a datos contenidos en otras tablas.

Halla los pedidos aceptados por Sue Smith.

SELECT NUM_PEDIDO, IMPORTE,

FROM PEDIDOS, REPVENTAS

WHERE REP = NUM_EMPL

AND NOMBRE = `Sue Smith'

NUM_PEDIDO IMPORTE

112979 $15,000.00

113065 $ 2,130.00

112993 $ 1,896.00

113048 $ 3,750.00

Pero no se puede utilizar una composición en una sentencia DELETE. La sentencia DELETE paralela es ilegal:

DELETE FROM PEDIDOS, REPVENTAS

WHERE REP = NUM_EMPL

AND NOMBRE = `Sue Smith'

Error: Más de una tabla especificada en la cláusula FROM.

El modo de manejar la petición es con una de las condiciones de búsqueda subconsulta. He aquí una forma válida de la sentencia DELETE que realiza la petición:

Suprime los pedidos aceptados por Sue Smith.

DELETE FROM PEDIDOS

WHERE REP = (SELECT NUM_EMPL

FROM REPVENTAS

WHERE NOMBRE = `Sue Smith')

4 filas suprimidas.

La consulta halla el número de empleado de Sue Smith y la cláusula WHERE selecciona entonces los pedidos tratados por ese número de empleado.

5.15. MODIFICACION DE DATOS DE LA BASE DE DATOS

Típicamente, los valores de los datos almacenados en una base de datos se modifican cuando se producen cambios correspondientes en el mundo exterior. Por ejemplo, en la base de datos ejemplo:

- Cuando un cliente llama para modificar la cantidad de un pedido, la columna CANT de la fila apropiada en la Tabla PEDIDOS debe ser modificada.
- Cuando un director se traslada de una oficina a otra, la columna DIR en la Tabla OFICINAS y la columna OFICINA_REP en la Tabla REPVENTAS deben ser modificadas para que reflejen la nueva asignación.

La sentencia UPDATE

La sentencia UPDATE modifica los valores de una o más columnas en las filas seleccionadas de una tabla única. La tabla destino a actualizar se indica en la sentencia y es necesario disponer de permiso para actualizar la tabla así como cada una de las columnas individuales que serán modificadas. La cláusula WHERE selecciona las filas de la tabla a modificar. La cláusula SET especifica qué columnas se van a actualizar y calcula los nuevos valores.

UPDATE *nombre−de−tabla* SET *nombre−de−columna = expresión*

,

WHERE *condición−de−búsqueda*

Eleva el límite de crédito de la empresa Acme Manufacturing a $60.0000 y la reasigna a Mary Jones (múmero de empleado 109(.

UPDATE CLIENTES

SET LIMITE_CREDITO = 60000.00, REP_CLIE = 109

WHERE EMPRESA = `Acme Mfg.'

1 fila actualizada.

En este ejemplo, la cláusula WHERE identifica una sola fila de la Tabla CLIENTES y la cláusula SET asigna nuevos valores a dos de las columnas de esta fila.

Transfiere todos los vendedores de la oficina de Chicago (número 12) a la oficina de New York (número 11) y rebaja sus cuotas un 10 por 100.

UPDATE REPVENTAS

SET OFICINA_REP = 11, CUOTA = 0.9 * CUOTA

WHERE OFICINA_REP = 12

1 fila actualizada.

En este caso, la cláusula WHERE selecciona varias filas de la Tabla REPVENTAS y el valor de las columnas OFICINA_REP y CUOTA se modifica en todas ellas. Conceptualmente, SQL procesa la sentencia UPDATE al recorrer la Tabla REPVENTAS fila a fila, actualizando aquellas filas para las cuales la condición de búsqueda produce un resultado TRUE y omitiendo aquéllas para las cuales la condición de búsqueda produce un resultado FALSE o NULL.

La cláusula SET en la sentencia UPDATE es una lista de asignaciones separadas por comas. Cada asignación identifica una columna destino a actualizar y especifica cómo calcular el nuevo valor para la columna destino. Cada columna destino debería aparecer solamente una vez en la lista; no debería haber dos asignaciones para la misma columna destino.

La expresión en cada asignación puede ser cualquier expresión SQL válida que genere un valor tipo de dato apropiado para la columna destino. La expresión se debe poder calcular con los valores de la fila que actualmente está en actualización en la tabla destino. No pueden incluirse funciones de columna ni subconsulta.

Actualización de todas las filas.

La cláusula WHERE en la sentencia UPDATE es opcional. Si se omite la cláusula WHERE, entonces se actualizan todas las filas de la tabla destino, como en este ejemplo:

Eleva todas las cuota un 5%.

UPDATE REPVENTAS

SET CUOTA = 1.05 * CUOTA

10 filas actualizadas.

UPDATE con subconsulta.

Al igual que con la sentencia DELETE, las subconsultas pueden jugar un papel importante en la sentencia UPDATE ya que permiten seleccionar las filas a actualizar en base a información contenida en otras tablas. He aquí varios ejemplos de sentencias UPDATE que utilizan subconsultas:

Eleva en $5.000 el límite de crédito de cualquier cliente que haya remitido una orden de más de $25.000.

UPDATE CLIENTES

SET LIMITE_CREDITO = LIMITE_CREDITO + 5000.00

WHERE NUM_CLIE IN (SELECT DISTINCT CLIE

FROM PEDIDOS

WHERE IMPORTE > 25000.00)

4 filas actualizadas.

Reasigna todos los clientes atendidos por vendedores cuyas ventas son menores al 80 por 100 de sus cuotas.

UPDATE CLIENTES

SET REP_CLIE = 105

WHERE REP_CLIE IN (SELECT NUM_EMPLP

FROM REPVENTAS

WHERE VENTAS < (0.8 * CUOTA))

2 filas actualizadas.

Las subconsultas en la cláusula WHERE de la sentencia UPDATE pueden anidarse a cualquier nivel y pueden contener referencias externas a la tabla destino de la sentencia UPDATE.

La tabla destino no puede aparecer en la cláusula FROM de ninguna subconsulta a ningún nivel de anidación.

Cualquier referencia a la tabla destino en las subconsultas son por tanto referencias externas a la fila de la tabla destino que actualmente está siendo comprobada por las cláusulas WHERE de la sentencia UPDATE.

5.16. RESUMEN

- La sentencia INSERT de una fila añade una fila de datos a una tabla. Los valores para la nueva fila se especifican en la sentencia como constantes.
- La sentencia INSERT multifila añade cero o más filas a una tabla. Los valores para las nuevas filas provienen de una consulta, especificada como parte de la sentencia INSERT.
- La sentencia DELETE suprime cero o más filas de datos de una tabla. Las filas a suprimir son especificadas mediante una condición de búsqueda.

- La sentencia UPDATE modifica los valores de una o más columnas en cero o más filas de una tabla. Las filas a actualizar son especificadas mediante una condición de búsqueda.
- A diferencia de la sentencia SELECT, que puede operar sobre múltiples tablas, las sentencias INSERT, DELETE y UPDATE funcionan solamente sobre una única tabla cada vez.

5.17 EJERCICIOS

- **EJERCICIO:**

Se trata de una sencilla base de datos relacional para una pequeña empresa de distribución. La base de datos almacena la información necesaria para implementar una pequeña aplicación de procesamiento de pedidos.

- los *clientes* que compran los productos de la empresa;
- los *pedidos* remitidos por esos clientes;
- los *vendedores* que venden los productos a los clientes; y
- las *oficinas* de ventas donde trabajan los vendedores

Hay una tabla separada de datos para cada clase diferente de entidad. Las peticiones de base de datos que se hace utilizando el lenguaje SQL se corresponde con actividades del mundo real, tales como emisión, cancelación y cambio de pedidos por parte de los clientes, contratación y despido de vendedores, etc.

- Crear la base de datos.
- Crear las 4 tablas.
- Insertar datos en todas las tablas.

Tabla OFICINAS

OFICINA	CIUDAD	REGION	OBJETIVO	VENTAS
22	Denver	Oeste	$300,000.00	$186,042.00
11	New York	Este	$575,000.00	$692,637.00
12	Chicago	Este	$800,000.00	$735,042.00
13	Atlanta	Este	$350,000.00	$367.911.00
21	Los Angeles	Oeste	$725,000.00	$835,915.00

Tabla PEDIDOS

NUM_PEDIDO	FECHA_PEDIDO	CLIE	REP	FAB	PRODUCTO	CANT	IMPORTE
112961	17-DIC-89	2117	106	REI	A244L	7	$31,500.00
113012	1-ENE-90	2111	105	ACI	41003	35	$3,745.00
112989	03-ENE-90	2101	106	FEA	114	6	$1,458.00
113051	10-FEB-90	2118	108	QSA	XK47	4	$1,420.00
112968	12-OCT-89	2102	101	ACI	41004	34	$3,978.00
113036	30-ENE-90	2107	110	ACI	4100Z	9	$22,500.00
113045	02-FEB-90	2112	108	REI	A244L	10	$45,000.00
112963	17-DIC-89	2103	105	ACI	41004	28	$3,276.00
113013	14-ENE-90	2118	108	BIC	41003	1	$652.00
113058	23-FEB-90	2108	109	FEA	112	10	$1,480.00
112997	08-ENE-90	2124	107	BIC	41003	1	$652.00

112983	27-DIC-89	2103	105	ACI	41004	6	$702.00
113024	20-ENE-90	2114	108	QSA	XK47	20	$7,100.00
113062	24-FEB-90	2124	107	FEA	114	10	$2,430.00
112979	12-OCT-89	2114	102	ACI	4100Z	6	$15,000.00
113027	22-ENE-90	2103	105	ACI	41002	54	$4,104.00
113007	08-ENE-90	2112	108	IMM	773C	3	$2,925.00
113069	02-MAR-90	2109	107	IMM	775C	22	$1,350.00
113034	29-ENE-90	2107	110	REI	A245C	8	$632.00
112992	04-NOV-89	2118	108	ACI	41002	10	$760.00
112975	12-OCT-89	2111	103	REI	A244G	6	$2,100.00
113055	15-FEB-90	2108	101	ACI	4100X	6	$150,.00
113048	10-FEB-90	2120	102	IMM	779C	2	$3,750.00
112993	04-ENE-89	2106	102	REI	A245C	24	$1,896.00
113065	27-FEB-90	2106	102	QSA	XK47	6	$2,130.00
113003	25-ENE-90	2108	109	IMM	779C	3	$5,625.00
113049	10-FEB-90	2118	108	QSA	XK47	2	$776.00
112987	31-DIC-89	2103	105	ACI	4100Y	11	$27,500.00
113057	18-FEB-90	2111	103	ACI	4100X	24	$600.00
113042	02-FEB-90	2113	101	REI	A244R	5	$22,500.00

Tabla CLIENTES

NUM_CLIE	EMPRESA	REP_CLIE	LIMITE_CREDITO
2111	JCP inc.	103	$50,000.00
2102	First Corp.	101	$65,000.00
2103	Acme Mfg.	105	$50,000.00
2123	Carter & Sons	102	$40,000.00
2107	Ace International	110	$35,000.00
2115	Smithson Corp.	101	$20,000.00
2101	Jones Mfg.	106	$65,000.00
2112	Zetacorp	108	$50,000.00
2121	QMA Assoc.	103	$45,000.00
2114	Orion Corp.	102	$20,000.00
2124	Peter Brothers	107	$40,000.00
2108	Holm & Landis	109	$55,000.00
2117	J.P. Sinclair	106	$35,000.00
2122	Three-Way Lines	105	$30,000.00
2120	Rico Enterprises	102	$50,000.00
2106	Fred Lewis Corp.	102	$65,000.00
2119	Solomon Inc.	109	$25,000.00
2118	Midwest Systems	108	$60,000.00
2113	Ian & Schmidt	104	$20,000.00
2109	Chen Associates	107	$25,000.00

2105	AAA Investments	101	$45,000.00

Tabla REPVENTAS

NUM_EMPL	NOMBRE	EDAD	OFICINA_REP	TITULO	CONTRATO	DIRECTOR	CUOTA	VEN
105	Bill Adams	37	13	Rep. ventas	1−ENE−88	104	$350,000.00	$367
109	Mary Jones	31	11	Rep. ventas	12−OCT−9	106	$300,000.00	$392
102	Sue Smith	48	21	Rep. ventas	10−DIC−86	108	$350,000.00	$474
106	Sam Clark	52	11	Vp ventas	14−JUN−88	NULL	$275,000.00	$299
104	Bob Smith	33	12	Dir ventas	19−MAY−87	106	$200,000.00	$142
101	Dan Roberts	45	12	Rep. Ventas	20−OCT−86	104	$300,000.00	$305
110	Tom Snyder	41	NULL	Rep. Ventas	13−ENE−90	101	NULL	$75,9
108	Larry Fitch	62	21	Dir ventas	12−OCT−89	106	$350,000.00	$361
103	Paul Cruz	29	12	Rep. Ventas	01−MAR−87	104	$275,000.00	$286
107	Nancy Angelli	49	22	Rep. Ventas	14−NOV−87	108	$300,000.00	$186

TABLA PRODUCTOS

ID_FAB	ID_PRODUCTO	DESCRIPCIÓN	PRECIO	EXISTENCIAS
REI	2A45C	UNION TRINQUETE	79	210
ACI	4100Y	DESMONTADOR	2750	25
QSA	XK47	REDUCTOR	355	38
BIC	41672	PLACA	180	0
IMM	779C	ABRAZADERA 90	1875	9
ACI	4103	ARTICULO TIPO3	107	207
ACI	41004	ARTICULO TIPO 4	117	139
BIC	41003	TIRADOR	652	3
IMM	887P	PERNO ABRAZADERA	250	24
QSA	XK48	REDUCTOR	134	203

REI	2A44L	BISAGRA IZQUIDA	4500	12
FEA	112	CUBIERTA	148	115
IMM	887H	SOPORTE ABRAZADERA	54	223
BIC	41089	RETEN	225	78
ACI	41001	ARTICULO TIPO 1	55	277
IMM	775C	ABRAZADERA 500	1425	5
ACI	410Z	MONTADOR	2500	28
QSA	XK48A	REDUCTOR	117	37
ACI	41002	ARTICULO TIPO 2	76	167
REI	2A44R	BISAGRA DERECH	4500	12
IMM	773C	ABRAZADERA 300	975	28
ACI	4100X	AJUSTADOR	25	37
FEA	114	BANCADA MOTOR	243	15
IMM	887X	RETÉN ABRAZADERA	475	32
REI	2A44G	PASADOR BISAGRA	350	14

TEMA VI

CONSULTAS SIMPLES

6.1. CLAUSULA SELECT

6.2. CLAUSULA FROM

6.3. RESULTADOS DE CONSULTAS

6.4. CLAUSULA WHERE

6.5. COMNDICIONES DE BUSQUEDA

- **Test de comparación**
- **Test de rango**
- **Test de pertenencia a conjunto**
- **Test de correspondencia con patrón**
- **Test de valor nulo**
- **Condiciones de búsqueda compuestas**

6.6. COLUMNAS CALCULADAS

6.7. SELECCIÓN DE TODAS LAS COLUMNAS

6.8. FILAS DUPLICADAS (DISTINCT)

6.9. ORDENACION DE LOS RESULTADOS DE UNA CONSULTA

6.10. REGLAS PARA PROCESAMIENTO DE CONSULTAS DE TABLA UNICA

6.11. COMBINACION DE LOS RESULTADOS DE UNA CONSULTA

6.12. CONSULTAS DE RESUMEN, HAVING, GROUP BY

- **SINTAXIS DE LA ORDEN SELECT**
- **RESUMEN**
- **EJERCICIOS**
- **CLAUSULA SELECT**

La cláusula SELECT que empieza cada sentencia SELECT especifica los ítems de datos a recuperar por la consulta. Los ítems se especifican generalmente mediante una *lista de selección*, una lista de *ítems de selección* separados por comas. Cada ítem de selección de la lista genera una única columna de resultados de consulta, en orden de izquierda a derecha. Un ítem de selección puede ser:

SELECT item1, item2,...

- Un *nombre de columna*, identificando una columna de la tabla designada en la cláusula FROM. Cuando un nombre de columna aparece como ítem de selección, SQL simplemente toma el valor de esa columna de cada fila de la tabla de base de datos y lo coloca en la fila correspondiente de los resultados de la consulta.
- Una *constante*, especificando que el mismo valor constante va a aparecer en todas las filas de los resultados de la consulta.
- Una *expresión* SQL, indicando que SQL debe calcular el valor a colocar en los resultados, según el estilo especificado por la expresión.

- **CLAUSULA FROM**

La cláusula FROM consta de la palabra clave FROM, seguida de una lista de especificaciones de tablas separadas por comas. Cada especificación de tabla identifica una tabla que contiene datos a recuperar por la consulta. Estas tablas se denominan *tablas fuente* de la consulta (y de la sentencia SELECT), ya que constituyen la fuente de todos los datos que aparecen en los resultados. Todas las consultas de este capítulo tienen una única tabla fuente, y todas las cláusulas FROM contienen un solo nombre de tabla.

- **RESULTADOS DE CONSULTAS**

El resultado de una consulta SQL es siempre una tabla de datos, semejante a las tablas de la base de datos. Si se escribe una sentencia SELECT utilizando SQL interactivo, el DBMS visualizará los resultados de la consulta en forma tabular sobre la pantalla de la computadora.

Generalmente los resultados de la consulta formarán una tabla con varias columnas y varias filas. Por ejemplo, esta consulta produce una tabla de tres columnas (ya que pide tres ítems de datos) y diez filas (ya que hay diez vendedores):

Lista los nombres, oficinas y fecha de contrato de todos los vendedores.

SELECT NOMBRE, OFICINA_REP, CONTRATO

FROM REPVENTAS

NOMBRE OFICINA_REP CONTRATO

Bill Adams 13 12−FEB−88

Mary Jones 11 12−OCT−89

Sue Smith 21 10−DIC−86

Sam Clark 11 14−JUN−88

Bob Smith 12 19−MAY−87

Dan Roberts 12 20−OCT−86

Tom Snyder NULL 13−ENE−90

Larry Fitch 21 12−OCT−89

Paul Cruz 12 01−MAR−87

Nancy Angelli 22 14−NOV−88

Las consultas SQL más sencillas solicitan columnas de datos de una única tabla en la base de datos.

Lista de la población, región y ventas de cada oficina.

SELECT CIUDAD, REGION, VENTAS

FROM OFICINAS

CIUDAD REGION VENTAS

Denver Oeste $186,042.00

New York Este $692,637.00

Chicago Este $735,042.00

Atlanta Este $367,911.00

Los Angeles Oeste $835,915.00

La sentencio SELECT para consultas sencillas como ésta sólo incluye las dos cláusulas imprescindibles. Las cláusula SELECT designa a las columnas solicitadas; la cláusula FROM designa a la tabla que las contiene.

Conceptualmente, SQL procesa la consulta recorriendo la tabla nominada en la cláusula FROM, una fila cada vez. Por cada fila, SQL toma los valores de las columnas solicitadas en la lista de selección y produce una única fila de resultados. Los resultados contienen por tanto una fila de datos por cada fila de la tabla.

Tabla OFICINAS

OFICINA	CIUDAD	REGION	OBJETIVO	VENTAS
22	Denver	Oeste	$300,000.00	$186,042.00
11	New York	Este	$575,000.00	$692,637.00
12	Chicago	Este	$800,000.00	$735,042.00
13	Atlanta	Este	$350,000.00	$350,000.00
21	Los Angeles	Oeste	$725,000.00	$835,915.00

Resultados de la consulta

CIUDAD	REGION	VENTAS
Denver	Oeste	$186,042.00
New York	Este	$692,637.00
Chicago	Este	$735,042.00
Atlanta	Este	$350,000.00
Los Angeles	Oeste	$835,915.00

- **CLAUSULA WHERE**

Las consultas SQL que recuperan todas las filas de una tabla son útiles para inspección y elaboración de informes sobre la base de datos, pero para poco más. Generalmente se deseará seleccionar solamente parte de las filas de una tabla, y sólo incluirán esas filas en los resultados. La cláusula WHERE se emplea para especificar las filas que se desean recuperar. He aquí algunos ejemplos de consultas simples que utilizan la cláusula WHERE:

Muestra las oficinas en donde las ventas exceden del objetivo.

SELECT CIUDAD, VENTAS, OBJETIVO

FROM OFICINAS

WHERE VENTAS > OBJETIVO

CIUDAD VENTAS OBJETIVO

New York $692,637.00 $575,000.00

Atlanta $367,911.00 $350,000.00

Los Angeles $835,915.00 $725,000.00

Muestra los empleados dirigidos por Bob Smith (empleado 104).

SELECT NOMBRE, VENTAS

FROM REPVENTAS

WHERE DIRECTOR = 104

NOMBRE VENTAS

Bill Adams $367,911.00

Dan Roberts $305,673.00

Paul Cruz $286,775.00

La cláusula WHERE consta de la palabra clave WHERE seguida de una condición de búsqueda que especifica las filas a recuperar. En la consulta anterior, por ejemplo, la condición de búsqueda es DIRECTOR = 104. Conceptualmente, SQL recorre cada fila de la Tabla REPVENTAS, una a una, y aplica la condición de búsqueda a la fila. Cuando aparece un nombre de columna en la condición de búsqueda a la fila (tal como la columna DIRECTOR en este ejemplo), SQL utiliza el valor de la columna en la fila actual. Por cada fila, la condición de búsqueda puede producir uno de los tres resultados.

- Si la condición de búsqueda es TRUE (cierta), la fila se incluye en los resultados de la consulta. Por ejemplo, la fila correspondiente a Bill Adams tiene el valor DIRECTOR correcto, y por tanto se incluye.
- Si la condición de búsqueda es FALSE (falsa), la fila se excluye de los resultados de la consulta.

NOMBRE	**DIRECTOR**				**NOMBRE**	**VENTAS**
Bill Adams	104				Bill Adams	$367,911.00
Mary Jones	106				Dan Roberts	$305,673.00
Sue Smith	108				Paul Cruz	$286,775.00
Sam Clark	NULL					
Bob Smith	106					
Dan Roberts	104					

TRUE

FALSE

Desconocido

- Si la condición de búsqueda tiene un valor NULL (desconocido), la fila se excluye de los resultados de la consulta. Por ejemplo, la fila correspondiente a Sam Clark tiene un valor NULL en la columna DIRECTOR, y por tanto se excluye.

