TEMA 3: El lenguaje SQL I: Consulta de datos.

3.1 Introducción.

Antes de proceder a explicar el lenguaje en SQL, veamos la descripción de la pequeña base de datos relacional que usaremos a lo largo del presente tema para la ejecución de los ejemplos.

La base de datos esta formada por cinco tablas, donde cada tabla almacena información referente a un tipo particular de entidad. El diagrama de la base de datos puede verse en la figura siguiente (figura 3.1.1).

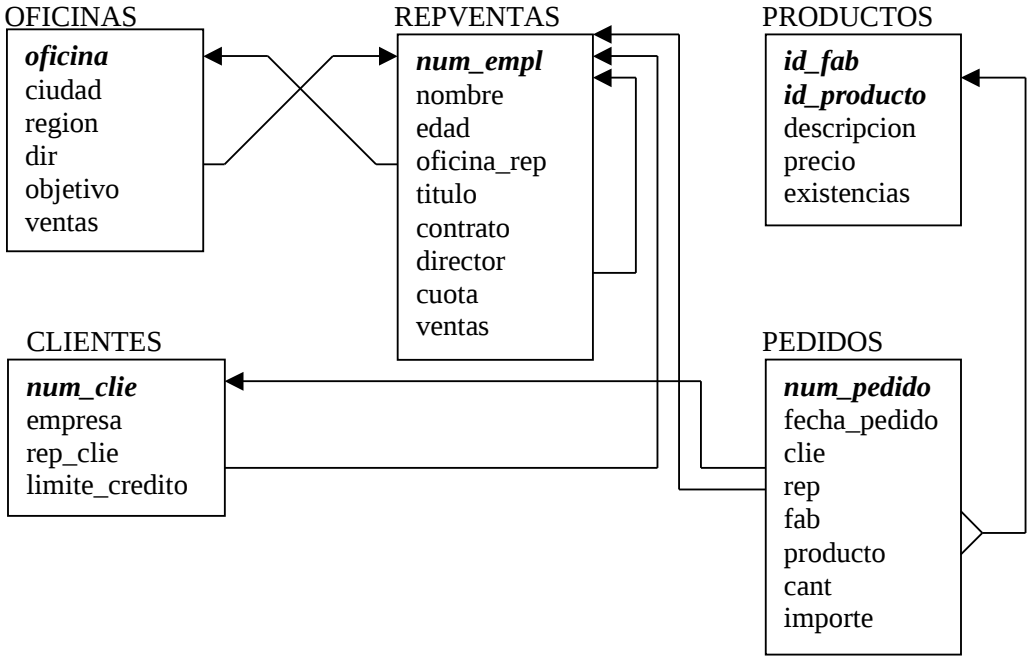


Figura 3.1.1: Diagrama de la base de datos de ejemplo.

La tabla OFICINAS (tabla 3.1.1) almacena datos acerca de cada una de las cinco oficinas de ventas incluyendo la ciudad en donde está localizada la oficina, la región de ventas a la que pertenece, etc.

OFICINA	*CIUDAD*	*REGION*	*DIR*	*OBJETIVO*	*VENTAS*
22	Toledo	Centro	108	27.500	34.432
11	Valencia	Este	106	52.500	40.063
12	Barcelona	Este	104	70.000	29.328
13	Alicante	Este	105	30.000	39.327
21	Madrid	Centro	108	60.000	81.309

Tabla 3.1.1: La tabla OFICINAS de la base de datos de ejemplo.

La tabla CLIENTES (tabla 3.1.2) almacena datos acerca de cada cliente, tales como el nombre de la empresa, el límite de crédito y el vendedor que atiende al cliente.