Básicamente la condición de búsqueda actúa como un filtro para las filas de la tabla. Las filas que satisfacen la condición de búsqueda atraviesan el filtro y forman parte de los resultados de la consulta. Las filas que no satisfacen la condición de búsqueda son atrapadas por el filtro y quedan excluidas de los resultados de la consulta.

- **CONDICIONES DE BUSQUEDA**

SQL ofrece un rico conjunto de condiciones de búsqueda que permite especificar muchos tipos diferentes de consultas eficaz y naturalmente aquí se resumen cinco condiciones básicas de búsqueda

(llamadas *predicados* en el estándar ANSI/ISO) y posteriormente se describirán en otras secciones:

- *Test de comparación.* Compara el valor de una expresión con el valor de otra.
- *Test de rango.* Examina si el valor de una expresión cae dentro de un rango especificado de valores.
- *Test de pertenencia a conjunto.* Comprueba si el valor de una expresión se corresponde con uno de un conjunto de valores.
- *Test de correspondencia con patrón.* Comprueba si el valor de una columna que contiene datos de cadena de caracteres se corresponde a un patrón especificado.
- *Test de valor nulo.* Comprueba si una columna tiene un valor NULL (desconocido).

6.5.1. Test de comparación

La condición de búsqueda más utilizada en una consulta SQL es el test de comparación. En un test de comparación, SQL calcula y compara los valores de dos expresiones SQL por cada fila de datos. Las expresiones pueden ser tan simples como un nombre de columna o una constante, o pueden ser expresiones aritméticas más complejas. He aquí algunos ejemplos de tests de comparación típicos:

Halla los vendedores contratados antes de 1988.

SELECT NOMBRE

FROM REPVENTAS

WHERE CONTRATO < `01−ENE−88'

NOMBRE

Sue Smith

Bob Smith

Dan Roberts

Paul Cruz

Como operadores de comparación se pueden utilizar: =, <>, <, <=, >, >=.

La comparación de desigualdad se escribe como A<>B según especificación SQL ANSI/ISO. Varias implementaciones SQL utilizan notaciones alternativas, tales como A!=B. En algunos casos éstas son formas alternativas.

Cuando SQL compara los valores de dos expresiones en el test de comparación se pueden producir tres resultados:

- Si la comparación es cierta, el test produce un resultado TRUE.
- Si la comparación es falsa, el test produce un resultado FALSE.
- Si alguna de las dos expresiones produce un valor NULL, la comparación genera un resultado NULL.

Recuperación de una fila.

El test de comparación más habitual es el que comprueba si el valor de una columna es igual a cierta constante. Cuando la columna es una clave primaria, el test aísla una sola fila de la tabla, produciendo

una sola fila de resultados, como en este ejemplo:

Recupera el nombre y el límite de crédito del cliente número 2.107.

SELECT EMPRESA, LIMITE_CREDITO

FROM CLIENTES

WHERE NUM_CLIE = 2107

EMPRESA LIMITE_CREDITO

Ace International $35,000.00

Este tipo de consulta es el fundamento de los programas de recuperación de base de datos basadas en formularios. El usuario introduce el número de cliente en el formulario, y el programa utiliza el número para construir y ejecutar una consulta. Luego visualiza los datos recuperados en el formulario.

Consideraciones del valor NULL.

El comportamiento de los valores NULL en los tests de comparación puede revelar que algunas nociones obviamente ciertas referentes a consultas SQL no son, de hecho, necesariamente ciertas. Por ejemplo, podría parecer que los resultados de estas dos consultas:

Lista vendedores que superan sus cuotas.

SELECT NOMBRE

FROM REPVENTAS

WHERE VENTAS > CUOTA

NOMBRE

Bill Adams

Mary Jones

Sue Smith

Sam Clark

Dan Roberts

Larry Fitch

Paul Cruz

Lista los vendedores que están por debajo o en su cuota.

SELECT NOMBRE

FROM REPVENTAS

WHERE VENTAS <= CUOTA

NOMBRE

Bob Smith

Nancy Angelli

Deberían incluir cada fila de la Tabla REPVENTAS, pero las consultas producen siete y dos filas, respectivamente, haciendo un total de nueve filas, pero hay diez filas en la Tabla REPVENTAS. La fila de Tom Snyder tiene un valor NULL en la columna CUOTA, puesto que aún no se le ha asignado una cuota. Esta fila no aparece en ninguna de las consultas; desaparece en el test de comparación.

Como muestra este ejemplo, es necesario considerar la gestión del valor NULL cuando se especifica una condición de búsqueda. En la lógica trivaluada de SQL, una condición de búsqueda puede producir un resultado TRUE, FALSE o NULL. Sólo las filas en donde la condición de búsqueda genera un resultado TRUE se incluyen los resultados de la consulta.

6.5.2. Test de rango

SQL proporciona una forma diferente de condición de búsqueda con el test de rango (BETWEEN). El test de rango comprueba si un valor de dato se encuentra entre dos valores especificados. Implica el uso de tres expresiones SQL. La primera expresión define el valor a comprobar; las expresiones segunda y tercera definen los extremos superior e inferior del rango a comprobar. Los tipos de datos de las tres expresiones deben ser comparables. Este ejemplo muestra un test de rango típico:

SELECT NUM_PEDIDO, FECHA_PEDIDO, FAB, PRODUCTO, IMPORTE

FROM PEDIDOS

WHERE FECHA_PEDIDO BETWEEN `01−OCT−89' AND `31−DIC−89'

NUM_PEDIDO FECHA_PEDIDO FAB PRODUCTO IMPORTE

112961 17−DIC−89 REI A244L $31,500.00

112968 12−OCT−89 ACI 41004 $ 3,978.00

112963 17−DIC−89 ACI 41004 $ 3,276.00

112983 27−DIC−89 ACI 41004 $ 702.00

112979 12−OCT−89 ACI 41002 $15,000.00

112992 04−NOV−89 ACI 41002 $ 760.00

112975 12−OCT−89 REI A244G $ 2,100.00

112987 31−DIC−89 ACI 4100Y $27,500.00

El test BETWEEN incluye los puntos extremos del rango, por lo que los pedidos remitidos el 1 de octubre o el 31 de diciembre se incluyen en los resultados de la consulta.

La versión negada del test de rango (NOT BETWEEN) comprueba los valores que caen fuera del rango, como en este ejemplo:

Lista los vendedores cuyas ventas no están entre el 80 y el 120 por 100 de su cuota:

SELECT NOMBRE, VENTAS, CUOTA

FROM REPVENTAS

WHERE VENTAS NOT BETWEEN (0.8 * CUOTA) AND (1.2 * CUOTA)

NOMBRE VENTAS CUOTA

Mary Jones $392,725.00 $300,000.00

Sue Smith $474,050.00 $350,000.00

Bob Smith $142,594.00 $200,000.00

Nancy Angelli $186,042.00 $300,000.00

La expresión de test especificada en el test BETWEEN puede ser cualquier expresión válida, pero en la práctica generalmente es tan sólo un nombre de columna, como en los ejemplos anteriores.

- Si la expresión de test produce un valor NULL, o si ambas expresiones definitorias del rango producen valores NULL, el test BETWEEN devuelve un resultado NULL.
- Si la expresión que define el extremo inferior del rango produce un valor NULL, el test BETWEEN devuelve FALSE si el valor de test es superior al límite superior, y NULL en caso contrario.
- Si la expresión que define el extremo superior del rango produce un valor NULL, el test BETWEEN devuelve FALSE si el valor de test es menor que el límite inferior, y NULL en caso contrario.

Merece la pena advertir que el test BETWEEN no añade realmente potencia expresiva a SQL, ya que puede ser expresado mediante dos tests de comparación. El test de rango:

A BETWEEN B AND C

Es completamente equivalente a:

(A >= B) AND (A <= C)

6.5.3. Test de pertenencia a conjunto

Examina si un valor de dato coincide con uno de una lista de valores objetivo.

Lista los vendedores que trabajan en New York, Atlanta o Denver.

SELECT NOMBRE, CUOTA, VENTAS

FROM REPVENTAS

WHERE OFICINA_REP IN (11, 13, 22)

NOMBRE CUOTA VENTAS

Bill Adams $350,000.00 $367,911.00

Mary Jones $300,000.00 $392,725.00

Sam Clark $275,000.00 $299,912.00

Nancy Angelli $300,000.00 $186,042.00

Se puede comprobar si el valor del dato no corresponde a ninguno de los valores objetivos utilizando la forma NOT IN del test de pertenencia a conjunto. La expresión de test en un test IN puede ser cualquier expresión SQL, pero generalmente es tan sólo un nombre de columna, como en los ejemplos precedentes, Si la expresión de test produce un valor NULL, el test In devuelve NULL. Todos los elementos en la lista de valores objetivo deben tener el mismo tipo de datos, y ese tipo debe ser comparable al tipo de dato de la expresión de test.

Al igual que el test BETWEEN, el test IN no añade potencia expresiva a SQL, ya que la condición de búsqueda

X IN (A, B, C)

Es completamente equivalente a:

(X = A) OR (X = B) OR (X= C)

6.5.4. Test de correspondencia con patrón

Se puede utilizar un test de comparación simple para recuperar las filas en donde el contenido de una consulta de texto se corresponde con un cierto texto particular.

Sin embargo, se podría olvidar fácilmente si el nombre de la empresa era Smith, Smithson o Smithsonian. El test de correspondencia con patrón de SQL puede ser utilizado para recuperar los datos sobre la base de una correspondencia parcial del nombre de cliente.

El test de correspondencia con patrón (LIKE), comprueba si el valor de datos de una columna se ajusta a un *patrón* especificado. El patrón es una cadena que puede incluir uno o más caracteres *comodines*. Estos caracteres se interpretan de una manera especial.

Caracteres comodines.

El carácter comodín signo de porcentaje (%) se corresponde con cualquier secuencia de cero o más caracteres.

Muestra el límite de crédito de Smithson Corp.

SELECT EMPRESA, LIMITE_CREDITO

FROM CLIENTES

WHERE EMPRESA LIKE `Smith% Corp.'

La palabra clave LIKE dice a SQL que compare la columna NOMBRE con el patrón Smith% Corp.. Cualquiera de los nombres siguientes se ajustarían al patrón:

Smith Corp., Smithson Corp., Smithsen Corp., Smithsonian Corp.

El carácter comodín subrayado (_) se corresponde con cualquier carácter simple. Si se está seguro que el nombre de la empresa es o bien Smithson o bien Smithsen, por ejemplo, se puede utilizar esta consulta:

SELECT EMPRESA, LIMITE_CREDITO

FROM CLIENTES

WHERE EMPRESA LIKE `Smiths_n Corp.'

Los caracteres comodines pueden aparecer en cualquier lugar de la cadena patrón, y puede haber varios caracteres comodines dentro de una misma cadena.

Se pueden localizar cadenas que no se ajusten a un patrón utilizando el formato NOT LIKE del test de correspondencia de patrones. El test LIKE debe aplicarse a una columna con un tipo de datos cadena. Si el valor del dato en la columna es NULL, el test LIKE devuelve un resultado NULL.

Caracteres escape *.

Uno de los problemas de la correspondencia con patrones en cadenas es cómo hacer corresponder los propios caracteres comodines como caracteres literales. Para comprobar la presencia de un carácter tanto por ciento en una columna de datos de texto, por ejemplo, no se puede simplemente incluir el signo del tanto por cien en el patrón, ya que SQL lo trataría como un comodín.

El estándar SQL especifica una manera de comparar literalmente caracteres comodines, utilizando un *carácter escape* especial. Cuando el carácter escape aparece en el patrón, el carácter que le sigue inmediatamente se trata como un carácter literal en lugar de cómo un carácter comodín. El carácter escapado puede ser uno de los dos caracteres comodines, o el propio carácter de escape, que ha tomado ahora un significado especial dentro del patrón.

Halla los productos cuyo id comience con las cuatro letras A%BC.

SELECT NUM_PEDIDO, PRODUCTO

FROM PEDIDOS

WHERE PRODUCTO LIKE `A$%BC%' ESCAPE `$'

El primer signo de porcentaje en el patrón, que sigue a un carácter escape, es tratado como un signo literal; el segundo funciona como un comodín.

6.5.5. Test de valor nulo

Los valores NULL crean una lógica trivaluada para las condiciones de búsqueda en SQL. Para una fila determinada, el resultado de una condición de búsqueda puede ser TRUE o FALSE, o puede ser NULL debido a que una de las columnas utilizadas en la evaluación de la condición de búsqueda contiene un valor NULL:

A veces es útil comprobar explícitamente los valores NULL en una condición de búsqueda y gestionarlos directamente. SQL proporciona un test especial de valor nulo (NULL).

Halla el vendedor que aún no tiene asignada una oficina.

SELECT NOMBRE

FROM REPVENTAS

WHERE OFICINA_REP IS NULL

La forma negada del test de valor nulo (IS NOT NULL) encuentra lñas filas que no contiene un valor NULL:

Lista los vendedores a los que se les ha asignado una oficina.

SELECT NOMBRE

FROM REPVENTAS

WHERE OFICINA_REP IS NOT NULL

A diferencia de las condiciones de búsqueda descritas anteriormente, el test de valor nulo no puede producir un resultado NULL. Será siempre TRUE o FALSE.

- **Condiciones de búsqueda compuestas**

Las condiciones de búsqueda simples, descritas en las secciones precedentes devuelven un valor TRUE, FALSE o NULL cuando se aplican a una fila de datos.

Utilizando las reglas de la lógica, se pueden combinar estas condiciones de búsqueda SQL simples para formar otras más complejas.

La palabra clave OR se utiliza para combinar dos condiciones de búsqueda cuando una o la otra (o ambas) deban ser ciertas.

Halla los vendedores que están por debajo de la cuota o con ventas inferiores a $300.000.

SELECT NOMBRE, CUOTA, VENTAS

FROM REPVENTAS

WHERE VENTAS < CUOTA

OR VENTAS < 300000.00

También se puede utilizar la palabra clave AND para combinar dos condiciones de búsqueda que

deban ser ciertas simultáneamente:

```
SELECT NOMBRE, CUOTA, VENTAS

FROM REPVENTAS

WHERE VENTAS < CUOTA

AND VENTAS < 300000.00
```

Finalmente, se puede utilizar la palabra clave NOT para seleccionar filas en donde la condición de búsqueda es falsa:

Halla todos los vendedores que están por debajo de la cuota, pero cuyas ventas no son inferiores a $150.000.

```
SELECT NOMBRE, CUOTA, VENTAS

FROM REPVENTAS

WHERE VENTAS < CUOTA

AND NOT VENTAS < 150000.00
```

Utilizando las palabras clave AND, OR y NOT y los paréntesis para agrupar los criterios de búsqueda, se pueden construir criterios de búsqueda muy complejos.

Cuando se combinan más de dos condiciones de búsqueda con AND, OR y NOT, el estándar especifica que NOT tiene la precedencia más alta, seguido de AND y por último OR. Para asegurar la portabilidad, es siempre una buena idea utilizar paréntesis y suprimir cualquier posible ambigüedad.

6.6. COLUMNAS CALCULADAS

Además de las columnas cuyos valores provienen directamente de la base de datos, una consulta SQL puede incluir *columnas calculadas* cuyos valores se calculan a partir de los valores de los datos almacenados. Para solicitar una columna calculada, se especifica una expresión SQL en la lista de selección. Las expresiones SQL pueden contener sumas, restas, multiplicaciones y divisiones.

También se pueden utilizar paréntesis para construir expresiones más complejas. Naturalmente las columnas referenciadas en una expresión aritmética deben tener un tipo numérico. Si se intenta sumar, restar, multiplicar o dividir columnas que contienen datos de texto, SQL reportará un error.

Esta consulta muestra una columna calculada simple:

Lista la ciudad, la región y el importe por encima o por debajo del objetivo para cada oficina.

```
SELECT CIUDAD, REGION, (VENTAS−OBJETIVO)

FROM OFICINAS
```

CIUDAD REGION (VENTAS−OBJETIVOS)

Denver Oeste −$113,958.00

New York Este $117,637.00

Chicago Este −$64,958.00

Atlanta Este $17,911.00

Los Angeles Oeste $110,915.00

Para procesar la consulta, SQL examina las oficinas, generando una fila de resultados por cada fila de la Tabla OFICINAS. Las dos primeras columnas de resultados provienen directamente de la Tabla OFICINAS. La tercera columna de los resultados se calcula, fila a fila, utilizando los valores de datos de la fila actual de la Tabla OFICINAS.

Muchos productos SQL disponen de operaciones aritméticas adicionales, operaciones de cadenas de caracteres y funciones internas que pueden ser utilizadas en expresiones SQL. Estas pueden aparecer en expresiones de la lista de selección.

Lista el nombre, el mes y el año de contrato para cada vendedor.

```
SELECT NOMBRE, MONTH(CONTRATO), YEAR(CONTRATO)

FROM REPVENTAS
```

También se pueden utilizar constantes SQL por sí mismas como ítems en una lista de selección. Esto puede ser útil para producir resultados que sean más fáciles de leer e interpretar.

Lista las ventas para cada ciudad.

```
SELECT CIUDAD, `tiene ventas de', VENTAS

FROM OFICINAS
```

CIUDAD TIENE VENTAS DE VENTAS

Denver tiene ventas de $186,042.00

New York tiene ventas de $692,637.00

Chicago tiene ventas de $735,042.00

Atlanta tiene ventas de $367,911.00

Los Angeles tiene ventas de $835,915.00

Los resultados de la consulta parecen consistir en una frase distinta por cada oficina, pero realmente es una tabla de tres columnas. Las columnas primera y tercera contienen valores procedentes de la Tabla OFICINAS. La columna siempre contiene la misma cadena de texto de quince caracteres.

6.7. SELECCIÓN DE TODAS LAS COLUMNAS

A veces es conveniente visualizar el contenido de todas las columnas de una tabla. Esto puede ser particularmente útil cuando uno va a utilizar por primera vez una base de datos y desea obtener una rápida comprensión de su estructura y de los datos que contiene. Por conveniencia, SQL permite utilizar un asterisco (*) en lugar de la lista de selección como abreviatura de todas las columnas.

Muestra todos los datos de la Tabla OFICINAS.

SELECT *

FROM OFICINAS

El resultado de la consulta contiene las seis columnas de la Tabla OFICINAS, en el mismo orden de izquierda a derecha que tienen en la tabla.

La selección de todas las columnas es muy adecuada cuando se está utilizando el SQL interactivo de forma casual. Debería evitarse en SQL programado, ya que cambios en la estructura de la base de datos pueden hacer que un programa falle.

- **FILAS DUPLICADAS (DINSTINCT)**

Si una consulta incluye la clave primaria de una tabla en su lista de selección, entonces cada fila de resultados será única (ya que la clave primaria tienen un valor diferente en cada fila). Si no se incluye la clave primaria en los resultados, pueden producirse filas duplicadas. Por ejemplo, supongamos que se hace la siguiente petición:

Lista los números de empleado de todos los directores de oficinas de ventas.

SELECT DIR

FROM OFICINAS

DIR

108

106

104

105

108

Los resultados tienen cinco filas (uno por cada oficina), pero dos de ellas son duplicados exactos la una de la otra. Porque Larry Fitch dirige las oficinas tanto de Los Ángeles como de Denver y su número de empleado (108) aparece en ambas filas de la Tabla OFICINAS. Estos resultados no son probablemente lo que se pretende cuiando se hace la consulta. Si hubiera cuatro directores diferentes, cabría esperar que sólo aparecieran en los resultados cuatro números de empleado.

Se pueden eliminar las filas duplicadas de los resultados de la consulta insertando la palabra clave DISTINCT en la sentencia SELECT justo antes de la lista de selección. He aquí una versión de la consulta anterior que produce los resultados deseados:

Lista los números de empleado de todos los directores de oficinas de ventas.

SELECT DISTINCT DIR

FROM OFICINAS

Conceptualmente, SQL efectúa esta consulta generando primero un conjunto completo de resultados (cinco filas) y eliminando luego las filas que son duplicados exactos de alguna otra para formar los resultados finales, La palabra clave DISTINCT puede ser especificada con independencia de los contenidos de la lista SELECT.

También se puede especificar la palabra clave ALL para indicar explícitamente que las filas duplicadas sean incluidas, pero es innecesario ya que este es el comportamiento por omisión.

- **ORDENACION DE LOS RESULTADOS DE UNA CONSULTA**

Al igual que las filas de una tabla en la base de datos, las filas de los resultados de una consulta no están dispuestas en ningún orden particular. Se puede pedir a SQL que ordene los resultados de una consulta incluyendo la cláusula ORDER BY en la sentencia SELECT. La cláusula ORDER BY consta de las palabras claves ORDER BY, seguidas de una lista de especificaciones de ordenación separadas por comas. Por ejemplo, los resultados de esta consulta están ordenados en dos columnas, REGION y CIUDAD.

Muestra las ventas de cada oficina, ordenadas en orden alfabético por región y dentro de cada región por ciudad.

SELECT CIUDAD, REGION, VENTAS

FROM OFICINAS

ORDER BY REGION, CIUDAD

La primera especificación de ordenación (REGION) es la clave de la ordenación *mayor*, las que le sigan (CIUDAD, en este caso) son progresivamente claves de ordenación *menores*, utilizadas para desempatar cuando dos filas de resultados tienen los mismos valores para las claves mayores. Utilizando la cláusula ORDER BY se puede solicitar la ordenación en secuencia ascendente o descendente, y se puede ordenar con respecto a cualquier elemento en la lista de selección de la consulta.

Por omisión, SQL ordena los datos en secuencia ascendente. Para solicitar ordenación en secuencia descendente, se incluye la palabra clave DESC en la especificación de ordenación, como en este ejemplo:

Lista las oficinas, clasificadas en orden descendente de ventas, de modo que las oficinas con mayores aparezcan en primer lugar.

SELECT CIUDAD, REGION, VENTAS

FROM OFICINAS

ORDER BY VENTAS DESC

CIUDAD REGION VENTAS

Los Angeles Oeste $835,915.00

Chicago Este $735,042.00

New York Este $692,637.00

Atlanta Este $367,911.00

Denver Oeste $186,042.00

También se puede utilizar la palabra clave ASC para especificar el orden ascendente, pero puesto que ésta es la secuencia de ordenación por omisión, la palabra clave se suele omitir.

Si la columna de resultados de la consulta utilizada para ordenación es una columna calculada, no tiene nombre de columna que se pueda emplear en una especificación de ordenación. En este caso, debe especificarse un número de columna en lugar de un nombre, como en este ejemplo.

Lista las oficinas, clasificadas en orden descendente de rendimiento de ventas de modo que las oficinas con mejor rendimiento aparezcan primero.

SELECT CIUDAD, REGION, (VENTAS−OBJETIVO)

FROM OFICINAS

ORDER BY 3 DESC

Estos resultados están ordenados por la tercera columna, que es la diferencia calculada entre VENTAS y OBJETIVO para cada oficina.

- **REGLAS PARA PROCESAMIENTO DE CONSULTAS DE TABLA UNICA**

Las consultas de tabla única son generalmente sencillas, y normalmente es fácil entender el significado de una consulta tan sólo leyendo la sentencia SELECT.

Para generar los resultados de una consulta correspondiente a una sentencia SELECT:

- Comenzar con la tabla designada en la cláusula FROM.
- Si hay cláusula WHERE, aplicar su condición de búsqueda a cada fila de la tabla, reteniendo aquellas filas para las cuales la condición de búsqueda es TRUE, y descartando aquéllas para las cuáles es FALSE o NULL.
- Para cada fila restante, calcular el valor de cada elemento en la lista de selección para producir una única fila de resultados. Por cada referencia de columna, utilizar el valor de la columna en la fila actual.
- Si se especifica SELECT DISTINCT, eliminar las filas duplicadas de los resultados que se hubieran producido.
- Si hay una cláusula ORDER BY, ordenar los resultados de la consulta según se especifiquen.

Las filas generadas por este procedimiento forman los resultados de la consulta.

- **COMBINACION DE LOS RESULTADOS DE UNA CONSULTA**

Ocasionalmente, es conveniente combinar los resultados de dos o más consultas en una única tabla de resultados totales. SQL permite esta capacidad gracias a la característica UNION de la sentencia SELECT.

Lista todos los productos en donde el precio del producto exceda de $2.000 o en donde más de $30.000 del producto hayan sido incluidos en un solo pedido.

La operación UNION produce una única tabla de resultados que combina las filas de la primera consulta con las filas de los resultados de la segunda consulta. La sentencia SELECT que especifica la operación UNION tiene el siguiente aspecto:

SELECT ID_FAB, ID_PRODUCTO

FROM PRODUCTOS

WHERE PRECIO >2000.00

UNION

SELECT DISTINCT FAB, PRODUCTO

FROM PEDIDOS

WHERE IMPORTE >30000.00

Hay varias restricciones sobre las tablas que pueden combinarse con una operación UNION:

- Ambas tablas deben contener el mismo número de columnas.
- El tipo de datos de cada columna en la primera tabla debe ser el mismo que el tipo de datos de la columna correspondiente en la segunda tabla.
- Ninguna de las dos tablas puede estar ordenadas con la cláusula ORDER BY. Sin embargo, los resultados combinados pueden ser ordenados, según se describe en la sección siguiente.

Los nombres de columna de las dos consultas combinadas mediante una UNION no tienen que ser idénticos. En el ejemplo anterior, la primera tabla de resultados tenía columnas de nombre ID_FAB e ID_PRODUCTO, mientras que la segunda tabla de resultados tenía columnas de nombres FAB y PRODUCTOS. Puesto que las columnas de las dos tablas pueden tener nombres diferentes, las columnas de los resultados producidos por la operación UNION están sin designar.

Solamente permite nombres de columna o una especificación de todas las columnas (SELECT *) en la lista de selección, y prohíbe las expresiones en la lista de selección.

Por omisión, la operación UNION elimina las filas duplicadas como parte de su procesamiento. Por tanto, el grupo combinado de resultados contiene una sola fila para el producto REI−a244l;

Lista todos los productos en donde el precio del producto exceda de $2.000 o en donde más de $30.0000 del producto hayan sido incluidos en un solo pedido.

SELECT ID_FAB, ID_PRODUCTO

FROM PRODUCTOS

WHERE PRECIO >2000.00

UNION

SELECT DISTINCT FAB, PRODUCTO

FROM PEDIDOS

WHERE IMPORTE >30000.00

ACI 4100Y

REI A244L

ACI 4100Z

REI A244R

IMM 775C

REI A244L

REI A244R

La eliminación de filas duplicadas en los resultados de la consulta es un proceso que consume mucho tiempo, especialmente si los resultados contienen un gran número de filas. Si se sabe, en base a las consultas individuales implicadas, que la operación UNION no puede producir filas duplicadas, se debería utilizar específicamente la operación UNION ALL, ya que la consulta se ejecutará mucho más rápidamente.

Uniones y ordenación *

La cláusula ORDER BY no puede aparecer en ninguna de las dos sentencias SELECT combinadas por una operación UNION. No tendría mucho sentido ordenar los dos conjuntos de resultados de ninguna manera, ya que éstos se dirigen directamente a la operación UNION y nunca son visibles al usuario. Sin embargo, el conjunto combinado de los resultados de la consulta producidos por la operación UNION puede ser ordenado especificando una cláusula ORDER BY después de la segunda sentencia SELECT. Ya que las columnas producidas por la operación UNION no tienen nombres, la cláusula ORDER BY debe especificar las columnas por número.

He aquí la misma consulta de productos con los resultados ordenados por fabricante y número de producto:

Lista todos los productos en donde el precio del producto supera a $2.000 o en donde más de $30.0000 del producto hayan sido incluidos en un solo pedido, clasificados por fabricante y número de producto.

SELECT ID_FAB, ID_PRODUCTO

FROM PRODUCTOS

WHERE PRECIO >2000.00

UNION

SELECT DISTINCT FAB, PRODUCTO

FROM PEDIDOS

WHERE IMPORTE >30000.00

ORDER BY 1, 2

ACI 4100Y

ACI 4100Z

IMM 775C

REI A244L

REI A244R

Uniones múltiples *.

La operación UNION puede ser utilizada repetidamente para combinar tres o más conjuntos de resultados. La unión de la TABLA B y la TABLA C en la figura produce una única tabla combinada. Esta tabla se combina luego con la TABLA A en otra operación UNION. La consulta de la figura se escribe de este modo:

SELECT *

FROM A

UNION (SELECT *

FROM B

UNION

SELECT *

FROM C) **Tabla A**

Tabla B			Bill			
Bill			Mary			
Sue			George			**Resultados de la consulta**
Julia			Fred			
Harry						Bill
			Bill			Mary
Tabla C						George
Mary			Sue			Fred
George			Julia			Sue
Bill			Harry			Julia
Harry			Mary			Harry
			George			

Los paréntesis que aparecen en la consulta indican qué UNION debería ser realizada en primer lugar. De hecho, si todas las uniones de la sentencia eliminan filas duplicadas, o si todas ellas retienen filas duplicadas, el orden en que se efectúan no tienen importancia. Estas tres expresiones son equivalentes:

A UNION (B UNION C)

(A UNION B) UNION C

(A UNION C) UNION B

Sin embargo, si las uniones implican una mezcla de UNION y UNION ALL, el orden de la evaluación si importa. Si esta expresión:

A UNION ALL B UNION C

Se interpreta como:

A UNION ALL (B UNION C)

Entonces se producen **diez** filas de resultados (seis de la UNION interna, más cuatro filas de la TABLA A). Sin embargo, si se interpreta como obtendrá 7 filas de resultado (quita los repetidos):

(A UNION ALL B) UNION C

- **CONSULTAS DE RESUMEN, HAVING, GROUP BY**

SQL permite resumir datos de la base de datos mediante un conjunto de *funciones de columna*. Una función de columna SQL acepta una columna entera de datos como argumento y produce un único dato que resume la columna. Las funciones de columna ofrecen diferentes tipos de datos resumen:

- ♦ SUM() calcula el total de una columna.
- ♦ AVG() calcula el valor promedio de una columna.
- ♦ MIN() encuentra el valor más pequeño de una columna.
- ♦ MAX() encuentra el valor mayor de una columna.
- ♦ COUNT() cuenta el número de valores de una columna.
- ♦ COUNT(*) cuenta las filas de una consulta.

El argumento de una función columna puede ser un solo nombre de columna o puede ser una expresión SQL.

Cálculo del total de una columna (SUM)

La función columna SUM() calcula la suma de una columna de valores de datos. Los datos de la columna deben tener un tipo numérico (entero, decimal, coma flotante o monetario). El resultado de la función SUM() tiene el mismo tipo de datos básico que los datos de la columna, pero el resultado puede tener una precisión superior.

¿Cuáles son las cuotas y ventas totales para todos los vendedores?

SELECT SUM(CUOTA), SUM(VENTAS)

FROM REPVENTAS

SUM(CUOTA) SUM(VENTAS)

$2,700,000.00 $2,893,352.00

Cálculo del promedio de una columna

La función de columna AVG() calcula el promedio de una columna de valores de datos. Al igual que una función SUM(), los datos de la columna deben tener un tipo numérico. Ya que la función AVG() suma los valores de la columna y luego lo divide por el número de valores, su resultado puede tener un tipo de dato diferente al de los valores de columna.

Por ejemplo, si se aplica la función AVG() a una columna de enteros, el resultado será un número decimal o un número de coma flotante, dependiendo del producto DBMS concreto que se esté utilizando.

Calcula el precio medio de los productos del fabricante ACI.

SELECT AVG(PRECIO)

FROM PRODUCTOS

WHERE ID_FAB = `ACI'

AVG(PRECIO)

$804.29

Determinación de valores extremos (MIN y MAX()

Las funciones de columna MIN() Y MAX() determinan los valores menor y mayor de una columna, respectivamente. Los datos de la columna pueden contener información numérica, de cadena o de fecha/hora. El resultado de la función MIN() y MAX() tiene exactamente el mismo tipo de dato que los datos de la columna.

¿Cuáles son las cuotas asignadas mínima y máxima?

SELECT MIN(CUOTA), MAX(CUOTA)

FROM REPVENTAS

MIN(CUOTA) MAX(CUOTA)

$200,000.00 $350,000.00

¿Cuál es la fecha de pedido más antigua en la base de datos?

SELECT MIN(FECHA_PEDIDO)

FROM PEDIDOS

MIN(FECHA_PEDIDO)

04−ENE−89

Cuando las funciones de columnas MIN() y MAX() se aplican a datos numéricos, SQL compara los números en orden algebraico (los números negativos grandes son menores que los números negativos pequeños, los cuáles son menores que cero, el cual a su vez es menor que todos los números positivos). Las fechas se comparan secuencialmente (las fechas más antiguas son más pequeñas que las fechas más recientes).

Cuando se utiliza MIN() YMAX() con datos de cadenas, la comparación de las cadenas depende del conjunto de caracteres que esté siendo utilizado. En una computadora personal o en una minicomputadora los dígitos se encuentran delante de las letras en la secuencia de ordenación, y todos los caracteres mayúsculas se encuentran delante de todos los caracteres minúsculas.

Cuenta de valores de datos (COUNT)

La función de columna COUNT() cuenta el número de valores de datos que hay en una columna. Los datos de la columna pueden ser de cualquier tipo. La función COUNT() siempre devuelve un entero, independientemente del tipo de datos de la columna.

¿Cuántos clientes hay?

SELECT COUNT(NUM_CLIE)

FROM CLIENTES

COUNT(NUM_CLIE)

21

¿Cuántos vendedores superan su cuota?

SELECT COUNT(NOMBRE)

FROM REPVENTAS

WHERE VENTAS > CUOTA

COUNT(NOMBRE)

7

¿Cuántos pedidos de más de $25.000 hay en los registros?

SELECT COUNT (IMPORTE)

FROM PEDIDOS

WHERE IMPORTE > 25000.00

COUNT(IMPORTE)

4

Obsérvese que la función COUNT() ignora los valores de los datos de la columna; simplemente cuenta cuántos datos hay. En consecuencia, no importa realmente qué columna se especifica como argumento de la función COUNT().

SQL permite una función de columna especial COUNT(*) que cuenta filas en lugar de valores de datos. He aquí la misma consulta, reescrita una vez más para utilizar la función COUNT (*):

SELECT COUNT(*)

FROM PEDIDOS

WHERE IMPORTE > 25000.00

COUNT (*)

4

Si se piensa en la función COUNT(*) como en una función cuenta filas, la consulta resulta más fácil de leer. En la práctica, se utiliza casi siempre la función COUNT(*) en lugar de la función COUNT() para contar filas.

Valores NULL y funciones de columna

Las funciones de columna SUM(), AVG(), MIN(), MAX() y COUNT() aceptan cada una de ellas una columna de valores de datos como argumento y producen un único valor como resultado. ¿Qué sucede si uno o más de los valores de la columna es un valor NULL? El estándar SQL ANSI/ISO especifica que los valores NULL de la columna sean ignorados por las funciones de la columna.

Esta consulta muestra como la función de columna COUNT() ignora los valores NULL de una columna:

SELECT COUNT(*), COUNT (VENTAS), COUNT(CUOTA)

FROM REPVENTAS

COUNT(*) COUNT(VENTAS) COUNT(CUOTA)

10 10 9

La tabla REPVENTAS contiene diez filas, por lo que COUNT(*) devuelve una cuenta de diez. La columna VENTAS contiene diez valores no NULL, por lo que la función COUNT(VENTAS) también devuelve una cuenta de diez. La columna CUOTA es NULL para el vendedor más reciente. LA función COUNT(CUOTA) ignora este valor NULL y devuelve una cuenta de nueve. Debido a estas anomalías, la función COUNT(*) es utilizada casi siempre en lugar de la función COUNT(), a menos que específicamente, se desee excluir del total los valores NULL de una columna particular.

Ignorar los valores NULL tiene poco impacto en las funciones de columna MIN() y MAX().

SELECT SUM(VENTAS), SUM(CUOTA),

(SUM(VENTAS) − SUM(CUOTA)), SUM(VENTAS−CUOTA)

FROM REPVENTAS

SUM(VENTAS) SUM(CUOTA) (SUM(VENTAS) − SUM(CUOTA)) SUM (VENTAS−CUOTA)

$2,893.532.00 $2,700,000.00 $193,532.00 $117,547.00

Sería de esperar que las dos expresiones:

(SUM(VENTAS) − SUM(CUOTA) Y SUM(VENTAS−CUOTA)

en la lista de selección produjeran resultados idénticos, pero el ejemplo muestra que no es así. El vendedor con un valor NULL en la columna CUOTA es de nuevo la razón. La expresión:

SUM(VENTAS)

totaliza las ventas para los diez vendedores, mientras que la expresión:

SUM(CUOTA)

Totaliza solamente los nueve valores de cuota no NULL. La expresión:

SUM(VENTAS) − SUM(CUOTA)

calcula la diferencia de estos dos importes. Sin embargo, la función de columna:

SUM(VENTAS−CUOTA)

Tiene un valor de argumento no NULL para sólo nueve de los diez vendedores. En la fila con un valor de cuota NULL, la resta produce un NULL, que es ignorado por la función SUM(). Por tanto, las ventas del vendedor sin cuota, que están incluidas en el cálculo previo, se excluyen de este cálculo.

- Si todos los datos de una columna son NULL, las funciones de columna SUM(), AVG(), MIN(), y MAX() devuelven un valor NULL; la función COUNT() devuelve un valor de cero.
- Si no hay datos en la columna (es decir, la columna está vacía), las funciones de columna SUM(), AVG(), MIN() y MAX() devuelven un valor cero.
- La función COUNT(*) cuenta filas, y no depende de la presencia o ausencia de valores NULL en la columna.

Eliminación de filas duplicadas (DISTINCT)

Recuérdese que se puede especificar la palabra clave DISTINCT al comienzo de la lista de selección para eliminar las filas duplicadas del resultado de la consulta. También se puede pedir a SQL que elimine valores duplicados de una columna antes de aplicarle una función de columna. Para eliminar valores duplicados, la palabra clave DISTINCT se incluye delante del argumento de la función de columna, inmediatamente después del paréntesis abierto.

¿Cuántos títulos diferentes tienen los vendedores?

SELECT COUNT (DISTINCT TITULO)

FROM REPVENTAS

COUNT (DISTINCT TITULO)

3

Cuando se utiliza la palabra clave DISTINCT, el argumento de la función columna debe ser un simple nombre de columna; no puede ser una expresión. El estándar no permite el uso de la palabra clave DISTINCT con las funciones de columna MIN() y MAX(). DISTINCT no puede ser especificado con la función COUNT(*).

La palabra clave DISTINCT sólo se puede especificar una vez en una consulta. Si aparece en l argumento de una función de columna, no puede aparecer en ninguna otra. Si se especifica delante de la lista de selección, no puede aparecer en ninguna función de columna.

Consultas agrupadas (Cláusula GROUP BY)

Las consultas resumen descritas hasta ahora son, como los totales al final de un informe. Condensan todos los datos detallados del informe en una única fila resumen de datos.

¿Cuál es el tamaño medio de pedido?

SELECT AVG(IMPORTE)

FROM PEDIDOS

AVG(IMPORTE)

$8,256.37

¿Cuál es el pedido medio de cada vendedor?

SELECT REP, AVG(IMPORTE)

FROM PEDIDOS

GROUP BY REP

REP AVG(IMPORTE)

101 $8,876.00

102 $5,694.00

103 $1,350.00

105 $7,685.40

106 $16,479.00

107 $11,477.33

108 $3,552.50

110 $11,566.00

La primera consulta es una consulta resumen simple como la de los ejemplos anteriores. La segunda consulta produce varias filas resumen, una fila por cada grupo, resumiendo los pedidos aceptados por un solo vendedor. Conceptualmente, SQL lleva a cabo la consulta del modo siguiente:

- SQL divide los pedidos en grupos de pedidos, un grupo por cada vendedor. Dentro de cada grupo, todos los pedidos tienen el mismo valor en la columna REP.
- Por cada grupo, SQL calcula el valor medio de la columna IMPORTE para todas las filas del grupo, y genera una única fila resumen de resultados. La fila contiene el valor de la columna REP del grupo y el pedido medio calculado.

Una consulta que incluya la cláusula GROUP BY se denomina *consulta agrupada*, ya que agrupa los datos de las tablas fuentes y produce una única fila resumen por cada grupo de filas. Las columnas indicadas en la cláusula GROUP BY se denominan *columnas de agrupación* de la consulta ya que son las que determinan cómo se dividen las filas en grupos.

Múltiple columnas de agrupación

SQL puede agrupar resultados de consulta basándose en contenidos de dos o más columnas. Por ejemplo, supongamos que se desea agrupar los pedidos por vendedor y por cliente. Esta consulta agrupa los datos basándose en ambos criterios:

Calcula los pedidos totales por cada cliente y por cada vendedor.

SELECT REP, CLIE, SUM(IMPORTE)

FROM PEDIDOS

GROUP BY REP, CLIE

REP CLIE SUM(IMPORTE)

101 2102 $3,978.00

101 2108 $ 150.00

101 2113 $22,500.00

102 2106 $4,026.00

102 2114 $15,000.00

102 2120 $3,750.00

103 2111 $2,750.00

105 2103 $35,582.00

105 2111 $3,745.00

Las ordenes de agrupación deben agruparse por los campos que no son funciones de columna en la lista de campos.

La cláusula COMPUTE calcula subtotales y sub−subtotales como se muestra en este ejemplo:

Calcula los pedidos totales para cada cliente de cada vendedor, ordenados por vendedor, y dentro de cada vendedor por cliente.

SELECT REP, CLIE, IMPORTE

FROM PEDIDOS

ORDER BY REP, CLIE

COMPUTE SUM(IMPORTE) BY REP, CLIE

COMPUTE SUM(IMPORTE), AVG(IMPORTE) BY REP

REP CLIE IMPORTE

101 2102 $3,978.00

sum

−−−−−−− −−−−−−−−−−−−

$3,978.00

101 2108 $150.00

sum

−−−−−−− −−−−−−−−−−−−

$150.00

101 2113 $22,500.00

sum

−−−−−−− −−−−−−−−−−−−

$22,500.00

sum

−−−−−−− −−−−−−−−−−−−

$26,628.00

avg

−−−−−−− −−−−−−−−−−−−

$8,876.00

102 2106 $2,130.00

102 2106 $1,896.00

sum

−−−−−−− −−−−−−−−−−−−

$4,026.00

102 2114 $15,000.00

sum

−−−−−−− −−−−−−−−−−−−

$15,000.00

102 2120 $3,750.00

sum

−−−−−−− −−−−−−−−−−−−

$3,750.00

sum

−−−−−−− −−−−−−−−−−−−

$22,776.00

avg

−−−−−−− −−−−−−−−−−−−

$5,694.00

Restricciones en consultas agrupadas

Las columnas de agrupación deben ser columnas efectivas de las tablas designadas en la cláusula FROM de la consulta. No se pueden agrupar las filas basándose en el valor de una expresión calculada.

También hay restricciones sobre los elementos que pueden aparecer en la lista de selección de una consulta agrupada. Todos los elementos de la lista de selección deben tener un único valor por cada grupo de filas. Básicamente, esto significa que un elemento de selección en una consulta agrupada puede ser:

- Una constante
- Una función de columna
- Una columna de agrupación, que por definición tiene el mismo valor en todas las filas del

grupo

- Una expresión que afecte a combinaciones de los anteriores

En la práctica, una consulta agrupada incluirá siempre una columna de agrupación y una función de columna en su lista de selección.

Valores NULL en columnas de agrupación

Un valor NULL presenta un problema especial cuando aparece en una columna de agrupación. Si el valor de la columna es desconocido, ¿en qué grupo debería colocarse la fila? En la cláusula WHERE, cuando se comparan dos valores NULL diferentes, el resultado es NULL (no TRUE), es decir, los dos valores NULL no se consideran iguales. Aplicando el mismo convenio a la cláusula GROUP BY se forzaría a SQL a colocar cada fila con una columna de agrupación NULL en un grupo aparte.