NUM_CLIE	*EMPRESA*	*REP_CLIE*	*LIMITE_CREDITO*
2111	EVBE S.A.	103	50.000
2102	Exclusivas del Este S.L.	101	65.000

2103	Pino S.L.	105	50.000
2123	Hnos. Martinez S.A.	102	40.000
2107	Distribuciones Sur S.A.	110	35.000
2115	AFS S.A.	101	20.000
2101	Exclusivas Soriano S.A.	106	65.000
2112	Lopez Asociados S.L.	108	50.000
2121	Hernandez & hijos S.L.	103	45.000
2114	Componentes Fernandez S.A.	102	20.000
2124	Domingo S.L.	107	40.000
2108	Zapater Importaciones S.A.	109	55.000
2117	Hnos. Ramon S.L.	106	35.000
2122	JPF S.L.	105	30.000
2120	Distribuciones Montiel S.L.	102	50.000
2106	Construcciones Leon S.A.	102	65.000
2119	Martinez & Garcia S.L.	109	25.000
2118	Exclusivas Norte S.A.	108	60.000
2113	Importaciones Martin S.L.	104	20.000
2109	Roda & Castedo S.L.	103	25.000
2105	MALB S.A.	101	45.000

Tabla 3.1.2: La tabla CLIENTES de la base de datos de ejemplo.

La tabla REPVENTAS (tabla 3.1.3) almacena el número de empleado, el nombre, la edad, las ventas anuales hasta la fecha y otros datos referentes a cada vendedor.

NUM_EMPL	*NOMBRE*	*EDAD*	*OFICINA_REP*	*TITULO*	*CONTRATO*
106	Jose Maldonado	52	11	VP Ventas	14/06/1998
104	Carlos Martinez	33	12	Dir. Ventas	19/05/1997
105	Belen Aguirre	37	13	Dir. Ventas	12/02/1998
109	Maria Garcia	31	11	Rep. Ventas	12/10/1999
108	Lorenzo Fernandez	62	21	Dir. Ventas	12/10/1999
102	Soledad Martinez	48	21	Rep. Ventas	10/12/1996
101	Daniel Gutierrez	45	12	Rep. Ventas	20/10/1996
110	Antonio Valle	41	NULL	Rep. Ventas	13/01/2000
103	Pedro Cruz	29	12	Rep. Ventas	01/03/1997
107	Natalia Martin	49	22	Rep. Ventas	14/11/1998

DIRECTOR	*CUOTA*	*VENTAS*
NULL	25.000	32.958
106	17.500	0
104	30.000	39.327
106	27.500	7.105
106	30.000	58.533
108	30.000	22.776
104	27.500	26.628
101	NULL	23.123
104	25.000	2.700
108	27.500	34.432

Tabla 3.1.3: La Tabla REPVENTAS de la base de datos de ejemplo.

La tabla PRODUCTOS (tabla 3.1.4) almacena datos acerca de cada producto disponible para venta, tal como el fabricante, el número del producto, su descripción y su precio.

ID_FAB	*ID_PRODUCTO*	*DESCRIPCION*	*PRECIO*	*EXISTENCIAS*
REI	2A45C	V Stago Trinquete	79	210
ACI	4100Y	Extractor	2.750	25
QSA	XK47	Reductor	355	38

BIC	41672	Plate	180	0
IMM	779C	Riostra 2-Tm	1.875	9
ACI	41003	Artículo Tipo 3	107	207
ACI	41004	Artículo Tipo 4	117	139
BIC	41003	Manivela	652	3
IMM	887P	Perno Riostra	250	24
QSA	XK48	Reductor	134	203
REI	2A44L	Bisagra Izqda.	4.500	12
FEA	112	Cubierta	148	115
IMM	887H	Soporte Riostra	54	223
BIC	41089	Reten	225	78
ACI	41001	Artículo Tipo 1	55	277
IMM	775C	Riostra 1-Tm	1.425	5
ACI	4100Z	Montador	2.500	28
QSA	XK48A	Reductor	117	37
ACI	41002	Artículo Tipo 2	76	167
REI	2A44R	Bisagra Dcha.	4.500	12
IMM	773C	Riostra 1/2-Tm	975	28
ACI	4100X	Ajustador	25	37
FEA	114	Bancada Motor	243	15
IMM	887X	Retenedor Riostra	475	32
REI	2A44G	Pasador Bisagra	350	14

Tabla 3.1.4: La tabla PRODUCTOS de la base de datos de ejemplo.