En la práctica, esta regla se demuestra que es demasiado rígida. En vez de ello, el estándar SQL ANSI/ISO considera que dos valores NULL son iguales a efectos de la cláusula GROUP BY. Si dos filas tienen NULL en las mismas columnas de agrupación no NULL, se agrupan dentro del mismo grupo de filas.

SELECT PELO, OJOS, COUNT(*)

FROM REPVENTAS

GROUP BY PELO, OJOS

NOMBRE	PELO	OJOS
Cindy	Castaño	Azules
Louise	NULL	Azules
Harry	NULL	Azules
Samantha	NULL	NULL
Joanne	NULL	NULL
George	Castaño	NULL
Mary	Castaño	NULL
Paula	Castaño	NULL
Kevin	Castaño	NULL
Joel	Castaño	Negros
Susan	Rubio	Azules
Marie	Rubio	Azules

Condiciones de búsqueda de grupos (cláusula HAVING)

Al igual que la cláusula WHERE puede ser utilizada para seleccionar y rechazar filas individuales que participan en una consulta, la cláusula HAVING puede ser utilizada para seleccionar y rechazar grupos de filas. El formato de la cláusula HAVING es análogo al de la cláusula WHERE, consistiendo en la palabra clave HAVING seguida de una condición de búsqueda.

¿Cuál es el tamaño de pedido promedio para cada vendedor cuyos pedidos totalizan más de $30.000?

SELECT REP, AVG(IMPORTE)

FROM PEDIDOS

GROUP BY REP

HAVING SUM(IMPORTE) > 30000.00

REP AVG(IMPORTE)

105 $7,865.40

106 $16,479.00

107 $11,477.33

- $8,376.14

La cláusula GROUP BY dispone primero de los pedidos en grupos por vendedor. La cláusula HAVING elimina entonces los grupos en donde el total de los pedidos no excede de $30.000. Finalmente, la cláusula SELECT calcula el tamaño de pedido medio para cada uno de los grupos restantes y genera los resultados de la consulta.

Las condiciones de búsqueda que se pueden especificar en la cláusula HAVING son las mismas de la cláusula WHERE.

Por cada oficina con dos o más personas, calcular la cuota total y las ventas totales para todos los vendedores que trabajan en la oficina.

SELECT CIUDAD, SUM(CUOTA), SUM(REPVENTAS.VENTAS)

FROM OFICINAS, REPVENTAS

WHERE OFICINA = OFICINA_REP

GROUP BY CIUDAD hay una oficina por ciudad

HAVING COUNT(*) >= 2

CIUDAD SUM(CUOTA) SUM(REPVENTAS.VENTAS)

Chicago $775,000.00 $735,042.00

Los Angeles $700,000.00 $835,915.00

New York $575,000.00 $692,637.00

SQL gestiona esta consulta del modo siguiente:

- Compone las Tablas OFICINAS y REPVENTAS para hallar la ciudad en donde trabaja cada vendedor.
- Agrupa las filas resultantes por oficina.
- Elimina los grupos con dos o menos filas, éstas representan oficinas que no satisfacen el criterio de la cláusula HAVING.
- Calcula la cuota total y las ventas totales para cada grupo.

Muestra el precio, las existencias y la cantidad total de los pedidos de cada producto para los cuales la

cantidad total pedida es superior al 75 por 100 de las existencias.

```
SELECT DESCRIPCION, PRECIO, EXISTENCIAS, SUM(CANT)

FROM PRODUCTOS, PEDIDOS

WHERE FAB = ID_FAB

AND PRODUCTO = ID_PRODUCTO

GROUP BY ID_FAB, ID_PRODUCTO, DESCRIPCION, PRECIO, EXISTENCIAS

HAVING SUM(CANT) > (0.75 * EXISTENCIAS)

ORDER BY EXISTENCIAS DESC
```

DESCRIPCION PRECIO EXISTENCIAS SUM(CANT)

Reductor $355.00 38 32

Ajustador $25.00 37 30

Bancada motor $243.00 15 16

Bisagra Dcha. $4,500.00 12 15

Riostra 1−Tm $1,425.00 5 22

Para procesar esta consulta, SQL efectúa conceptualmente los siguientes pasos:

- Compone las Tablas PRODUCTOS y PEDIDOS para obtener la descripción, precio y existencias de cada producto pedido.
- Agrupa las filas resultantes por fabricante e id de producto.
- Elimina los grupos en donde la cantidad pedida es menor al 75% de las existencias.
- Calcula la cantidad total pedida para cada grupo.
- Genera una fila resumen de resultados por cada grupo.
- Ordena los resultados para que los productos con el mayor valor de existencias aparezcan en primer lugar.

Restricciones en condiciones de búsqueda de grupos

La cláusula HAVING se utiliza para incluir o excluir grupos de filas de los resultados de la consulta, por lo que la condición de búsqueda que especifica debe ser aplicable al grupo en su totalidad en lugar de a filas individuales. Esto significa que un elemento que aparezca dentro de la condición de búsqueda en una cláusula HAVING puede ser:

- ♦ Una constante
- ♦ Una función de columna, que produzca un único valor resumen de las filas del grupo
- ♦ Una columna de agrupación, que por definición tiene el mismo valor en todas las filas del grupo
- ♦ Una expresión que afecte a combinaciones de los anteriores

En la práctica, la condición de búsqueda de la cláusula HAVING incluirá siempre al menos una

función de columna. Si no lo hiciera, la condición de búsqueda podría expresarse con la cláusula WHERE y aplicarse a filas individuales. El modo más fácil de averiguar si una condición de búsqueda pertenece a la cláusula WHERE o a la cláusula HAVING es recordar cómo se aplican ambas cláusulas:

- La cláusula WHERE se aplica a filas individuales, por lo que las expresiones que contiene deben ser calculables para filas individuales.
- La cláusula HAVING se aplica a grupos de filas, por lo que las expresiones que contengan deben ser calculables para un grupo de filas.

Valores NULL y condiciones de búsqueda de grupos

Al igual que la condición de búsqueda de la cláusula WHERE, la condición de búsqueda de la cláusula HAVING puede producir uno de los tres resultados siguientes:

- Si la condición de búsqueda es TRUE, se retiene el grupo de filas y contribuye con una fila resumen a los resultados de la consulta.
- Si la condición de búsqueda es FALSE, el grupo de filas se descarta y no contribuye con una fila resumen a los resultados de la consulta.
- Si la condición de búsqueda es NULL, el grupo de filas se descarta y no contribuye con una fila resumen a los resultados de la consulta.

HAVING sin GROUP BY

La cláusula HAVING se utiliza casi siempre juntamente con la cláusula GROUP BY, pero la sintaxis de la sentencia SELECT no lo precisa. Si una cláusula HAVING aparece sin una cláusula GROUP BY, SQL considera el conjunto entero de resultados detallados como un único grupo. En otras palabras, las funciones de columna de la cláusula HAVING se aplican a un solo y único grupo para determinar si el grupo está incluido o excluido de los resultados, y ese grupo está formado por todas las filas. El uso de una cláusula HAVING sin una cláusula correspondiente GROUP BY casi nunca se ve en la práctica.

- **SINTAXIS DE LA ORDEN SELECT**

La Figura muestra el formato completo de la sentencia SELECT, que consta de seis cláusulas. Las cláusulas SELECT y FROM de la sentencia son necesarias. Las cuatro cláusulas restantes (where, order, group y having) son opcionales. Se incluyen en la sentencia SELECT solamente cuando se desean utilizar las funciones que proporcionan. La función de cada cláusula esta resumida a continuación:

- La cláusula SELECT lista los datos a recuperar por la sentencia SELECT. Los ítems pueden ser columnas de la base de datos o columnas a calcular por SQL cuando efectúe la consulta.
- La cláusula FROM lista las tablas que contienen los datos a recuperar por la consulta.

SELECT *ítem seleccionado*

ALL ,

DISTINCT *

FROM *especificación−de−tabla*

,

WHERE *condición de búsqueda*

GROUP BY *columna−de−agrupación*

,

HAVING *condición−de−búsqueda*

ORDER BY *especificación−de−ordenación*

,

- ♦ La cláusula WHERE dice a SQL que incluya sólo ciertas filas de datos en los resultados de la consulta.
- ♦ La cláusula GROUP BY especifica una consulta resumen. En vez de producir una fila de resultados por cada fila de datos de la base de datos, una consulta resumen agrupa todas las filas similares y luego produce una fila resumen de resultados para cada grupo.
- ♦ La cláusula HAVING dice a SQL que incluya sólo ciertos grupos producidos por la cláusula GROUP BY en los resultados de la consulta. Al igual que la cláusula WHERE, utiliza una condición de búsqueda para especificar los grupos deseados.
- ♦ La cláusula ORDER BY ordena los resultados de la consulta en base a los datos de una o más columnas.

- **RESUMEN**
  - ♦ La sentencia SELECT se utiliza para expresar una consulta SQL. Toda sentencia SELECT produce una tabla de resultados que contienen una o más columnas y cero o más filas.
  - ♦ La cláusula FROM especifica la(s) tabla(s) que contiene(n) los datos a recuperar por una consulta.
  - ♦ La cláusula SELECT especifica la(s) columna(s) de datos a incluir en los resultados de la consulta, que pueden ser columnas de datos de la base de datos o columnas calculadas.
  - ♦ La cláusula WHERE selecciona las filas a incluir en los resultados aplicando una condición de búsqueda a las filas de la base de datos.
  - ♦ Una condición de búsqueda puede seleccionar filas mediante comparación de valores, mediante comparación de un valor con un rango o un grupo de valores, por correspondencia con un patrón de cadena o por comprobación de valores NULL.
  - ♦ Las condiciones de búsqueda simples pueden combinarse mediante AND, OR y NOT para formar condiciones de búsqueda más complejas.
  - ♦ La cláusula ORDER BY especifica que los resultados de la consulta deben ser ordenados en sentido ascendente o descendente, basándose en los valores de una o más columnas.
  - ♦ La operación UNION puede ser utilizada dentro de una sentencia SELECT para combinar dos o más conjuntos de resultados y formar un único conjunto.
  - ♦ Las consultas resumen utilizan funciones de columna SQL para condensar una columna de valores en un único valor que resuma la columna.
  - ♦ Las funciones de columna pueden calcular el promedio, suma, el valor mínimo y máximo de una columna, contar el número de valores de datos de una columna o contar el número de filas de los resultados de la consulta.
  - ♦ Una consulta resumen sin una cláusula GROUP BY genera una única fila de resultados, resumiendo todas las filas de una tablas o de un conjunto compuestos de tablas.
  - ♦ Una consulta resumen con una cláusula GROUP BY genera múltiples filas de resultados.
- **EJERCICIOS**
- Muestra el nombre, las ventas y la cuota del empleado número 105.
- Lista las oficinas cuyas ventas están por debajo del 80 por 100 del objetivo. Mostrar la ciudad, ventas y el objetivo.

- Lista las oficinas no dirigidas por el empleado número 108. Mostrar la ciudad y nº de empleado.
- Hallar los pedidos cuyo importe es superior a 20.000 e inferior a 29.999. Mostrar el número de pedido e importe.
- Hallar los pedidos remitidos un jueves en enero de 1990. Mostrar el número de pedido, fecha de pedido e importe.
- Hallar todos los pedidos obtenidos por cuatro vendedores específicos (elegir 4 que ya tengáis). Mostrar el número de pedido, representante e importe.
- Buscar el nombre de las empresas cuyo nombre empiezan en Smiths, acaba la palabra en n y en el centro hay una letra desconocida. Por detrás, el nombre puede tener: Corp o Inc.
- Hallar todos los nombres de vendedores que cumplan alguna de las 3 siguientes opciones:
- Trabajan en Denver, New York o Chicago.
- No tienen director y fueron contratados a partir de Junio de 1988.
- Sus ventas están por encima de la cuota, pero tienen ventas de 600.000 o menos.
- Por cada producto mostrar el Identificador del producto, su descripción, el inventario (existencias por el precio).
- Mostrar cada vendedor con su cuota elevada en un 3 por 100 de sus ventas anuales. Además, mostrar el nombre de vendedor y su cuota actual.
- Listar las oficinas, clasificadas en orden alfabético por región, y dentro de cada región por orden descendente de rendimiento de ventas (ventas menos objetivo). Por cada oficina se mostrará la ciudad, región y el rendimiento de ventas.
- Calcular el rendimiento de la cuota promedio de los vendedores. Dar la orden pertinente.
- Calcular el importe medio de los pedidos realizados por el cliente Acme Mfg.
- Dar la orden que calcule el mejor rendimiento de ventas de todos los vendedores.
- Listar el rango de cuotas asignadas en cada oficina. Mostrar la oficina, el rango superior e inferior.
- Mostrar el número de vendedores asignados en cada oficina.
- Mostrar los diferentes clientes que son atendidos por cada vendedor.

TEMA VII

CONSULTAS MULTIPLES

7.1 CONSULTA DE VARIAS TABLAS

7.2. COMPOSICIONES CON CRITERIOS DE SELECCIÓN DE FILA

7.3. MULTIPLES COLUMNAS DE EMPAREJAMIENTO

7.4. CONSULTAS DE TRES O MAS TABLAS

7.5. OTRAS EQUICOMPOSICIONES

7.6 COMPOSICION BASADAS EN DESIGUALDAD

7.7 CONSIDERACIONES DE SQL PARA CONSULTAS MULTITABLA

7.7.1 Nombres de columna cualificados

7.7.2. Selecciones de todas las columnas

7.7.3. Alias de tablas

7.7.4. Autocomposiciones

7.8. LA ESTRUCTURA DE UNA COMPOSICION

7.8.1. Multiplicación de tablas

7.8.2. Reglas para procesamiento de consultas multitabla

7.9. COMPOSICIONES EXTERNAS, INTERNA Y COMPLETA

7.10 RESUMEN

7.11. EJERCICIOS

7.1. CONSULTA DE VARIAS TABLAS

Muchas consultas útiles solicitan datos procedentes de dos o más tablas de la base de datos. Por ejemplo, estas peticiones de datos para la base de datos ejemplo extraen los datos de dos, tres o cuatro tablas:

- ♦ Lista los vendedores y las oficinas en donde trabajan (Tablas REPVENTAS y OFICINAS).
- ♦ Lista los pedidos remitidos la última semana, mostrando el importe del pedido, el nombre del cliente que lo ordenó y el nombre del producto solicitado (tablas PEDIDOS, CLIENTES y REPVENTAS).

SQL permite recuperar datos que corresponden a estas peticiones mediante consultas multitabla que *componen* (*join*) datos procedentes de dos o más tablas.

Para comprender mejor las facilidades que SQL proporciona para consultas multitabla lo mejor es comenzar con una petición simple que combina datos de dos tablas diferentes:

Lista todos los pedidos, mostrando el número de pedido y su importe, y el nombre y el balance contable del cliente que los solicitó.

Los cuatro datos específicos solicitados están evidentemente almacenados en dos tablas diferentes:

- ♦ La tabla PEDIDOS contiene el número de pedido y el importe de cada pedido, pero no tiene los nombres de cliente ni los límites de crédito.
- ♦ La Tabla CLIENTES contiene los nombres de cliente y sus balances pero le falta la información referente a los pedidos.

Existe, sin embargo, un enlace entre estas dos tablas. En cada fila de la Tabla PEDIDOS, la columna CLIE contiene el número del cliente que ordenó el pedido, el cual se corresponde con el valor en la columna NUM_CLIE de una de las filas de la Tabla CLIENTES. Evidentemente, la sentencia SELECT que gestiona la petición debe utilizar de algún modo este enlace entre tablas para generar sus resultados.

Antes de examinar la sentencia SELECT correspondiente a la consulta, es instructivo imaginar cómo podríamos gestionar manualmente la petición, utilizando lápiz y papel.

- Comenzar escribiendo los nombres de las cuatro columnas para los resultados de la consulta. Luego pasar a la Tabla PEDIDOS y comenzar con el primer pedido.
- Recorrer la fila para hallar el número de pedido (112.961) y el importe ($31,500.00) y copiar ambos valores en la primera fila de resultados de la consulta.
- Recorrer la fila para hallar el número de cliente que remitió el pedido (2.117), y pasar a la Tabla CLIENTES para hallar el número de cliente 2.117 buscándolo en la columna NUM_CLIE.

- Recorrer la fila de la Tabla CLIENTES para hallar el nombre del cliente (J.p. Sinclair) y su límite de crédito ($35,000.00), y copiarlos a la tabla de resultados.
- Ya se ha generado una fila de resultados de la consulta. Se regresa a la Tabla PEDIDOS y se continúa con la fila siguiente. El proceso se repite, comenzando por el paso 2, hasta que se agotan los pedidos.

Naturalmente ésta no es la única manera de generar los resultados de la consulta, pero independientemente de cómo se haga, dos cosas serán ciertas:

- ♦ Cada fila de resultados de la consulta extraerá sus datos de un *par* específico de filas, una de la Tabla PEDIDOS y otra de la Tabla CLIENTES.
- ♦ El par de filas en cuestión se determinará haciendo coincidir los contenidos de las columnas correspondientes de las tablas.

COMPOSICIONES SIMPLES (EQUICOMPOSICIONES)

El proceso de formar parejas de filas haciendo coincidir los contenidos de las columnas relacionadas se denomina *componer* las tablas. La tabla resultante (que contiene datos de las dos tablas originales) se denomina una *composición* entre las dos tablas. (una composición basada en una coincidencia exacta entre dos columnas se denomina más precisamente una *equicomposición.*

Las composiciones son el fundamento del procesamiento e consultas multitabla en SQL. Todos los datos de una base de datos relacional están almacenados en sus columnas con valores explícitos, de modo que todas las relaciones posibles entre tablas pueden formarse comparando los contenidos de las columnas relacionadas.

Lista todos los pedidos mostrando su número, importe, número de cliente y el límite de crédito del cliente.

```
SELECT NUM_PEDIDO, IMPORTE, EMPRESA, LIMITE_CREDITO

FROM PEDIDOS, CLIENTES

WHERE CLIE = NUM_CLIE
```

Esta tiene el mismo aspecto que las consultas del capítulo anterior, con dos características novedosas. Primero, la cláusula FROM lista dos tablas en lugar de una sola. En segundo lugar, la condición de búsqueda:

```
CLIE = NUM_CLIE
```

Compara columnas de dos tablas diferentes. A estas dos columnas las denominamos las *columnas de emparejamiento* para las dos tablas. Como todas las condiciones de búsqueda, ésta restringe las filas que aparecen en los resultados de la consulta. Puesto que ésta es una consulta de dos tablas, la condición de búsqueda restringe las parejas de filas que generan los resultados. Realmente captura el espíritu de la correspondencia manual de las columnas muy bien, diciendo:

Genera resultados sólo para los pares de filas en los que el número de clientes (CLIE) en la Tabla PEDIDOS coincide con el número de cliente (NUM_CLIE) en la Tabla CLIENTES.

La sentencia SELECT no dice nada acerca de cómo SQL debería ejecutar la consulta. No hay mención de comenzar con los pedidos o comenzar con los clientes. En lugar de ello, la consulta dice a SQL *qué* resultados deberían aparecer, y deja a SQL que decida cómo generarlos.

Consultas padre/hijo

Las consultas multitabla más comunes implican a dos tablas que tienen una relación natural padre/hijo. La consulta referente a pedidos y clientes de la sección precedente es un ejemplo de tal tipo de consulta. Cada pedido (hijo) tiene un cliente asociado (padre), y cada cliente (padre) puede tener muchos pedidos asociados (hijos). Los pares de filas que generan los resultados de la consulta son combinaciones de fila pdre/hijo.

Se puede recordar que las claves ajenas y las claves primarias crean relaciones padre/hijo en una base de datos SQL. La tabla que contiene la clave ajena es el hijo en la relación; la tabla con la clave primaria es el padre. Para ejercitar la relación padre/hijo en una consulta debe especificarse una condición de búsqueda que compare la clave ajena y la clave primaria. Lista cada uno de los vendedores y la ciudad y región en donde trabajan.

SELECT NOMBRE, CIUDAD, REGION

FROM REPVENTAS, OFICINAS

WHERE OFICINA_REP = OFICINA

NOMBRE CIUDAD REGION

Mary Jones New York Este

Sam Clark New York Este

Bob Smith Chicago Este

Paul Cruz Chicago Este

Dan Roberts Chicago Este

Bill Adams Atlanta Este

Sue Smith Los Angeles Oeste

Larry Fitch Los Angeles Oeste

Nancy Angelli Denver Oeste

Tabla OFICINAS

OFICINA	CIUDAD	REGION	OBJETIVO	VENTAS
22	Denver	Oeste	$300,000.00	$186,042.00
11	New York	Este	$575,000.00	$692,637.00
12	Chicago	Este	$800,000.00	$735,042.00
13	Atlanta	Este	$350,000.00	$350,000.00
21	Los Angeles	Oeste	$725,000.00	$835,915.00

Tabla REPVENTAS Resultados de la consulta

NUM_EMPL	NOMBRE	ED	OFIC_REP	TITULO		NOMBRE	CIUDAD	REGION
105	Bill Adams	37	13	Rep Ventas				
109	Mary Jones	31	11	Rep Ventas				
102	Sue Smith	48	21	Rep Ventas				
106	Sam Clark	52	11	VP Ventas				
104	Bob Smith	33	12	Dir Ventas				
101	Dan Roberts	45	12	Rep Ventas				
110	Tom Snyder	41	NULL	Rep Ventas				
108	Larry Fitch	62	21	Dir Ventas				
103	Paul Cruz	29	12	Rep Ventas				
107	Nancy Angelli	49	22	Rep Ventas				

La Tabla REPVENTAS (hijo) contiene PFICINA_REP, una clave ajena para la Tabla OFICINAS (padre). Esta relación se utiliza para hallar la fila OFICINA correcta para cada vendedor, de modo que pueda incluirse la ciudad y región correctas en los resultados de la consulta.

He aquí otra consulta que referencia las mismas dos tablas, pero con los papeles de padre e hijo invertidos.

Lista las oficinas y los nombres y títulos de sus directores.

SELECT CIUDAD, NOMBRE, TITULO

FROM OFICINAS, REPVENTAS

WHERE DIR = NUM_EMPL

CIUDAD NOMBRE TITULO

Chicago Bob Smith Dir Ventas

Atlanta Bill Adams Rep Ventas

New York Sam Clark VP Ventas

Denver Larry Fitch Dir Ventas

Los Angeles Larry Fitch Dir Ventas

La Tabla OFICINAS (hijo) contiene DIR, una clave ajena para la Tabla REPVENTAS (padre). Esta relación se utiliza para hallar la fila REPVENTAS correcta para cada vendedor, de modo que puedan incluirse el nombre y el título correctos del director en los resultados de la consulta.

7.2. COMPOSICIONES CON CRITERIOS DE SELECCIÓN DE FILA

La condición de búsqueda que especifica las columnas de emparejamiento en una consulta multitabla puede combinarse con otras condiciones de búsqueda para restringir aún más los contenidos de los resultados. Supongamos que se desea volver a ejecutar la consulta anterior, mostrando únicamente las oficinas con mayores objetivos de ventas.

Lista las oficinas con un objetivo superior a $600.000 y su director.

```
SELECT CIUDAD, NOMBRE, TITULO

FROM OFICINAS, REPVENTAS

WHERE DIR = NUM_EMPL

AND OBJETIVO > 600,000.00
```

CIUDAD NOMBRE TITULO

Chicago Bob Smith Dir Ventas

Con la condición de búsqueda adicional, las filas que aparecen en los resultados están aún más restringidas. El primer test (DIR = NUM_EMPL) selecciona solamente pares de filas OFICINAS y REPVENTAS que tienen la adecuada relación padre/hijo; el segundo test selecciona adicionalmente sólo aquellos pares de filas en donde la oficina está por encima del objetivo.

7.3. MULTIPLES COLUMNAS DE EMPAREJAMIENTO

La Tabla PEDIDOS y la Tabla PRODUCTOS de la base de datos ejemplo están relacionadas por un par de claves ajena/primaria. Las columnas FAB y PRODUCTO de la Tabla PEDIDOS forman juntas una clave ajena para la Tabla PRODUCTOS, respectivamente. Para componer las tablas basándose en esta relación padre/hijo, deben especificarse ambos pares de columnas de emparejamiento, tal como se muestra en este ejemplo:

Lista los pedidos, mostrando los importes y las descripciones del producto.

```
SELECT NUM_PEDIDO, IMPORTE, DESCRIPCION

FROM PEDIDOS, PRODUCTOS

WHERE FAB = ID_FAB

AND PRODUCTO = ID_PRODUCTO
```

NUM_PEDIDO IMPORTE DESCRIPCION

113027 $4,104.00 Artículo Tipo 2

112992 $760.00 Artículo Tipo 2

113012 $3,745.00 Artículo Tipo 3

112968 $3,978.00 Artículo Tipo 4

112963 $3,276.00 Artículo Tipo 4

112983 $702.00 Artículo Tipo 4

113055 $150.00 Ajustador

113057 $600.00 Ajustador

La condición de búsqueda en la consulta dice a SQL que los pares de filas relacionados en las Tablas PEDIDOS y PRODUCTOS son aquéllas en las que ambos pares de columnas coincidentes contienen los mismos valores.

7.4. CONSULTAS DE TRES O MAS TABLAS

SQL puede combinar datos de tres o más tablas utilizando las mismas técnicas básicas utilizadas para las consultas de dos tablas. He aquí un sencillo ejemplo de una composición con tres tablas:

Lista los pedidos superiores a $25.000, incluyendo el nombre del vendedor que tomó el pedido y el nombre del cliente que lo solicitó.

```
SELECT NUM_PEDIDO, IMPORTE, EMPRESA, NOMBRE

FROM PEDIDOS, CLIENTES, REPVENTAS

WHERE CLIE = NUM_CLIE

AND REP = NUM_EMPL

AND IMPORTE > 25000.00
```

NUM_PEDIDO IMPORTE EMPRESA NOMBRE

112987 $27,500.00 Acme Mfg. Bill Adams

113069 $31,350.00 Chen Associates Nancy Angelli

113045 $45,000.00 Zetacorp Larry Fitch

112961 $31,500.00 J. P. Sinclair Sam Clark

Esta consulta utiliza dos claves de la Tabla PEDIDOS. La columna CLIE es una clave ajena para la Tabla CLIENTES, que enlaza cada pedido con el cliente que lo remitió. La columna REP es una clave ajena para la Tabla REPVENTAS, que liga cada pedido con el vendedor que lo aceptó.

Tabla CLIENTES Tabla REPVENTAS

NUM_CLIE	EMPRESA	REP_CLIE	LIMITE_CREDITO	NUM_EMPL	NOMBRE	ED	OFIC_REP
2111	JCP inc.	103	$50,000.00	105	Bill Adams	37	13
2102	First Corp.	101	$65,000.00	109	Mary Jones	31	11
2103	Acme Mfg.	105	$50,000.00	102	Sue Smith	48	21
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

Tabla PEDIDOS

NUM_PEDIDO	FECHA_PED	CLIE	REP	DIR	...	IMPORTE
112961	12/17/1989	2117	106	REI	...	$31,500.00
113012	01/11/1990	2111	105	ACI	...	$3,745.00
112989	01/03/1990	2101	106	FEA	...	$1,458.00
.	.	.	.	.	...	.
.	.	.	.	.	...	.

Resultados de la consulta

NUM_PEDIDO	IMPORTE	EMPRESA	NOMBRE

No es extraño encontrar consultas de tres o incluso cuatro tablas en aplicaciones SQL de producción.

7.5. OTRAS EQUICOMPOSICIONES

La inmensa mayoría de las consultas multitabla se basan en relaciones padre/hijo, pero SQL no exige que las columnas de emparejamiento están relacionadas como clave primaria y clave ajena. Cualquier par de columnas de dos tablas pueden servir como columnas de emparejamiento, siempre que tengan tipos de datos comparables. He aquí un ejemplo de una columna que utiliza un par de fechas como columnas de emparejamiento:

Halla todos los pedidos recibidos en los días en que un nuevo vendedor fue contratado.

SELECT NUM_PEDIDO, IMPORTE, FECHA_PEDIDO, NOMBRE

FROM PEDIDOS, REPVENTAS

WHERE FECHA_PEDIDO = CONTRATO

NUM_PEDIDO IMPORTE FECHA_PEDIDO NOMBRE

112968 $3,978.00 12−OCT−89 Mary Jones

112979 $15,000.00 12−OCT−89 Mary Jones

112975 $2,100.00 12−OCT−89 Mary Jones

112968 $3,978.00 12−OCT−89 Larry Fitch

112979 $15,000.00 12−OCT−89 Larry Fitch

112975 $2,100.00 12−OCT−89 Larry Fitch

Los resultados de esta consulta provienen de los pares de filas de las Tablas PEDIDOS y REPVENTAS, en donde el valor en la columna FECH_PEDIDO coincide con el valor en la columna CONTRATO para el vendedor. Ninguna de estas columnas es una clave ajena o una clave primaria, y la relación entre los pares de filas es ciertamente una relación extraña, la única cosa que los pedidos y vendedores correspondientes tienen en común es que resultan tener las mismas fechas.

Tabla REPVENTAS

NUM_EMPL	NOMBRE	...	CONTRATO		NUM_PEDIDO	FECHA_PEDIDO	CLIE
105	Bill Adams	...	02/12/1988		113051	02/10/1990	2118
109	Mary Jones	...	10/12/1989		112968	10/12/1989	2102
102	Sue Smith	...	12/10/1986		113036	01/30/1990	2107
106	Sam Clark	...	06/14/1988		113062	02/24/1990	2124
104	Bob Smith	...	05/19/1987		112979	10/12/1989	2114
101	Dan Roberts	...	10/20/1986		113027	01/22/1990	2103
110	Tom Snyder	...	01/13/1990		112992	11/04/1989	2118
108	Larry Fitch	...	10/12/1989		112975	10/12/1989	2111
103	Paul Cruz	...	03/01/1987		113055	02/15/1990	2108
107	Nancy Angelli	...	11/14/1989		.	.	.

Las columnas de emparejamiento como las de este ejemplo generan una relación de muchos a muchos entre las dos tablas. Puede haber muchos pedidos que compartan una única fecha de contrato del vendedor, y más de un vendedor puede haber sido contratado en la fecha de pedidos diferentes (112.968, 112.975 y 112.979) fueron recibidos el 12 de octubre de 1989, y dos vendedores diferentes (Larry Fitch y Mary Jones) fueron contratados el mismo día. Los tres pedidos y los dos vendedores producen seis filas de resultados de la consulta.

Esta relación de muchos a muchos es diferente de la relación de uno a muchos creada por columnas de emparejamiento clave primaria/clave ajena. La situación puede resumirse del siguiente modo:

- Las composiciones que asocian claves primarias a claves ajenas siempre crean relaciones padre/hijo de uno a muchos.
- En general, las composiciones sobre las columnas de emparejamiento arbitrarias generan relaciones de muchos a muchos.

Estas dos situaciones diferentes no tienen nada que evr con cómo se escriba la sentencia SELECT que expresa la composición.

7.6. COMPOSICIONES BASADAS EN DESIGUALDAD

El término composición (*join)* se aplica a cualquier consulta que combina datos de dos tablas mediante comparación de los valores en una pareja de columnas de tablas.

Lista todas las combinaciones de vendedores y oficinas en donde la cuota del vendedor es superior al objetivo de la oficina.

SELECT NOMBRE, CUOTA, CIUDAD, OBJETIVO

FROM REPVENTAS, OFICINAS

WHERE CUOTA > OBJETIVO

NOMBRE CUOTA CIUDAD OBJETIVO

Bill Adams $350,000.00 Denver $300,000.00

Sue Smith $350,000.00 Denver $300,000.00

Larry Fitch $350,000.00 Denver $300,000.00

Como en todas las consultas de dos tablas, cada fila de resultados proviene de un par de filas, en este caso de las Tablas REPVENTAS y OFICINAS. La condición de búsqueda:

CUOTA > OBJETIVO

Selecciona pares de filas en donde la columna CUOTA de la fila REPVENTAS excede a la columna OBJETIVO de la fila OFICINAS. Observe que los pares de filas REPVENTAS y OFICINAS están relacionadas únicamente de este modo; no se requiere específicamente que la fila REPVENTAS represente a alguien que trabaje en la oficina representada por la fila OFICINAS. El ejemplo es un poco extremado, e ilustra por qué las composiciones basadas en desigualdades no son muy comunes.

7.7. CONSIDERACIONES SQL PARA CONSULTAS MULTITABLA

Algunas consultas multitabla no pueden ser expresadas sin las características adicionales del lenguaje SQL descritas en las secciones siguientes. Específicamente:

- Los *nombres de columna cualificados* son necesarios a veces en consultas multitabla para eliminar referencias de columna ambiguas.
- Las *selecciones de todas las columnas* (SELECT *) tienen un significado especial para las consultas multitabla.
- Las *autocomposiciones* pueden ser utilizadas para crear una consulta multitabla que relaciona una tabla consigo misma.
- Los *alias de tablas* pueden ser utilizadas en la cláusula FROM para simplificar nombres de columna cualificados y permitir referencias de columna no ambiguas en autocomposiciones.

7.7.1. Nombres de columna cualificados

La base de datos ejemplo incluye varias instancias en donde dos tablas contienen columnas con el mismo nombre. La Tabla OFICINAS y la Tabla REPVENTAS, por ejemplo, tienen ambas una columna de nombre VENTAS. La columna de la Tabla OFICINAS contiene las ventas anuales hasta la fecha para cada oficina; la de la Tabla REPVENTAS contiene las ventas anuales hasta la fecha de cada vendedor. Normalmente, no hay confusión entre ambas columnas, ya que la cláusula FROM determina cuál de ellas es la adecuada en una consulta determinada, como en estos ejemplos:

Muestra las ciudades en donde las ventas superan al objetivo.

SELECT CIUDAD, VENTAS

FROM OFICINAS

WHERE VENTAS > OBJETIVO

Sin embargo, he aquí una consulta en donde los nombres duplicados provocan un problema:

Muestra el nombre, las ventas y la oficina de cada vendedor.

SELECT NOMBRE, VENTAS, CIUDAD

FROM REPVENTAS, OFICINAS

WHERE OFICINA_REP = OFICINA

Error: Nombre de columna ambiguo VENTAS

Aunque la descripción textual de la consulta implica que se desea la columna VENTAS de la tabla REPVENTAS, la consulta SQL es ambigua. Para eliminar la ambigüedad, debe utilizarse un nombre de columna cualificado para identificar la columna. Un nombre de columna cualificado especifica el nombre de una columna y la tabla que contiene a la columna. Los nombres cualificados de las dos columnas VENTAS en base de datos ejemplo son:

OFICINAS, VENTAS y REPVENTAS.VENTAS

Un nombre de columna cualificado puede ser utilizado en una sentencia SELECT en cualquier lugar en donde se permite un nombre de columna. La tabla especificada en el nombre de columna cualificado, debe, naturalmente, corresponder a una de las tablas especificadas en la lista FROM. He aquí una versión corregida de la consulta anterior que utiliza un nombre de columna cualificado:

Muestra el nombre, las ventas y la oficina de cada vendedor.

```
SELECT NOMBRE, REPVENTAS.VENTAS, CIUDAD

FROM REPVENTAS, OFICINAS

WHERE OFICINA_REP = OFICINA
```

Utilizar nombres de columnas cualificados en una consulta multitabla es siempre una buena medida. La desventaja, naturalmente, es que hacen que el texto de la consulta sea mayor.

7.7.2. Selecciones de todas las columnas

SELECT * puede ser utilizado para seleccionar todas las columnas de la tabla designada en la cláusula FROM. En una consulta multitabla, el asterisco selecciona todas las columnas de todas las tablas listadas en la cláusula FROM. La siguiente consulta, por ejemplo, produciría 15 columnas de resultados, las 9 columnas de la Tabla REPVENTAS seguidas de las 6 columnas de la Tabla OFICINAS:

Informa sobre todos los vendedores y todas las oficinas en las que trabajan.

```
SELECT *

FROM REPVENTAS, OFICINAS

WHERE OFICINA_REP = OFICINA
```

Obviamente, la forma SELECT * de una consulta resulta ser mucho menos práctica cuando hay dos, tres o más tablas en la cláusula FROM.

En la siguiente consulta, el ítem de selección REPVENTAS.* se expande a una lista que contiene únicamente las columnas halladas en la Tabla REPVENTAS:

Informa acerca de todos los vendedores y los lugares en los que trabajan.

```
SELECT REPVENTAS.*, CIUDAD, REGION

FROM REPVENTAS, OFICINAS
```

WHERE OFICINA_REP = OFICINA

La consulta produciría once columnas de resultados, las nueve columnas de la Tabla REPVENTAS, seguidas de las dos columnas explícitamente solicitadas de la Tabla OFICINAS.

7.7.3. Alias de tablas

Si una consulta se refiere a la tabla de otro usuario, o si el nombre de una tabla es muy largo, puede ser tedioso escribir el nombre de la tabla como cualificador de una columna. Esta consulta, que referencia a la tabla CUMPLEAÑOS propiedad del usuario SAM:

Lista los nombres, cuotas y cumpleaños de los vendedores.

SELECT REPVENTAS, NOMBRE, CUOTA, SAM.CUMPLEAÑOS_FECHA

FROM REPVENTAS, SAM.CUMPLEAÑOS

WHERE REPVENTAS.NOMBRE = SAM.CUMPLEAÑOS.NOMBRE

Resulta más fácil de leer y escribir cuando se utilizan los alias S y B para las dos tablas:

Lista los nombres, cuotas y cumpleaños de los vendedores.

SELECT S.NOMBRE, S.CUOTA, B.FECHA

FROM REPVENTAS S, SAM.CUMPLEAÑOS B

WHERE S.NOMBRE = B.NOMBRE

La cláusula tiene dos funciones importantes:

- La cláusula FROM identifica todas las tablas que contribuyen con datos a los resultados de la consulta. Las columnas referenciadas en la sentencia SELECT deben provenir de una de las tablas designadas en la cláusula FROM.
- La cláusula FROM determina la marca que se utiliza para identificar la tabla en referencias de columna cualificadas dentro de la sentencia SELECT. Si se especifica un alias, éste pasa a ser la marca de la tabla; en caso contrario se utiliza como marca el nombre de la tabla, tal como aparece en la cláusula FROM.

La única exigencia para las marcas de tablas en la cláusula FROM es que todas las marcas de una cláusula FROM determinada deben ser distintas unas de otras.

7.7.4. Autocomposiciones (una tabla consigo misma)

Algunas consultas multitabla afectan a una relación que una tabla tiene consigo misma. Por ejemplo, supongamos que se desea listar los nombres de todos los vendedores y sus directores. Cada vendedor aparece como una fila en la Tabla REPVENTAS, y la columna DIRECTOR contiene el número de empleado del director del vendedor. Parecería que la columna DIRECTOR debería ser una clave ajena para la tabla que contiene datos referentes a los directores. En efecto así es, se trata de una clave ajena para la propia tabla REPVENTAS.

Si se tratara de expresar esta consulta como cualquier otra consulta de dos tablas implicando una coincidencia clave ajena/clave primaria, aparecería tal como esta:

```
SELECT NOMBRE, NOMBRE

FROM REPVENTAS, REPVENTAS

WHERE DIRECTOR = NUM_EMPL
```

Esta sentencia SELECT es ilegal debido a la referencia duplicada a la Tabla REPVENTAS en la cláusula FROM. También podría intentarse eliminar la segundareferencia a la Tabla REPVENTAS.

```
SELECT NOMBRE, NOMBRE

FROM REPVENTAS

WHERE DIRECTOR = NUM_EMPL
```

Esta consulta es legal, pero no hará lo que se desea que haga. Es una consulta monotabla, por lo que SQL recorre la Tabla REPVENTAS fila a fila, aplicando la condición de búsqueda.

```
DIRECTOR = NUM_EMPL
```

Las filas que satisfacen esta condición son aquéllas en donde las dos columnas tienen el mismo valor, es decir, las filas en donde un vendedor es su propio director. No existen tales filas, por lo que la consulta no produciría resultados.

Para comprender cómo resuelve SQL este problema, imagínate que hubiera *dos copias idénticas* de la Tabla REPVENTAS, una llamada EMPS, que contuviera los empleados, y otra llamada DIRS, que contuviera los directores. La columna DIRECTOR de la Tabla EMPS sería entonces una clave ajena para la Tabla DIRS, y la siguiente consulta funcionaría:

Lista los nombres de los vendedores y sus directores.

```
SELECT EMPS.NOMBRE, DIRS.NOMBRE

FROM EMPS, DIRS

WHERE EMPS.DIRECTOR = DIRS.NUM_EMPL
```

Puesto que las columnas de las dos tablas tienen nombres idénticos, todas las referencias de columnas están cualificadas.

SQL utiliza exactamente esta estrategia de tabla duplicada imaginaria para componer una tabla consigo misma. En lugar de duplicar realmente el contenido de la tabla. SQL simplemente permite referirse a ella mediante un nombre diferente, llamado un alias de tabla. He aquí la misma consulta, escrita utilizando los alias EMPS y DIRS para la Tabla REPVENTAS.

Lista los nombres de los vendedores y sus directores.

```
SELECT EMPS.NOMBRE, DIRS.NOMBRE

FROM REPVENTAS EMPS, REPVENTAS DIRS

WHERE EMPS.DIRECTOR = DIRS.NUM_EMPL
```

EMPS.NOMBRE DIRS.NOMBRE

Tom Snyder Dan Roberts

Bill Adams Bob Smith

Dan Roberts Bob Smith

Paul Cruz Bob Smith

Mary Jones Sam Clark

Bob Smith Sam Clark

Larry Fitch Sam Clark

Sue Smith Larry Fitch

Nancy Angelli Larry Fitch

La cláusula FROM asigna un alias copia de la tabla REPVENTAS especificando el nombre del alias inmediatamente después del nombre real de la tabla. Como muestra el ejemplo, cuando una cláusula FROM contiene un alias, el alias debe ser utilizado para identificar la tabla en referencias de columna cualificadas.

Lista los vendedores con una cuota superior a la de sus directores.

```
SELECT REPVENTAS, NOMBRE, REPVENTAS.CUOTA, DIRS.CUOTA

FROM REPVENTAS, REPVENTAS DIRS

WHERE REPVENTAS.DIRECTOR = DIRS.NUM_EMPL

AND REPVENTAS.CUOTA > DIRS.CUOTA
```

7.8. LA ESTRUCTURA DE UNA COMPOSICION

Cuando se componen muchas tablas o cuando las condiciones de búsqueda resultan complejas, sin embargo, es muy difícil tan sólo mirando una sentencia SELECT imaginar lo que significa. Por esta razón, es muy útil definir más cuidadosamente lo que es una composición y qué resultados se producen con una sentencia SELECT determinada.

7.8.1. Multiplicación de tablas

Una composición es un caso especial de una combinación más general de datos procedentes de dos tablas, conocida como el *producto cartesiano* de dos tablas. El producto de dos tablas es otra tabla (la *tabla producto*); que consta de todos los pares posibles de filas de las dos tablas. Las columnas de la *tabla producto* son todas las columnas de la primera tabla, seguidas de todas las columnas de la segunda tabla.

Si se especifica una consulta de dos Tablas sin una cláusula WHERE, SQL produce el producto de las dos tablas como resultado. Por ejemplo, esta consulta:

Muestra todas las combinaciones posibles de vendedores y ciudades.

SELECT NOMBRE, CIUDAD

FROM REPVENTAS, OFICINAS

Produciría el producto de las Tablas REPVENTAS y OFICINAS, mostrando todos los pares posibles vendedor/ciudad. Habría 50 filas de resultados (5 oficinas * 10 vendedores = 50 combinaciones).