La tabla PEDIDOS (tabla 3.1.5) lleva la cuenta de cada pedido remitido por un cliente, identificando al vendedor que aceptó el pedido, el producto solicitado, la cantidad y el importe del pedido, etc. Por simplicidad, cada pedido atañe a un solo producto.

NUM_PEDIDO	*FECHA_PEDIDO*	*CLIE*	*REP*	*FAB*	*PRODUCTO*	*CANT*	*IMPORTE*
112961	17/12/1999	2117	106	REI	2A44L	7	31.500
113012	11/01/2000	2111	105	ACI	41003	35	3.745
112989	03/01/2000	2101	106	FEA	114	6	1.458
113051	10/02/2000	2118	108	QSA	XK47	4	1.420
112968	12/10/1999	2102	101	ACI	41004	34	3.978
110036	30/01/2000	2107	110	ACI	4100Z	9	22.500
113045	02/02/2000	2112	108	REI	2A44R	10	45.000
112963	17/12/1999	2103	105	ACI	41004	28	3.276
113013	14/01/2000	2118	108	BIC	41003	1	652
113058	23/02/2000	2108	109	FEA	112	10	1.480
112997	08/01/2000	2124	107	BIC	41003	1	652
112983	27/12/1999	2103	105	ACI	41004	6	702
113024	20/01/2000	2114	108	QSA	XK47	20	7.100
113062	24/02/2000	2124	107	FEA	114	10	2.430
112979	12/10/1999	2114	102	ACI	4100Z	6	15.000
113027	22/02/2000	2103	105	ACI	41002	54	4.104
113007	08/01/2000	2112	108	IMM	773C	3	2.825
113069	02/03/2000	2109	107	IMM	775C	22	31.350
113034	29/01/2000	2107	110	REI	2A45C	8	632
112992	04/11/1999	2118	108	ACI	41002	10	760
112975	12/10/1999	2111	103	REI	2A44G	6	2.100
113055	15/02/2000	2108	101	ACI	4100X	6	150
113048	10/02/2000	2120	102	IMM	779C	2	3.750
112993	04/01/2000	2106	102	REI	2A45C	24	1.896
113065	27/02/2000	2106	102	QSA	XK47	6	2.130
113003	25/01/2000	2108	109	IMM	779C	3	5.625
113049	10/02/2000	2118	108	QSA	XK47	2	776
112987	31/12/1999	2103	105	ACI	4100Y	11	27.500
113057	18/02/2000	2111	103	ACI	4100X	24	600

113042	02/02/2000	2113	101	REI	2A44R	5	22.500

Tabla 3.1.5: La tabla PEDIDOS de la base de datos de ejemplo.

3.2 Tipos de datos.

Existen tres grandes tipos de datos en SQL:

1. Numéricos. Representan cantidades numéricas, susceptibles de participar en cálculos aritméticos.

2. Alfanuméricos. Representan combinaciones de caracteres cualesquiera.

3. De tiempo. Representan magnitudes temporales (años, meses, días, etc.).

Dentro de cada tipo de datos hay varios subtipos. Vamos a describir ahora los dominios correspondientes a cada uno de ellos y cómo se definen.

3.2.1 Datos numéricos.

Existen tres subtipos de datos numéricos: enteros, decimales y en coma flotante.

Enteros.

Sus valores son números enteros, positivos o negativos. Existen dos tipos de enteros: los pequeños y los grandes. Los primeros se definen con la palabra *SMALLINT* y su dominio son los números enteros comprendidos entre –32768 y 32767. Los segundos se definen con la palabra *INTEGER* y su dominio son los números enteros entre –2147483648 y 2147483647.

Decimales.

Sus valores son números positivos o negativos que pueden tener parte entera y parte fraccionaria. Estos tipos de datos se definen como *DECIMAL(p,s)*, donde el primer entero p se llama precisión y es el número total de dígitos (parte entera y parte fraccionaria) que puede tener un valor. El segundo entero es el número de dígitos que puede tener la parte fraccionaria y se conoce con el nombre de escala. El dominio de un DECIMAL(p,s) es el conjunto de todos los números racionales, positivos y negativos, formados por p dígitos de los que s son fraccionarios.