Tabla CHICAS

NOMBRE	CIUDAD
Mary	Boston
Susan	Chicago
Betty	Chicago

NOMBRE	CIUDAD
Sam	Chicago
James	Dallas

NOMBRE.CHICAS	CIUDAD.CHICAS	NOMBRE.CHICOS	CIUDAD.CHICOS
Mary	Boston	Sam	Chicago
Susan	Chicago	Sam	Chicago
Betty	Chicago	Sam	Chicago
Mary	Boston	James	Dallas
Susan	Chicago	James	Dallas
Betty	Chicago	James	Dallas

7.8.2. Reglas para procesamiento de consultas multitabla

Las reglas definen el significado de cualquier sentencia SELECT multitabla especificando un procesamiento que siempre genera el conjunto correcto de resultados de la consulta. Para generar los resultados de una consulta con una sentencia SELECT:

- Si la sentencia es una UNION de sentencias SELECT, aplicar los pasos 2 hasta 5 a cada una de las sentencias para generar sus resultados individuales.
- Formar el producto de las tablas indicadas en la cláusula FROM. Si la cláusula FROM designa una sola tabla, el producto es esa tabla.
- Si hay una cláusula WHERE, aplicar su condición de búsqueda a cada fila de la tabla producto reteniendo aquellas filas para las cuales la condición de búsqueda es TRUE (y descartando aquellas para las cuales es FALSE O NULL).
- Para cada fila restante, calcular el valor de cada elemento en la lista de selección para producir una única fila de resultados. Por cada referencia de columna, utilizar el valor de la columna en la fila actual.
- Si se especifica SELECT DISTINCT, eliminar las filas duplicadas de los resultados que se hubieran producido.
- Si la sentencia es una UNION de sentencia SELECT, mezclar los resultados de consulta para las sentencias individuales en una única tabla de resultados. Eliminar las filas duplicadas a menos que se haya especificado UNION ALL:
- Si hay una cláusula ORDER BY, ordenar la formación de los resultados de la consulta.

Las filas generadas por este procedimiento forman los resultados de la consulta. Para ver cómo funciona el procedimiento, considérese esta consulta:

Lista el nombre de la empresa y todos los pedidos para el número de cliente 2.103.

SELECT EMPRESA, NUM_PEDIDO, IMPORTE

FROM CLIENTES, PEDIDOS

WHERE NUM_CLIE = CLIE

AND NUM_CLIE = 2103

ORDER BY NUM_PEDIDO

EMPRESA NUM_PEDIDO IMPORTE

Acme Mfg. 112963 $3,276.00

Acme Mfg. 112983 $702.00

Acme Mfg. 112987 $27,500.00

Acme Mfg. 113027 $4,104.00

Siguiendo los pasos:

- La cláusula FROM genera todas las combinaciones posibles de filas de la Tabla CLIENTES (21 filas) y de la Tabla PEDIDOS (30 filas), produciendo una Tabla PRODUCTO de 630 filas.
- La cláusula WHERE selecciona únicamente aquellas filas de la Tabla PRODUCTO en donde los números de cliente coinciden (NUM_CLIE = CLIE) y el número de cliente es el especificado (NUM_CLIE = 2103). Solamente se seleccionan cuatro filas; las otras 626 se eliminan.
- La cláusula SELECT extrae las tres columnas solicitadas (EMPRESA, NUM_PEDIDO e IMPORTE_RED) de cada fila que queda de la Tabla PRODUCTO para generar cuatro filas de resultados detallados.
- La cláusula ORDER BY ordena las cuatro filas por la columna NUM_PEDIDO para generar el resultado final.

7.9. COMPOSICIONES EXTERNAS, INTERNA Y COMPLETA

La operación de composición SQL combina información procedente de dos tablas mediante la formación de *pares* de filas relacionadas en las dos tablas. Los pares de filas que componen la tabla compuesta son aquéllos en los que las columnas asociadas en cada una de las dos tablas tienen el mismo valor. Si una de las filas de una tabla no se empareja en este proceso, la composición puede producir resultados inesperados, como muestran estas consultas:

Lista los vendedores y las oficinas en que trabajan:

SELECT NOMBRE, OFICINA_REP

FROM REPVENTAS

NOMBRE OFICINA_REP

Bill Adams 13

Mary Jones 11

Sue Smith 21

Sam Clark 11

Bob Smith 12

Dan Roberts 12

Tom Snyder NULL

Larry Fitch 21

Paul Cruz 12

Nancy Angelli 22

Lista los vendedores y las ciudades en que trabajan.

SELECT NOMBRE, CIUDAD

FROM REPVENTAS, OFICINAS

WHERE OFICINA_REP = OFICINA

NOMBRE CIUDAD

Mary Jones New York

Sam Clark New York

Bob Smith Chicago

Paul Cruz Chicago

Dan Roberts Chicago

Bill Adams Atlanta

Sue Smith Los Angeles

Larry Fitch Los Angeles

Nancy Angelli Denver

Aparentemente, sería de esperar que estas dos consultas produjeran el mismo número de filas, pero la primera lista diez vendedores, y la segunda sólo nueve. ¿Por qué? Porque Tom Snyder está actualmente sin asignar y tiene un valor NULL en la columna OFICINAS__REP (que es la columna de emparejamiento para la composición). Este valor NULL no se corresponde con ninguno de los números de oficina en la Tabla OFICINAS, por lo que la oficina de Tom en la Tabla REPVENTAS está sin emparejar. Como resultado, desaparece de la composición. La composición SQL estándar tiene por tanto el potencial de perder información si las tablas que se componen contienen filas sin emparejar.

Lista los vendedores y las ciudades en que trabajan.

SELECT NOMBRE, CIUDAD

FROM REPVENTAS, OFICINAS

WHERE OFICINA_REP * = OFICINA

NOMBRE CIUDAD

Tom Snyder NULL

Mary Jones New York

Sam Clark New York

Bob Smith Chicago

Paul Cruz Chicago

Dan Roberts Chicago

Bill Adams Atlanta

Sue Smith Los Angeles

Larry Fitch Los Angeles

Nancy Angelli Denver

Estos resultados se generan utilizando un tipo diferente de operación de composición, llamada *composición externa* (indicada por la notación * en la cláusula WHERE). La composición externa es una extensión de la composición estándar descrita con anterioridad, la cual a veces se denomina *composición interna.*

Tabla CHICAS Tabla CHICOS

NOMBRE	**CIUDAD**			**NOMBRE**	**CIUDAD**
Mary	Boston			John	Boston
Nancy	NULL			Henry	Boston
Susan	Chicago			George	NULL
Betty	Chicago			Sam	Chicago
Anne	Denver			James	Dallas

Tabla de composición externa

		NOMBRE.CHICAS	**CIUDAD.CHICAS**	**NOMBRE.CHICOS**	**CIUDAD.CHICOS**
		Mary	Boston	John	Boston
		Mary	Boston	Henry	Boston
		Susan	Chicago	Sam	Chicago

		Betty	Chicago	Sam	Chicago
Filas sin	**emparejar**	Anne	Denver	NULL	NULL
		Nancy	NULL	NULL	NULL
		NULL	NULL	James	Dallas
		NULL	NULL	George	NULL

La Tabla CHICAS lista cinco chicas y las ciudades en donde viven; la Tabla CHICOS lista cinco chicos y las ciudades en donde viven. Para encontrar las parejas chico/chica que viven en la misma ciudad, se podría utilizar esta consulta, que forma la composición interna de las dos tablas:

Lista los chicos y chicas que viven en la misma ciudad.

SELECT *

FROM CHICAS, CHICOS

WHERE CHICAS.CIUDAD = CHICOS.CIUDAD

La composición interna produce cuatro filas de resultados de la consulta. Obsérvese que dos de las chicas (Anne y Nancy) y dos de los chicos (James y George) no están presentados en los resultados. Estas filas no pueden emparejarse con ninguna fila de la otra tabla, por tanto faltan en los resultados de la composición interna. Dos de las filas sin emparejar (Anne y James) tienen valores válidos en sus columnas CIUDAD, pero no coinciden con ninguna de las ciudades de la tabla opuesta. Las otras dos filas sin emparejar (Nancy y George) tienen valores NULL en sus columnas CIUDAD, y por las reglas de gestión de NULL, en SQL, el valor NULL no coincide con ningún otro valor (ni siquiera con otro valor NULL).

Supóngase que se desea listar los pares chico/chica que comparten las mismas ciudades, incluyendo los chicos y chicas desemparejados. La composición externa de las Tablas CHICAS y CHICOS produce exactamente este resultado. He aquí la sentencia SQL que produce la composición externa:

Lista las chicas y chicos de la misma ciudad, incluyendo las chicas y chicos desemparejados.

SELECT *

FROM CHICAS, CHICOS

WHERE CHICAS.CIUDAD *=* CHICOS.CIUDAD

CHICAS.NOMBRE CHICAS.CIUDAD CHICOS.NOMBRE CHICOS.CIUDAD

Mary Boston John Boston

Mary Boston Henry Boston

Susan Chicago Sam Chicago

Betty Chicago Sam Chicago

Anne Denver NULL NULL

Nancy NULL NULL NULL

NULL NULL James NULL

NULL NULL GEORGE NULL

- Comenzar con la composición interna de las dos tablas, utilizando columnas de emparejamiento del modo normal.
- Por cada fila de la primera tabla que no haya correspondido a ninguna fila de la segunda tabla, añadir una fila a los resultados, utilizando los valores de las columnas de la primera tabla, y suponiendo un valor NULL, para todas las columnas de la segunda tabla.
- Por cada fila de la segunda tabla que no haya correspondido a ninguna de la primera tabla, añadir una fila a los resultados, utilizando los valores de las columnas de la segunda tabla, y suponiendo un valor NULL para todas las columnas de la primera tabla.
- La tabla resultante es la composición externa de las dos tablas.

Composiciones externas izquierda y derecha

Técnicamente, la composición externa producida por la consulta anterior se denomina composición *externa completa* de las dos Tablas. Ambas tablas son tratadas simétricamente en la composición externa completa. Hay otras dos composiciones externas bien definidas que no tratan a las dos tablas simétricamente.

Lista las chicas y chicos de la misma ciudad y las chicas desemparejadas.

SELECT *

FROM CHICAS, CHICOS

WHERE CHICAS.CIUDAD *= CHICOS.CIUDAD

CHICAS.NOMBRE CHICAS.CIUDAD CHICOS.NOMBRE CHICOS.CIUDAD

Mary Boston John Boston

Mary Boston Henry Boston

Susan Chicago Sam Chicago

Betty Chicago Sam Chicago

Anne Denver NULL NULL

Nancy NULL NULL NULL

La consulta produce seis filas de resultados, mostrando los pares chico/chica emparejados y las chicas desemparejadas. Los chicos desemparejados faltan en el resultado. Este tipo de SELECT es una composición externa izquierda.

Lista las chicas y chicos de la misma ciudad y las chicos desemparejadas.

SELECT *

FROM CHICAS, CHICOS

WHERE CHICAS.CIUDAD =* CHICOS.CIUDAD

CHICAS.NOMBRE CHICAS.CIUDAD CHICOS.NOMBRE CHICOS.CIUDAD

Mary Boston John Boston

Mary Boston Henry Boston

Susan Chicago Sam Chicago

Betty Chicago Sam Chicago

NULL NULL James Dallas

NULL NULL George NULL

Esta consulta también produce seis filas de resultados, mostrando los pares chico/chica coincidentes y los chicos no coincidentes. Esta vez las chicas desemparejadas faltan en el resultado. Este tipo de SELECT es una composición externa derecha.

En la práctica, las composiciones externa izquierda y derecha son más útiles que la composición externa total, especialmente en composiciones que afectan a emparejamientos clave ajena/clave primaria. En tal composición, la columna de clave ajena puede contener valores NULL, produciendo filas no emparejadas de la tabla hijo (la tabla que contiene la clave ajena). Una composición externa simétrica incluirá estas filas hijo no emparejadas en los resultados de la consulta, sin incluir además las filas padre no emparejadas.

Una composición externa puede utilizarse con cualquiera de los operadores de comparación. Por ejemplo, una composición externa izquierda utilizando una comparación mayor o igual que (>=) produciría un test de comparación como éste:

WHERE COL1 *>= COL2

7.10. RESUMEN

Este capítulo describe cómo gestiona SQL las consultas que combinan los datos de dos o más tablas:

- En una consulta multitabla, las tablas que contienen los datos son designadas en la cláusula FROM.
- Cada fila de resultados es una combinación de datos procedentes de una única fila en cada una de las tablas, y es la única fila que extrae sus datos de esa combinación particular.
- Las consultas multitabla más habituales utilizan las relaciones padre/hijo creadas por las claves primarias y claves ajenas.
- En general, las composiciones pueden construirse comparando cualquier par(es) de columnas de las dos tablas compuestas, utilizando un test de desigualdad o cualquier otro test de comparación.
- Una composición puede ser considerada como el producto de dos tablas del cual se han suprimido algunas de las filas.
- Una tabla puede componerse consigo misma; las autocomposiciones requieren el uso de alias.
- Las composiciones externas amplían la composición estándar (interna) reteniendo las filas no

emparejadas de las tablas compuestas en los resultados de la consulta.

7.11. EJERCICIOS

- Lista los pedidos superiores a 25.000, mostrando el nombre del cliente que remitió el pedido y el nombre del vendedor asignado a ese cliente, número de pedido e importe.

TEMA VIII

SUBCONSULTAS

8.1. INTRODUCCION

8.2. SUBCONSULTAS DE LA CLAUSULA WHERE

8.2.1. Referencias externas

8.2.2. Condiciones de búsqueda en subconsultas

8.2.2.1. Test de comparación subconsultas

8.2.2.2. Test de pertenencia a conjunto (IN)

8.2.2.3. Test de existencia (EXISTS)

8.2.2.4. Test cuantificados (ANY y ALL)

8.3. SUBCONSULTAS EN LA CLAUSULA HAVING

8.4. RESUMEN

8.5. RESUMEN FINAL. CONSULTAS SQL

8.6. EJERCICIOS

Notas

8.1. INTRODUCCION

La característica de subconsulta de SQL permite utilizar los resultados de una consulta como parte de otra. La característica de subconsulta es menos conocida que la característica de composición de SQL, pero juega un papel importante por tres razones:

- ♦ Una sentencia SQL con una subconsulta es frecuentemente el modo más natural de expresar una consulta, ya que se asemeja más a la descripción de la consulta en lenguaje natural.
- ♦ Las subconsultas hacen más fácil la escritura de sentencias SELECT, ya que permiten descomponer una consulta en partes (la consulta y sus subconsultas) y luego recomponer las partes.
- ♦ Hay algunas consultas que no pueden ser expresadas en el lenguaje SQL si utilizar una subconsulta.

Utilización de subconsultas

Una *subconsulta* es una consulta que aparece dentro de la cláusula WHERE o HAVING de otra

sentencia SQL. Las subconsultas proporcionan un modo eficaz y natural de gestionar peticiones de consultas que se expresan en términos de los resultados de otras consultas. He aquí un ejemplo de tal tipo de petición.

Lista las oficinas en donde el objetivo de ventas de la oficina exceden a la suma de las cuotas de los vendedores individuales.

La petición solicita una lista de oficinas de la Tabla OFICINAS, en donde el valor de la columna OBJETIVO satisface cierta condición. Parece razonable que la sentencia SELECT que expresa la consulta debería ser semejante a ésta:

```
SELECT CIUDAD

FROM OFICINAS

WHERE OBJETIVO > ???
```

El valor ??? necesita ser sustituido, y debería ser igual a la suma de las cuotas de los vendedores asignados a la oficina de cuestión. ¿Cómo se puede especificar ese valor en la consulta? Se sabe que la suma de las cuotas para una oficina específica (digamos, la oficina número 21) puede ser obtenida con esta consulta:

```
SELECT SUM(CUOTA)

FROM REPVENTAS

WHERE OFICINA_REP = 21
```

Pero ¿cómo se pueden poner los resultados de esta consulta en la consulta primera sustituyendo a los signos de interrogación? Parecería razonable comenzar con la primera consulta y reemplazar los ??? con la segunda consulta. Del modo siguiente:

```
SELECT CIUDAD

FROM OFICINAS

WHERE OBJETIVO > (SELECT SUM(CUOTA)

FROM REPVENTAS

WHERE OFICINA_REP = OFICINA)
```

De hecho, ésta es una consulta SQL correctamente construida. Por cada oficina, la consulta interna (la subconsulta) calcula la suma de las cuotas para los vendedores que trabajan en esa oficina. La consulta externa (la *consulta principal*) compara el objetivo de la oficina con el total calculado y decide si añadir la oficina a los resultados de la consulta principal. Trabajando conjuntamente, la consulta principal y la subconsulta expresan la petición original y recuperan los datos solicitados de la base de datos.

Las subconsultas SQL aparecen siempre como parte de la cláusula WHERE o la cláusula HAVING. En la cláusula WHERE, ayudan a seleccionar las filas individuales que aparecen en los resultados de la consulta. En la cláusula HAVING, ayudan a seleccionar los grupos de filas que aparecen en los

resultados de la consulta.

Como es una subconsulta:

- Una subconsulta debe producir una única columna de datos como resultado. Esto significa que una subconsulta siempre tiene un único elemento de selección en su cláusula SELECT.
- La cláusula ORDER BY no puede ser especificada en una subconsulta. Los resultados de la subconsulta se utilizan internamente por parte de la consulta principal y nunca son visibles al usuario, por lo que tiene poco sentido ordenarlas de algún modo.
- Una subconsulta no puede ser la UNION de varias sentencia SELECT diferentes; sólo se permite una única sentencia SELECT.
- Los nombres de columna que aparecen en una subconsulta pueden referirse a columnas de tablas de la consulta principal.

8.2. SUBCONSULTAS DE LA CLAUSULA WHERE

Las subconsultas suelen ser utilizadas principalmente en la cláusula WHERE de una sentencia SQL. Cuando aparece una subconsulta en la cláusula WHERE, ésta funciona como parte del proceso de selección de filas.

Lista las oficinas en donde el objetivo de ventas de la oficina excede a la suma de las cuotas de los vendedores individuales.

```
SELECT CIUDAD

FROM OFICINAS

WHERE OBJETIVO > (SELECT SUM(CUOTA)

FROM REPVENTAS

WHERE OFICINA_REP = OFICINA)
```

La consulta principal extrae sus datos de la Tabla OFICINAS, y la cláusula WHERE selecciona qué oficinas serán incluidas en los resultados de la consulta. SQL recorre las filas de la Tabla OFICINAS una a una, aplicándoles el test establecido en la cláusula WHERE. La cláusula WHERE compara el valor de la columna OBJETIVO de la fila actual con el valor producido por la subconsulta. Para examinar el valor OBJETUVO, SQL lleva a cabo la subconsulta, determinando la suma de las cuotas para los vendedores de la oficina actual. La subconsulta produce un número, y la cláusula WHERE compara el número con el valor OBJETIVO, seleccionando o rechazando la oficina actual en base a la comparación.

Tabla OFICINAS Subconsulta Tabla REPVENTAS

OFIC									

Tabla REPVENTAS..

CIUDAD	**OBJETIVO**

8.2.1. Referencias externas

Dentro del cuerpo de una subconsulta, con frecuencia es necesario referirse al valor de una columna en la fila actual de la consulta principal. Considérese una vez más la consulta del apartado anterior.

Lista las oficinas en donde el objetivo de ventas de la oficina excede a la suma de las cuotas de los vendedores individuales.

```
SELECT CIUDAD

FROM OFICINAS

WHERE OBJETIVO > (SELECT SUM(CUOTA)

FROM REPVENTAS

WHERE OFICNA_REP = OFICINA)
```

El papel de la subconsulta en esta sentencia SELECT es calcular la cuota total para los vendedores que trabajan en una oficina particular, específicamente, la oficina que actualmente está siendo examinada por la cláusula WHERE de la consulta principal. La subconsulta lo lleva a cabo explorando la Tabla REPVENTAS. Pero obsérvese que la columna OFICINA en la cláusula WHERE de la subconsulta no se refiere a una columna de la Tabla REPVENTAS; se refiere a una columna de la Tabla OFICINAS, que forma parte de la consulta principal. Conforme SQL recorre cada fila de la Tabla OFICINAS, utiliza el valor OFICINA de la fila actual cuando lleva a cabo la subconsulta.

La columna OFICINA en esta subconsulta es un ejemplo de *referencia externa*. Una referencia externa es un nombre de columna que no se refiere a ninguna de las tablas designadas en la cláusula FROM de la subconsulta en la cual aparece el nombre de la columna. En vez de ello, el nombre de columna se refiere a una columna de una tabla especificada en la Tabla FROM de la consulta *principal*.

8.2.2. Condiciones de búsqueda en subconsultas

Una subconsulta forma parte de una condición de búsqueda en la cláusula WHERE o HAVING. SQL ofrece estas *condiciones de búsqueda en subconsultas*, además de las descritas en otro tema anterior.

- *Test de comparación subconsulta.* Compara el valor de una expresión con un valor único producido por una subconsulta. Este test se asemeja al test de comparación simple.
- *Test de pertenencia a conjunto subconsulta.* Comprueba si el valor de una expresión coincide con uno del conjunto de valores producido por una subconsulta. Este test se asemeja al test de pertenencia a conjunto simple.
- *Test de existencia.* Examina si una subconsulta produce alguna fila de resultados.
- *Test de comparación cuantificada.* Compara el valor de una expresión con cada uno del conjunto de valores producido por una subconsulta.

8.2.2.1. Test de comparación subconsulta (=, <>, <, <=, >, >=)

El test de comparación subconsulta es una forma modificada del test de comparación simple. Compara el valor de una expresión producido por una subconsulta, y devuelve un resultado TRUE si la comparación es cierta. Este test se utiliza para comparar un valor de la fila que está siendo examinada con un valor *único* producido por una subconsulta, como en este ejemplo:

Lista los vendedores cuyas cuotas son iguales o superiores al objetivo de la oficina de ventas de Atlanta.

```
SELECT NOMBRE

FROM REPVENTAS

WHERE CUOTA >= (SELECT OBJETIVO

FROM OFICINAS

WHERE CIUDAD = `Atlanta')
```

La subconsulta del ejemplo recupera el objetivo de ventas de la oficina de Atlanta. El valor se utiliza entonces para seleccionar los vendedores cuyas cuotas son superiores o iguales al objetivo.

El test de comparación subconsulta ofrece los mismos seis operadores de comparación (=, <>, <, <=, >, >=) disponibles con el test de comparación simple. La subconsulta especificada en este test debe producir una *única fila* de resultados. Si la subconsulta produce múltiples filas, la comparación no tiene sentido y SQL informa de una condición de error. Si la subconsulta no produce filas o produce un valor NULL, el test de comparación devuelve NULL.