En coma flotante.

Sus valores representan números positivos o negativos que pueden tener parte entera y parte fraccionaria, como los datos de tipo decimal, pero con más cifras. Este tipo de datos se definen con la palabra *FLOAT* o la palabra *REAL* (ambas son equivalentes). Su dominio es el conjunto de números racionales que son aproximaciones a los números reales comprendidos entre ciertos límites (generalmente los límites son los números cuyo valor absoluto esta comprendido entre 1.175494351e-38 y 3.402823466e38).

Internamente están codificados en lo que en términos informáticos se denomina coma flotante, lo que conlleva que puedan aparecer decimales inesperados en algunas operaciones. Así, puede ocurrir que en la suma de dos valores que debieran dar un número entero aparezcan decimales (por ejemplo 99.99999 en vez de 100). Esto hace a este tipo de datos menos adecuado, en general, que el tipo DECIMAL para aplicaciones comerciales, contables o financieras. En cambio, son más adecuados para aplicaciones científicas o de ingeniería en que se necesite manejar números muy grandes.

3.2.2 Datos alfanuméricos.

Se utilizan para almacenar descripciones, nombres de personas, direcciones, etc. En general información de tipo textual o descriptiva. Sus valores son secuencias de caracteres cualesquiera, tomados del conjunto de caracteres disponible. Al número de caracteres que forman un valor alfanumérico se le denomina longitud. Los datos de tipo alfanumérico pueden ser de dos subtipos: de longitud fija o de longitud variable.

De longitud fija.

Este tipo de datos se define como *CHAR(n)*, donde n indica la longitud. Su dominio es el de todas las secuencias de caracteres de longitud n. La longitud n debe estar comprendida entre 1 y 254, ambos inclusive.

De longitud variable.

Se definen como *VARCHAR(n)*. Su dominio es el de todas las secuencias de caracteres de longitud comprendida entre 0 y n. Una secuencia de longitud cero se llama el valor vacío. La longitud máxima suele ser 32767.

3.2.3 Datos de tiempo.

Sus valores son números que representan un punto en el tiempo con una precisión de días, segundos o microsegundos. Existen tres subtipos:

Fechas.

Se definen con la palabra *DATE* y representan un día determinado. Sus valores constan de tres partes, todas ellas números enteros sin signo. La primera parte tiene cuatro dígitos y representa un número de año. La segunda tiene dos dígitos y representa el mes y la tercera, también de dos dígitos, representa el día.

Su dominio es el conjunto de todas las fechas validas según el calendario gregoriano, empezando en el día 1 de Enero del año 1 y terminando el 31 de Diciembre del año 9999, ambos inclusive.

Horas.

Se definen con la palabra *TIME* y representan una hora determinada con precisión de segundos. Sus valores constan de tres partes, todas ellas números enteros de dos dígitos sin signo. La primera parte representa la hora, la segunda representa el

minuto y la tercera los segundos. Si la hora tiene el valor 24, la segunda y tercera partes deben valer cero.

Su dominio es de todos los valores de horas, minutos y segundos válidos en un reloj de 24 horas.

Instante.

Se definen con la palabra *TIMESTAMP* y representan un punto en el tiempo con precisión de microsegundos. Para ello sus valores constan de siete partes, todas ellas números enteros sin signo que representan año, mes, día, hora, minuto, segundo y microsegundo. Sus valores son iguales a los de DATE y TIME y los microsegundos tienen una precisión de 6 dígitos.

Su dominio es el conjunto de valores que identifican todos los microsegundos comprendidos entre el 1 de Enero del año 1 y el 31 de Diciembre del año 9999, ambos inclusive.

3.3 Constantes.

Entre los elementos componentes de una sentencia SQL, se encuentran las constantes, también llamadas literales. Son secuencias de caracteres que representan un valor determinado, numérico o alfanumérico. Las constantes pueden ser igual que los tipos numéricas, alfanuméricas o de tiempo.

Las numéricas son números enteros positivos o negativos (10, +10, -10), números decimales (3.04, +3.04, -3.04) o números en notación exponencial (3.14e-5).