Lista todos los productos del fabricante ACI para los cuales las existencias superan a las existencias del producto ACI−41004.

```
SELECT DESCRIPCION, EXISTENCIAS

FROM PRODUCTOS

WHERE ID_FAB = `ACI'

AND EXISTENCIAS > (SELECT EXISTENCIAS

FROM PRODUCTOS

WHERE ID_FAB = `ACI'

AND ID_PRODUCTO = `41004')
```

DESCRIPCION EXISTENCIAS

Artículo Tipo 3 207

Artículo Tipo 1 277

Artículo Tipo 2 167

Obsérvese que el test de comparación subconsulta permite una subconsulta únicamente en el lado derecho del operador de comparación. Esta composición:

```
A < (subconsulta)
```

Está permitida, pero esta comparación:

```
(subconsulta) > A
```

no está permitida.

8.2.2.2. Test de pertenencia a conjunto (IN)

El test de pertenencia a conjunto subconsulta (IN) es una forma modificada del test de pertenencia a conjunto simple. Compara un único valor de datos con una columna de valores producida por una subconsulta y devuelve un resultado TRUE si el valor coincide con uno de los valores de la columna.

Lista los vendedores que no trabajan en oficinas dirigidas por Larry Fitch (empleado 108).

SELECT NOMBRE

FROM REPVENTAS

WHERE OFICINA_REP NOT IN (SELECT OFICINA

FROM OFICINAS

WHERE DIR = 108)

NOMBRE

Bill Adams

Mary Jones

Sam Clark

Bob Smith

Dan Roberts

Paul Cruz

e*xpresión−de−test* IN *subconsulta*

NOT

La subconsulta produce una columna de valores, y la cláusula WHERE de la consulta principal, comprueba si un valor de una fila de la consulta principal coincide con uno de los valores de la columna.

8.2.2.3. Test de existencia (EXISTS)

El test de existencia (EXISTS) comprueba si una subconsulta produce alguna fila de resultados. No hay test de comparación simple que se asemeja al test de existencia; solamente se utiliza con subconsultas.

He aquí un ejemplo de una petición que se puede expresar sencillamente utilizando un test de existencia:

Lista los productos para los cuales se ha recibido un pedido de $25.000 o más. La petición podría ser

fácilmente expresada también de esta forma:

Lista los productos para los cuales existe al menos un pedido en la Tabla PEDIDOS a) que se refiere al producto en cuestión y b) que tiene un importe de al menos $25.000.

La sentencia SELECT utilizada para recuperar la lista solicitada de productos se asemeja a la petición así expresada:

```
SELECT DISTINCT DESCRIPCION

FROM PRODUCTOS

WHERE EXISTS (SELECT NUM_PEDIDO

FROM PEDIDOS

WHERE PRODUCTO = ID_PRODUCTO

AND FAB = ID_FAB

AND IMPORTE >= 25,000.00)
```

DESCRIPCION

500−1b Brace

Left Hinge

Right Hinge

Widget Remover

Conceptualmente, SQL procesa esta consulta recorriendo la Tabla PRODUCTOS y efectuando la subconsulta para cada producto. La subconsulta produce para una columna que contiene los números de pedidos de aquellos pedidos del producto actual que superan $25,000.00. Si hay alguno de tales pedidos (es decir, si la columna no está vacía), el test EXISTS es TRUE. Si la subconsulta no produce filas, el test EXISTS es FALSE. El test EXISTS no puede producir un valor NULL.

Se puede invertir la lógica del test EXISTS utilizando la forma NOT EXISTS. En este caso, el test es TRUE si la subconsulta no produce filas, y FALSE en caso contrario.

Se puede observar, que la condición de búsqueda EXISTS no utiliza realmente los resultados de la subconsulta. Simplemente comprueba si la búsqueda produce algún resultado. Por esta razón, SQL suaviza la regla de que las subconsultas deben devolver una única columna de datos, y permite utilizar la forma SELECT * en la subconsulta de un test EXISTS. La subconsulta podría por tanto haber sido escrita:

Lista los productos para los cuales se ha recibido un pedido de $25.000 o más.

```
SELECT DESCRIPCION

FROM PRODUCTOS
```

WHERE EXISTS (SELECT *

FROM PEDIDOS

WHERE PRODUCTO = ID_PRODUCTO

AND FAB = ID_FAB

AND IMPORTE >= 25000.00)

En la práctica, la subconsulta en un test EXISTS se escribe siempre utilizando la notación SELECT *.

Obsérvese que la subconsulta incluye una referencia externa a una columna de la tabla de la consulta principal. En la práctica, la subconsulta de un test EXISTS siempre contienen una referencia externa que enlaza la subconsulta a la fila que actualmente está siendo examinada por la consulta principal.

8.2.2.4. Test cuantificados (ANY y ALL)

La versión subconsulta del test IN comprueba si un valor de dato es igual a algún valor en una columna de los resultados de la subconsulta. SQL proporciona dos test cuantificados, ANY y ALL, que extienden esta noción a otros operadores de comparación, tales como mayor que (>) y menor que (<). Ambos tests comparan un valor de dato con la columna de valores producidos por una subconsulta.

expresión−de−test = ANY subconsulta

<> ALL

<

<=

>

>=

El Test **ANY *.** El test ANY se utiliza conjuntamente con uno de los seis operadores de comparación SQL (=, <>, <, <=, >, >=) para comparar un único valor de test con una columna de valores producidos por una subconsulta. Para efectuar el test, SQL utiliza el operador de comparación especificado para comparar el valor de test con *cada* valor de datos de la columna, uno cada vez.

Si *alguna* de las comparaciones individuales producen un resultado TRUE, el test ANY devuelve un resultado TRUE.

Lista los vendedores que han aceptado un pedido que represente más del 10 por 100 de su cuota.

SELECT NOMBRE

FROM REPVENTAS

WHERE (0.1 * CUOTA) < ANY (SELECT IMPORTE

FROM PEDIDOS

WHERE REP = NUM_EMPL)

NOMBRE

Sam Clark

Larry Fitch

Nancy Angelli

Conceptualmente, la consulta principal examina cada fila de la Tabla REPVENTAS, una a una. La subconsulta encuentra todos los pedidos aceptados por el vendedor actual y devuelve una columna que contiene el importe de esos pedidos. La cláusula WHERE de la consulta principal calcula entonces el diez por ciento de la cuota del vendedor actual y la utiliza como valor del test, comparándolo con todos los importes de pedidos producidos por la subconsulta. Si hay *algún* importe que exceda al valor de test calculado, el test < ANY devuelve TRUE y el vendedor queda incluido en los resultados de la consulta. Si no, el vendedor no se incluye en los result6ados de la consulta.

- Si la subconsulta produce una columna vacía de resultados, el test ANY devuelve FALSE.
- Si el test de comparación es TRUE para al *menos uno* de los valores de datos en la columna, entonces la condición de búsqueda ANY devuelve FALSE.
- Si el test de comparación es FALSE para *todos* de los valores de datos de la columna, entonces la condición de búsqueda ANY devuelve FALSE.
- Si el test de comparación no es TRUE para ningún valor de datos en la columna, pero es NULL para uno o más de los valores, entonces la condición de búsqueda ANY devuelve NULL. En esta situación, no se puede concluir si existe un valor producido por la subconsulta para el cual el test de comparación sea cierto; puede haberlo o no, dependiendo de los valores correctos de los datos NULL (desconocido).

El operador de comparación ANY puede ser engañoso al utilizarlo en la práctica, especialmente junto con el operador de comparación desigualdad (<>). He aquí un ejemplo que expone el problema:

Lista los nombres y edades de todas las personas en el equipo de ventas que no dirigen una oficina.

SELECT NOMBRE, EDAD

FROM REPVENTAS

WHERE NUM_EMPL <> ANY (SELECT DIR

FROM OFICINAS)

La subconsulta:

SELECT DIR

FROM OFICINAS

Produce obviamente los números de empleados que son directores, y por tanto la consulta *parece* decir;

Halla los vendedores que no son directores de ninguna oficina.

Pero esto *no* es lo que la consulta dice. Lo que dice en realidad es:

Halla cada uno de los vendedores que, *para alguna oficina*, no es el director de esa oficina.

Naturalmente para cualquier vendedor, es posible hallar *alguna* oficina en donde ese vendedor no es el director. Los resultados de la consulta incluirían a todos los vendedores, y por tanto fallaría en responder a la cuestión que le fue propuesta. La consulta correcta es:

```
SELECT NOMBRE, EDAD

FROM REPVENTAS

WHERE NOT (NUM_EMPL = ANY (SELECT DIR

FROM OFICINAS)
```

Siempre se puede transformar una consulta con un test ANY en una consulta con un test EXISTS trasladando la comparación al *interior* de la condición de búsqueda de la subconsulta. He aquí una forma alternativa de la consulta, utilizando el test EXISTS.

```
SELECT NOMBRE, EDAD

FROM REPVENTAS

WHERE NOT EXISTS (SELECT *

FROM OFICINAS

WHERE NUM_EMPL = DIR)
```

El test **ALL ***. Al igual que el test ANY, el test ALL se utiliza conjuntamente con uno de los seis operadores de comparación SQL (=, <>, <, <=, >, >=) para comparar un único valor de test con una columna de valores de datos producidos por una subconsulta. Para efectuar el test, SQL utiliza el operador de comparación especificado para comparar el valor de test con todos y cada uno de los valores de datos de la columna. Si todas las comparaciones individuales producen un resultado TRUE, el test ALL devuelve un resultado TRUE.

Lista las oficinas y sus objetivos en donde todos los vendedores tienen ventas que superan el 50 por 100 del objetivo de la oficina.

```
SELECT CIUDAD, OBJETIVO

FROM OFICINAS

WHERE (0.50 * OBJETIVO) < ALL(SELECT VENTAS

FROM REPVENTAS

WHERE OFICINA_REP = OFICINA)
```

CIUDAD OBJETIVO

Denver $300,000.00

New York $575,000.00

Atlanta $350,000.00

Conceptualmente, la consulta principal examina cada fila de la Tabla OFICINAS, una a una. La subconsulta encuentra todos los vendedores que trabajan en la oficina actual y devuelve una columna que contiene las ventas correspondientes a cada vendedor. La cláusula WHERE de la consulta principal calcula entonces el 50% del objetivo de la oficina y lo utiliza como valor de test, comparándolo con todos los valores de ventas producidos por la subconsulta. Si todos los valores de ventas exceden a valor de test calculado, el test < ALL devuelve >TRUE y la oficina se incluye en los resultados de la consulta. Si no, la oficina no se incluye en los resultados de la consulta.

- Si la consulta produce una columna vacía de resultados, el test ALL devuelve TRUE.
- Si el test de comparación es TRUE para todos y cada uno de los valores de datos en la columna, entonces la condición de búsqueda ALL devuelve TRUE.
- Si el test de comparación es FALSE para algún valor de dato en la columna entonces la condición de búsqueda ALL devuelve FALSE.
- Si el test de comparación no es FALSE para ningún valor de datos en la columna, pero es NULL para uno o más de los valores, entonces la condición de búsqueda ALL devuelve NULL.

Los errores sutiles que pueden ocurrir con el test ANY cuando se combina con el operador de comparación desigualdad (<>) también ocurren con el test ALL. Como con el test ANY, el test ALL siempre puede convertirse a un test EXISTS equivalente mediante el traslado de la comparación al interior de la subconsulta.

8.3. SUBCONSULTAS EN LA CLAUSULA HAVING

Aunque las subconsultas suelen encontrarse sobre todo en la cláusula WHERE, también pueden utilizarse en la cláusula HAVING de una consulta. Cuando una subconsulta aparece en la cláusula HAVING, funciona como parte de la selección de grupo de filas efectuada por la cláusula HAVING. Consideremos esta consulta con una subconsulta:

Lista los vendedores cuyo tamaño de pedido medio para productos fabricados por ACI es superior al tamaño de pedido medio global.

```
SELECT NOMBRE, AVG(IMPORTE)

FROM REPVENTAS, PEDIDOS

WHERE NUM_EMPL = REP

AND FAB = `ACI'

GROUP BY NOMBRE

HAVING AVG(IMPORTE) > (SELECT AVG(IMPORTE)

FROM PEDIDOS)
```

NOMBRE AVG(IMPORTE)

Sue Smith $15,000.00

Tom Snyder $22,500.00

La subconsulta calcula el tamaño de pedido medio global. Se trata de una sencilla subconsulta que contiene referencias externas, por lo que SQL puede calcular el promedio una vez y utilizarlo luego repetidamente en la cláusula HAVING. La consulta general recorre la Tabla PEDIDOS, hallando todos los pedidos de productos ACI, y los agrupa por vendedor. La cláusula HAVING comprueba entonces cada grupo de filas para ver si el tamaño de pedido medio de ese grupo es superior al promedio de todos los pedidos, calculado con antelación. Si es así, el grupo de filas es retenido; si no, el grupo de filas es descartado. Finalmente, la cláusula SELECT produce una fila resumen por cada grupo, mostrando el nombre del vendedor y el tamaño de pedido medio para cada uno.

8.4. RESUMEN

- Una subconsulta es una consulta dentro de una consulta. Las subconsultas aparecen dentro de una de las condiciones de búsqueda subconsulta en la cláusula WHERE o HAVING.
- Cuando aparece una subconsulta en la cláusula WHERE, los resultados de la subconsulta se utilizan para seleccionar las filas individuales que contribuyen a los datos de los resultados de la consulta principal.
- Cuando una subconsulñta aparece en la cláusula HAVING, los resultados de la subconsulta se utilizan para seleccionar los grupos de filas que contribuyen con datos a los resultados de la consulta.
- La forma subconsulta del test de comparación utiliza uno de los operadores de comparación simple para comparar un valor de test con el valor único devuelto por una subconsulta.
- La forma subconsulta del test de pertenencia a conjunto (IN) compara el valor de test con el conjunto de valores devuelto por una subconsulta.
- El test de existencia (EXISTS) comprueba si una subconsulta devuelve algún valor.
- Los tests cuantificados (ANY y ALL) utilizan uno de los operadores de comparación simple para comparar un valor de test con todos los valores devueltos por una subconsulta, comprobando si la comparación resulta cierta para alguno o todos los valores.
- Una subconsulta puede incluir una *referencia externa* a una tabla de cualquiera de las consultas que la contienen, enlazando la subconsulta con la fila actual de esa consulta.

8.5. RESUMEN FINAL. CONSULTAS SQL

Las cláusulas de las sentencias SELECT proporcionan un conjunto potente y flexible de características para recuperar datos de la base de datos. Cada cláusula juega un papel específico en la recuperación de datos:

- La cláusula FROM especifica las tablas fuente que contribuyen con datos a los resultados de la consulta. Todos los nombres de columna en el cuerpo de l sentencia SELECT deben identificar sin ambigüedad a una columna de una tabla fuente en una consulta externa.
- La cláusula WHERE, si está presente, selecciona combinaciones individuales de filas procedentes de las tablas fuente que participan en los resultados de la consulta. Las subconsultas en la cláusula WHERE se evalúan para cada fila individual.
- La cláusula GROUP BY, si está presente, agrupa las filas individuales seleccionadas por la cláusula WHERE en grupos de filas.
- La cláusula HAVING, si está presente, selecciona grupos de filas que participan en los resultados de la consulta. Las subconsultas de la cláusula HAVING se evalúan para cada grupo de filas.

- La cláusula SELECT determina qué valores de datos aparecen realmente como columnas en los resultados finales.
- La palabra clave DISTINCT, si está presente, elimina filas duplicadas de los resultados de la consulta.
- El operador UNION, si está presente, mezcla los resultados producidos por las sentencias SELECT individuales en un único conjunto de resultados de la consulta.
- La cláusula ORDER BY, si está presente, ordena los resultados finales basándose en una o más columnas.

8.6. EJERCICIOS

- Lista todos los clientes atendidos por Bill Adams.
- Lista los vendedores que trabajan en oficinas que superen su objetivo.
- Lista los oficinas en donde haya un vendedor cuya cuota represente más del 55 por 100 del objetivo de la oficina.
- Lista los nombres y edades de los vendedores que tienen cuotas por encima del promedio (de cuotas).
- Lista los clientes cuyos vendedores están asignados a oficinas de la región de ventas Este.
- Lista los vendedores que tienen más de 40 años y que dirigen a un vendedor cuyas ventas están por encima de la cuota.

TEMA IX

VISTAS

9.1. INTRODUCCION

9.2. CREACION DE UNA VISTA (CREATE VIEW)

9.2.1. Vistas horizontales

9.2.2. Vistas verticales

9.2.3. Vistas con subconjuntos fila/columna

9.2.4. Vistas agrupadas

9.2.5. Vistas compuestas

9.3. ACTUALIZACION DE UNA VISTA

9.4. ACTUALIZACIONES DE VISTAS EN PRODUCTOS SQL COMERCIALES

9.5. COMPROBACION DE ACTUALIZACIONES DE VISTAS (CHECK OPTION)

9.6. ELIMINACION DE UNA VISTA (DROP VIEW)

9.7. RESUMEN

Notas

9.1. INTRODUCCION

Las tablas de una base de datos definen la estructura y organización de sus datos. Sin embargo, SQL

también permite mirar los datos almacenados de otros modos mediante la definición de vistas alternativas de datos. Una *vista* es una consulta SQL que está permanentemente almacenada en la base de datos y a la que se le asigna un nombre. Los resultados de una consulta almacenada son visibles a través de la vista y SQL permite acceder a estos resultados como si fueran, de hecho, una tabla real en la base de datos.

Las vistas son parte importante de SQL, por varias razones:

- Las vistas permiten acomodar el aspecto de una base de datos de modo que diferentes usuarios la vean desde diferentes perspectivas.
- Las vistas permiten restringir acceso a los datos, permitiendo que diferentes usuarios sólo vean ciertas filas o ciertas columnas de una tabla.
- Las vistas simplifican el acceso a la base de datos mediante la presentación de la estructura de los datos almacenados del modo que sea más natural a cada usuario.

¿Qué es una vista?

Una vista es una tabla virtual en la base de datos cuyos contenidos están definidos por una consulta. Para el usuario de la base de datos, la vista aparece igual que una tabla real, con un conjunto de columnas designadas y filas de datos. Pero a diferencia de una tabla real, una vista no existe en la base de datos como conjunto almacenado de valores. En su lugar, las filas y columnas de datos visibles a través de la vista son los resultados producidos por la consulta que define la vista. SQL crea la ilusión de la vista dándole a ésta un nombre semejante a un nombre de tabla y almacenando la definición de la vista en la base de datos.

Cómo maneja el DBMS las vistas

Cuando el DBMS encuentra una referencia a una vista en una sentencia SQL, determina la definición de la vista almacenada en la base de datos.

Para vistas sencillas, el DBMS puede construir cada fila de las vistas sobre la marcha, extrayendo los datos para la fila de las tablas fuente. Para vistas más complejas, el DBMS debe materializar realmente la vista; es decir, el DBMS debe llevar a cabo efectivamente la consulta que define la vista y almacenar sus resultados en una temporal. El DBMS confirma las peticiones de acceso a la vista a partir de esta tabla temporal y descarta la tabla cuando ya no es necesaria.

V**entajas de las vistas**

- *Seguridad*. Cada usuario puede obtener permiso para acceder a la base de datos únicamente a través de un pequeño conjunto de vistas que contienen los datos específicos que el usuario está autorizado a ver, restringiendo así el acceso del usuario a los datos almacenados.
- *Simplicidad de consulta.* Una vista puede extraer datos de varias tablas diferentes y presentarlos como una única tabla, haciendo que consultas multitabla se formulen como consultas de una sola tabla con respecto a la vista.
- *Simplicidad estructurada.* Las vistas pueden dar a un usuario una visión personalizada de la estructura de la base de datos presentando ésta como un conjunto de tablas virtuales que tienen sentido para ese usuario.
- *Aislamiento frente al cambio.* Una vista puede presentar una imagen consistente inalterada de la estructura de la base de datos incluso si las tablas fuentes subyacentes se dividen, reestructuran o cambian de nombre.
- *Integridad de datos.* Si se accede a los datos y se introducen a través de una vista, el DBMS puede comprobar automáticamente los datos para asegurarse que satisfacen restricciones de integridad especificadas.

Desventajas de las vistas

- *Rendimiento.* Las vistas crean la *apariencia* de una tabla, pero el DBMS debe traducir las consultas con respecto a la vista en consultas con respecto a las tablas fuente subyacentes.
- *Restricciones de actualización.* Cuando un usuario trata de actualizar filas de una vista, el DBMS debe traducir la petición a una actualización sobre las filas de las tablas fuente subyacentes.

9.2. CREACION DE UNA VISTA (CREATE VIEW)

La sentencia CREATE VIEW se utiliza para crear una vista. La sentencia asigna un nombre a la vista y especifica la consulta que define la vista. Para crear la vista con éxito es preciso tener permiso para acceder a todas las tablas referenciadas en la consulta. Solamente se especifican los nombres de las columnas; el tipo de datos, la longitud y las otras características de cada columna se deducen de la definición de las columnas en las tablas fuente. Si la lista de nombres de columnas se omite de la sentencia CREATE VIEW, cada columna de la vista adopta el nombre de la columna correspondiente de la consulta. La lista de nombres de columnas debe ser especificada si la consulta incluye columnas calculadas o si produce dos columnas con nombres idénticos.

Aunque todas las vistas se crean del mismo modo en la práctica se utilizan diferentes tipos de vistas para propósitos diferentes.

9.2.1. Vistas horizontales

Un uso común de las vistas es restringir el acceso de un usuario a las filas seleccionadas de una tabla. Por ejemplo, puede ser deseable que un director de ventas solamente vea filas de REPVENTAS correspondientes a los vendedores de la región propia del director. Para lograr esto, se pueden definir dos vistas, del modo siguiente:

Crea una vista que muestre a los vendedores de la región Este.

CREATE VIEW REPESTE AS

SELECT *

FROM REPVENTAS

WHERE OFICINA_REP IN (11, 12, 13)

Ahora se puede dar a cada director de ventas permiso para acceder a la vista REPESTE negándoles el permiso para acceder a la propia Tabla REPVENTAS.

Una vista horizontal divide horizontalmente la tabla fuente para crear la vista. Todas las columnas de la tabla fuente participan en la vista, pero sólo algunas de sus filas son visibles a través de la vista. Las vistas horizontales son adecuadas cuando la tabla fuente contiene datos que relacionan a varias organizaciones o usuarios. Proporcionan una tabla privada para cada usuario, compuesta únicamente de las filas necesarias para ese usuario.

Define una vista que contiene únicamente las oficinas de la región Este.

CREATE VIEW OFICINASESTE AS

SELECT *

```
FROM OFICINAS

WHERE REGION = `Este'
```

Define una vista para Sue Smith (empleada número 102) que contiene solamente los pedidos remitidos por clientes asignados a ella.

```
CREATE VIEW PEDIDOSSUE AS

SELECT *

FROM PEDIDOS

WHERE CLIE IN (SELECT NUM_CLIE

FROM CLIENTES

WHERE NUM_CLIE = 2102)
```

En cada uno de estos ejemplos, la vista se deriva de una única tabla fuente. La vista se define mediante una consulta SELECT * y por tanto tiene exactamente las mismas columnas que la tabla fuente. La cláusula WHERE determina qué filas de la tabla fuente son visibles en la vista.

9.2.2. Vistas verticales

Otro uso habitual de las vistas es restringir el acceso de un usuario a sólo ciertas columnas de una tabla. Por ejemplo, en nuestra base de datos, el departamento de procesamiento de pedidos puede necesitar acceder al número de empleado, su nombre y la asignación de oficina. Sin embargo, no hay necesidad de que el personal de procesamiento de pedidos vea las ventas actuales del vendedor o su cuota. Esta vista selectiva de la Tabla REPVENTAS puede ser construida con la vista siguiente:

Crea una vista mostrando información seleccionada de cada vendedor.

```
CREATE VIEW INFOREP AS

SELECT NUM_EMPL, NOMBRE, OFICINA_REP

FROM REPVENTAS
```

Dando al personal de procesamiento de pedidos acceso a esta vista y denegándole acceso a la propia Tabla REPVENTAS, el acceso a datos de cuotas y ventas específicoa se restringe efectivamente.

Una vista como INFOREP se denomina con frecuencia *vista vertical*. Una vista vertical divide la tabla fuente verticalmente para crear la vista. Las vistas verticales suelen encontrarse allí donde los datos almacenados en una tabla son utilizados por varios usuarios o grupos de usuarios. Proporcionan una tabla privada a cada usuario, compuesta únicamente de las columnas necesarias a ese usuario.

Define una vista de la Tabla OFICINAS para el personal de procesamiento de pedidos que incluye la ciudad, el número de oficina y región.

```
CREATE VIEW INFOOFICINA AS
```

```
SELECT OFICINA, CIUDAD, REGION

FROM OFICINAS
```

En cada uno de estos ejemplos, la vista se deriva de una tabla fuente única. La lista de selección en la definición de la vista determina qué columnas de la tabla fuente son visibles en la lista. Puesto que se trata de vistas verticales, cada fila de la tabla fuente está representada en la vista y la definición no incluye una cláusula WHERE.

9.2.3. Vistas con subconjuntos fila/columna

Es bastante habitual definir una vista que divida una tabla fuente tanto por la dimensión horizontal como por la vertical, como en este ejemplo:

Define una vista que contiene el número de cliente, el nombre de empresa y el límite de crédito de todos los clientes asignados a Bill Adams (empleado número 105)

```
CREATE VIEW CLIEBILL AS

SELECT NUM_CLIE, EMPRESA, LIMITE_CREDITO

FROM CLIENTES

WHERE REP_CLIE = 105
```

Los datos visibles a través de esta vista son un subconjunto fila/columna de la Tabla CLIENTES. Sólo las columnas explícitamente designadas en la lista de selección de la vista y las filas que satisfacen la condición de búsqueda son visibles a través de la vista.

9.2.4. Vistas agrupadas

La consulta especificada en una definición de vista puede incluir una cláusula GROUP BY. Este tipo de vista se denomina *vista agrupada*, ya que los datos visibles a través de ella son el resultado de una consulta agrupada. Las vistas agrupadas efectúan la misma función que las consultas agrupadas; agrupan filas relacionadas de datos y producen una fila de resultados de consulta para cada grupo, resumiendo los datos de ese grupo.

Define una vista que contiene datos de pedidos sumarios para cada vendedor.

```
CREATE VIEW PED_POR_REP (QUIEN, CUANTOS, TOTAL, INF, SUP, MEDIO) AS

SELECT REP, COUNT(*), SUM(IMPORTE), MIN(IMPORTE), MAX(IMPORTE),
AVG(IMPORTE)

FROM PEDIDOS

GROUP BY REP
```

La definición de una vista agrupada siempre incluirá una lista de nombres de columna. La lista asigna nombres a las columnas de la vista agrupada que se derivan en funciones de columna, tales como SUM() y MIN(). También puede especificar un nombre modificado para una columna de agrupación. En este ejemplo, la columna REP de la Tabla PEDIDOS pasa a ser la columna QUIEN en la vista

PED_POR_REP.

Una vez que se define esta vista agrupada, puede ser utilizada para simplificar consultas. Por ejemplo, esta consulta genera un sencillo informe que resume los pedidos de cada vendedor:

Muestra el nombre, los números de pedido, el importe total de pedidos y el pedido medio para cada vendedor.

```
SELECT NOMBRE, CUANTOS, TOTAL, MEDIO

FROM REPVENTAS, PED_POR_REP

WHERE QUIEN = NUM_EMPL

ORDER BY TOTAL DESC
```

A diferencia de una vista horizontal o vertical, las filas en una vista agrupada no tienen una correspondencia una a una con las filas de la tabla fuente. Una vista agrupada no es únicamente un filtro de su tabla fuente que oculta ciertas filas y columnas. Es un resumen de las tablas fuente y, por tanto, se requiere una sustancial cantidad de procesamiento del DBMS para mantener la ilusión de una tabla virtual para vistas agrupadas.

Las vistas agrupadas pueden ser utilizadas en consultas igual que cualquier otra vista más sencilla. Sin embargo, una vista agrupada no puede ser actualizada. La razón debería ser obvia a partir del ejemplo.

Puesto que cada fila de la vista agrupada corresponde a un *grupo* de filas de la tabla fuente y, puesto que las columnas de la vista agrupada contienen generalmente datos calculados, no hay manera de trasladar una petición de actualización a una actualización de las filas de la tabla fuente. Las vistas agrupadas funcionan por tanto como vistas de sólo lectura, que pueden participar en consultas, pero en actualizaciones.

Las vistas agrupadas están también sujetas a las restricciones SQL sobre funciones de columna animadas. Las funciones de columna animadas, tales como:

```
MIN(MIN(A))
```

No son legales en expresiones SQL. Aunque la vista agrupada oculta las funciones de columna de su lista de selección al usuario, el DBMS sigue conociéndolas y fuerza la restricción. Por ejemplo:

Para cada oficina de ventas, muestra el rango de los tamaños medios de pedido para todos los vendedores que trabajan en la oficina.

```
SELECT OFICINA_REP, MIN(MEDIO), MAX(MEDIO)

FROM REPVENTAS, PED_POR_REP

WHERE NUM_EMPL = QUIEN

GROUP BY OFICINA_REP
```

La consulta que se está pidiendo pero que no funciona es:

```
SELECT OFICINA_REP, MIN(AVG(IMPORTE)), MAX(AVG(IMPORTE))

FROM REPVENTAS, PEDIDOS

WHERE NUM_EMPL = REP

GROUP BY REP

GROUP BY OFICINA_REP
```

Esta consulta es ilegal a causa del doble GROUP BY y de las funciones de columna anidadas. Desgraciadamente, como muestra este ejemplo, una sentencia SELECT agrupada perfectamente razonable puede, de hecho, producir un error si una de sus tablas fuente resulta ser una vista agrupada. No hay manera de anticipar esta situación; debe comprenderse la causa del error cuando SQL informa de él.

9.2.5. Vistas compuestas

Una de las razones más frecuentes para utilizar vistas compuestas es simplificar las consultas multitabla. Especificando una consulta de dos o tres tablas en la definición de vista, se puede crear una *vista compuesta* que extrae sus datos de dos o tres tablas diferentes y presenta los resultados de la consulta como una única tabla virtual. Una vez definida la vista, con frecuencia se puede utilizar una consulta simple de una sola tabla con respecto a la vista para peticiones que en caso contrario requerirían una composición de dos o tres tablas.

Por ejemplo, supongamos que Sam Clark, el vicepresidente de ventas, efectúa con frecuencia consulta sobre la Tabla PEDIDOS de la base de datos ejemplo. Sin embargo, Sam no quisiera trabajar con números de clientes y empleado. En vez de ello, le gustaría poder utilizar una versión de la Tabla PEDIDOS que tenga nombres en vez de números. He aquí una vista que satisface las necesidades de Sam:

Crea una vista de la Tabla PEDIDOS con nombres en vez de números.

```
CREATE VIEW INFO_PEDIDO (NUM_PEDIDO, EMPRESA, NOMBRE_REP, IMPORTE) AS

SELECT NUM_PEDIDO, EMPRESA, NOMBRE, IMPORTE

FROM PEDIDOS, CLIENTES, REPVENTAS

WHERE CLIE = NUM_CLIE

AND REP = NUM_EMPL
```

Esta vista está definida mediante una composición de tres tablas. Como con la vista agrupada, el procesamiento necesario para crear la ilusión de una tabla virtual para esta vista es considerable. Cada fila de la vista se deriva de una *combinación* de una fila de la Tabla PEDIDOS, una de la fila de la Tabla CLIENTES y una fila de la Tabla REPVENTAS.

Aunque tiene una definición relativamente compleja, esta vista puede proporcionar algunos beneficios reales.

Muestra los pedidos actuales totales para cada empresa y para cada vendedor.

```
SELECT NOMBRE_REP, EMPRESA, SUM(IMPORTE)

FROM INFO_PEDIDO

GROUP BY NOMBRE_REP, EMPRESA
```

Esta consulta es una sentencia SELECT de una sola tabla, que es considerablemente más simple que la sentencia SELECT de tres tablas para las tablas fuente:

```
SELECT NOMBRE, EMPRESA, SUM(IMPORTE)

FROM REPVENTAS, PEDIDOS, CLIENTES

WHERE REP = NUM_EMPL

AND CLIE = NUM_CLIE

GROUP BY NOMBRE, EMPRESA
```

Análogamente, es fácil generar un informe de los mayores pedidos, mostrando quién los remitió y quién los recibió, con esta consulta sobre la vista:

Mostrar los pedidos cuyo importe sea superior a 2000000:

```
SELECT EMPRESA, IMPORTE, NOMBRE_REP

FROM INFO_PEDIDO

WHERE IMPORTE > 20000.00

ORDER BY IMPORTE DESC
```

La vista hace mucho más fácil examinar lo que sucede en la consulta que si fuera expresada como la composición equivalente de tres tablas. Naturalmente, el DBMS debe trabajar igual de duro para generar los resultados de la consulta en la consulta de una tabla con respecto a la vista que si generara los resultados para la consulta equivalente de tres tablas. De hecho, el DBMS debe efectuar un poco más de trabajo para manejar la consulta con respecto a la vista. Sin embargo, para el usuario humano de la base de datos es mucho más fácil escribir y comprender la consulta de una única tabla que referencia a la vista.

9.3. ACTUALIZACION DE UNA VISTA

Crea una vista que muestre los vendedores de la región Este.

```
CREATE VIEW REPESTE AS

SELECT *

FROM REPVENTAS

WHERE OFICINA_REP IN (11, 12, 13)
```

Se trata de una vista horizontal directa, deducida de una tabla fuente única. Tiene sentido hablar de insertar una fila en esta vista; significa que la nueva fila debería ser insertada en la Tabla REPVENTAS subyacente a partir de la cual se derivó la vista. También tiene sentido suprimir una fila de la vista REPESTE; esto eliminaría la fila correspondiente de la Tabla REPVENTAS. Finalmente, actualizar una fila de la vista REPESTE también tiene sentido; actualizaría la fila correspondiente de la Tabla REPVENTAS. En cada caso la acción puede ser realizada con respecto a la fila correspondiente de la tabla fuente, preservando la integridad tanto de la tabla fuente como de la vista.

Define una vista que contiene datos de pedidos sumarios para cada vendedor.

CREATE VIEW PED_POR_REP (QUIEN, CUANTOS, TOTAL, INF, SUP, MEDIO) AS

SELECT REP, COUNT(*), SUM(IMPORTE), MIN(IMPORTE), MAX(IMPORTE), AVG(IMPORTE)

FROM PEDIDOS

GROUP BY REP

No existe una correspondencia una a una entre las filas de estas vistas y las filas de la Tabla PEDIDOS subyacente, por lo cual no tiene sentido hablar de insertar, suprimir o actualizar filas de esta vista. La vista PED_POR_REP no es actualizable; es una vista de sólo lectura.

La vista REPESTE y la vista PED_POR_REP son dos ejemplos extremos en términos de la complejidad de sus definiciones. Hay vistas más complejas que REPESTE donde sigue teniendo sentido actualizar la vista, y hay vistas menos complejas que PED_POR_REP en donde las actualizaciones no tienen sentido. De hecho, determinar qué vistas y cuáles pueden ser actualizadas no ha sido un importante problema de investigación en base de datos relacional a lo largo de los años.

Una vista puede ser actualizada si la consulta que la define satisface todas estas restricciones:

- No debe especificarse DISTINCT, es decir, las filas duplicadas *no* deben ser eliminadas de los resultados de la consulta.
- La cláusula FROM debe especificar solamente una tabla actualizable; es decir, la vista debe tener una única tabla fuente para la cual el usuario tiene los privilegios requeridos.
- Cada elemento de selección debe ser una referencia de columna simple; la lista de selección no puede contener expresiones, columnas calculadas o funciones de columna.
- La cláusula WHERE no debe incluir una subconsulta; sólo pueden aparecer condiciones de búsqueda simples fila a fila.
- La consulta no debe incluir una cláusula GROUP BY o HAVING.

Para que una vista sea actualizable, el DBMS debe ser capaz de relacionar cualquier fila de la vista con su fila fuente en la tabla fuente. Análogamente, el DBMS debe ser capaz de relacionar cada columna individual a actualizar con su columna fuente en la tabla fuente.

Si la vista satisface esta comprobación, es posible definir operaciones INSERT, DELETE y UPDATE significativas para la vista en términos de la(s) tabla(s) fuente.

9.4. ACTUALIZACIONES DE VISTAS EN PRODUCTOS SQL COMERCIALES

Hay muchas vistas que pueden ser actualizadas teóricamente y que no satisfacen todas las restricciones.

Crea una vista mostrando las ventas, cuota y la diferencia entre ambas para cada vendedor.

```
CREATE VIEW RENDVENTAS (NUM_EMPL, VENTAS, CUOTA, DIFE) AS

SELECT NUM_EMPL, VENTAS, CUOTA, (VENTAS − CUOTA)

FROM REPVENTAS
```

El estándar SQL1 no permite actualizaciones de esta vista, ya que su cuarta columna es una columna calculada. Sin embargo, cada fila de esta vista puede ser relacionada con una fila única en la tabla fuente (REPVENTAS). Por esta razón DB (y otras varias implementaciones SQL comerciales) permitirán operaciones DELETE dentro de esta vista.

DB2 permite operaciones UPDATE sobre las columnas NUM_EMPL, VENTAS, y CUOTA, ya que se derivan directamente de la tabla fuente. Únicamente la columna DIFE no puede ser actualizada. DB2 no permite la sentencia INSERT para la vista, ya que insertar un valor para la columna DIFE no tendría significado.

Las reglas específicas que determinan si una vista puede ser actualizada o no varían de un producto DBMS a otro, y suelen estar bastante detalladas.

El mejor modo de determinar la posibilidad de actualización de las vistas en un DBMS particular es consultar la guía de usuario o experimentar con diferentes tipos de vistas.

9.5. COMPROBACION DE ACTUALIZACIONES DE VISTAS (CHECK OPTION)

Si una vista se define mediante una consulta que incluye una cláusula WHERE, sólo las filas que satisfacen la condición de búsqueda son visibles en la vista. Otras filas pueden estar presentes en la tabla fuente de la cual se deriva a la vista, pero no son visibles a través de la vista. Por ejemplo, la vista REPESTE descrita anteriormente en este capítulo sólo contiene las filas de la Tabla REPVENTAS con valores específicos en la columna OFICINA_REP.

Crea una vista que muestre los vendedores de la región Este.

```
CREATE VIEW REPESTE AS

SELECT *

FROM REPVENTAS

WHERE OFICINA_REP IN (11, 12, 13)
```

Esta es una vista actualizable y para la mayoría de las implementaciones SQL comerciales. Se puede añadir un nuevo vendedor con esta sentencia INSERT.

```
INSERT INTO REPESTE (NUM_EMPL, NOMBRE, OFICINA_REP, EDAD, VENTAS)

VALUES (113, `Jake Kimball', 11, 43, 0.00)
```

El DBMS añadirá la nueva fila a la Tabla REPVENTAS subyacente y la fila será visible a través de la vista REPESTE. Pero consideremos qué ocurre cuando se añade un nuevo vendedor con esta sentencia INSERT.

INSERT INTO REPESTE (NUM_EMPL, NOMBRE, OFICINA_REP, EDAD, VENTAS)

VALUES (114, `Fred Roberts', 21, 47, 0.00)

Esta es una sentencia SQL perfectamente legal y el DBMS la insertará en la Tabla REPVENTAS. Sin embargo, la fila recién insertada no satisface la condición de búsqueda para la vista. Su valor de OFICINA_REP (21) especifica la oficina de Los Angeles, que está en la región Oeste. Como resultado, si se efectúa esta consulta inmediatamente después de la sentencia INSERT.

SELECT NUM_EMPL, NOMBRE, OFICINA_REP

FROM REPESTE

La fila recién añadida no aparecerá en la vista. Lo mismo ocurre si se cambia la asignación de oficina para uno de los vendedores actualmente en la vista. Esta sentencia UPDATE:

UPDATE REPESTE

SET OFICINA_REP = 21

WHERE NUM_EMPL = 104

Modifica una de las columnas para la fila de Bob Smith e inmediatamente hace que desaparezca de la vista. Naturalmente ambas filas desaparecidas aparecen en una consulta con respecto a la tabla subyacente.

SELECT NUM_EMPL, NOMBRE, OFICINA_REP

FROM REPVENTAS

NUM_EMPL NOMBRE OFICINA_REP

105 Bill Adams 13

109 Mary Jones 11

102 Sue Smith 21

106 Sam Clark 11

104 Bob Smith 21

101 Dan Roberts 12

110 Tom Snyder NULL

108 Larry Fitch 21

103 Paul Cruz 12

107 Nancy Angelli 22

114 Fred Roberts 21

El hecho de que las filas desaparezcan de la vista como resultado de una sentencia INSERT o UPDATE es desconcertante, como poco. Probablemente es desear que el DBMS detecte e impida este tipo de INSERT o UPDATE sobre la vista. SQL permite especificar este tipo de comprobación de integridad para vistas mediante la creación de la vista con una *opción de comprobación*. La opción de comprobación se especifica en la sentencia CREATE VIEW, como se muestra en esta definición de la vista REPESTE:

CREATE VIEW REPESTE AS

SELECT *

FROM REPVENTAS

WHERE OFICINA_REP IN (11, 12, 13)

WITH CHECK OPTION

Cuando se solicita la opción de comprobación para una vista, SQL comprueba *automáticamente* cada operación INSERT y cada operación UPDATE sobre la vista para asegurarse que las filas resultantes satisfagan el criterio de búsqueda de la definición de la vista. Si una fila insertada o modificada no satisficiera la condición, la sentencia INSERT o UPDATE fallaría y la operación no se llevaría a cabo.

9.6. ELIMINACION DE UNA VISTA

Debido a que las vistas se comportan como tablas, y una vista no puede tener el mismo nombre que una tabla, muchos productos DBMS utilizan la sentencia DROP TABLE para eliminar vistas. Otras implementaciones SQL proporcionan la sentencia DROP VIEW:

9.7. RESUMEN

- Una vista es una tabla virtual definida mediante una consulta. La vista `parece contener filas y columnas de datos, al igual que una tabla real, pero los datos visibles a través de la vista son, de hecho, los resultados de la consulta.
- Una vista puede ser un subconjunto simple fila/columna de una única tabla, puede sumarizar una tabla (una vista agrupada) o puede extraer sus datos de dos o más tablas (una vista compuesta).
- Una vista puede ser referenciada como una tabla real en una sentencia SELECT, INSERT, DELETE o UPDATE. Sin embargo, las vistas más complejas no pueden ser actualizadas; son vistas de sólo lectura.
- Las vistas suelen utilizarse para simplificar la estructura aparente de una base de datos, para simplificar consultas y para proteger ciertas filas y/o columnas frente a acceso no autorizado.

Pág.−216 SQL

SQL Pág.−217

Dirección

Intérprete de

Comandos

SGBD

Sistema

operativo

Intérprete de SQL

SGBD

Ejecución normal

BASE DE DATOS

.....

.....

Vista 1

Vista N

E

DIRECTOR = 106

DIRECTOR = 104

DIRECTOR = NULL

UNION

UNION

PRODUCTO

COMPOSICION

INTERNA

Filas sin emparejar

SELECT SUM(CUOTA) WHERE OFICINA_REP = 22

>?

SELECT SUM(CUOTA) WHERE OFICINA_REP = 21

>?

E

E

R1

R2

R3

E1

E

E2

E

a1

E

E

E1

E2

R

CLIENTE

EMPLEADO

DIRIGE

CLIENTE

NUEVO

USADO

EMPLEADO

COCHE

Pone a punto

compra

compra

Pone a punto

COCHE

EMPLEADO

Pone a punto

Compra−usado

CLIENTE

Compra nuevo

EMPLEADO

CLIENTE

COCHE

USADO

COCHE

NUEVO

COCHE

Compra

USADO

Es

nuevo

Pone a punto

EMPLEADO

CLIENTE

Es

usado

NUEVO

ESTUDIANTE

REALIZA

PROYECTO

LIBRO

SUMINISTRA

EDITORIAL

ASIGNATURA

CURSA

ALUMNO

ASIGNATURA

CURSA

ALUMNO

COCHE

♦