Las constantes alfanuméricas son secuencias de caracteres cualesquiera que empiezan y terminan con un apóstrofe (‘ABC’, ‘PEPE’’S’,’10.23’).

Las constantes de tiempo pueden representarse como dd/mm/aaaa ó aaaa/mm/dd, etc.

3.4 Valores nulos.

Sea cual sea el tipo de datos asignado a una columna, se puede permitir o no entre sus valores posibles uno especial denominado *nulo*. El valor nulo se especifica con la palabra predefinida *NULL*.

El valor nulo puede interpretarse como “desconocido”. Se utiliza en tablas en las que puede haber filas para las que no se conozcan los valores de todas sus columnas, o alguna de éstas no sea aplicable al concepto representado.

En una columna de tipo numérico el valor nulo no es lo mismo que el valor cero. En una columna de tipo alfanumérico, el nulo no es lo mismo que blanco o que el valor vacío (longitud cero).

3.5 Consultas simples.

Las consultas son el corazón del lenguaje SQL. La sentencia *SELECT*, que se utiliza para expresar consultas en SQL, es la más potente y compleja de las sentencias de SQL. Empezaremos en este punto con las consultas más simples para pasar posteriormente a consultas más complicadas.

3.5.1 La sentencia SELECT.

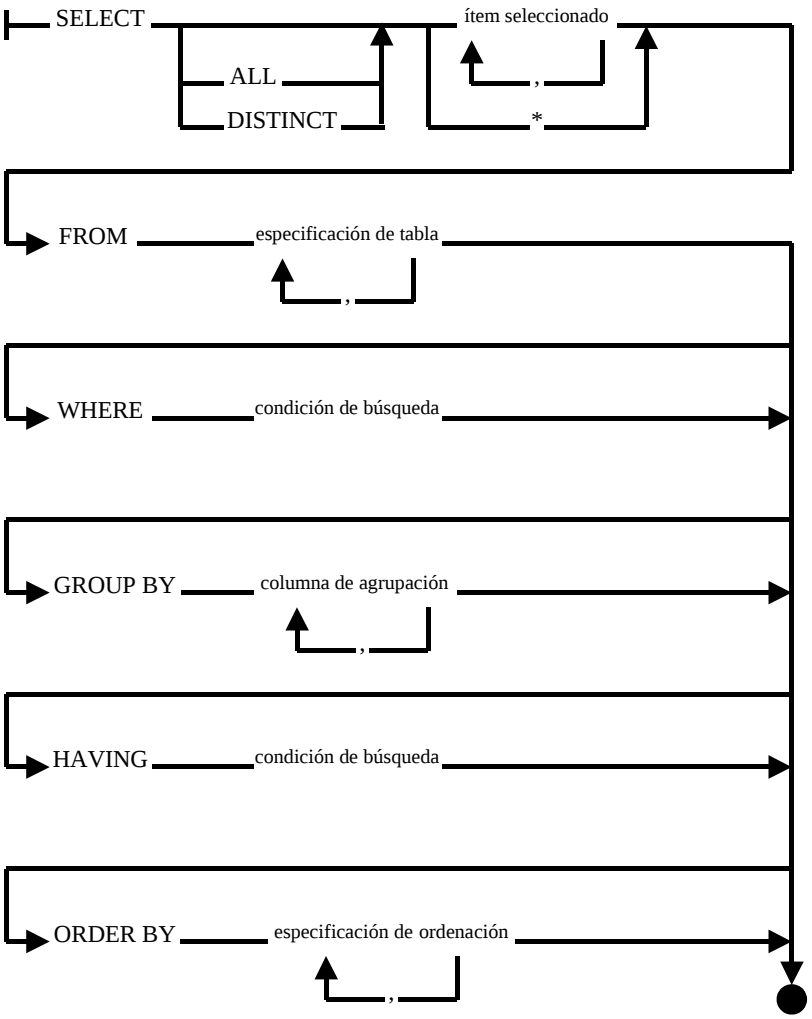


Figura 3.5.1.1: Diagrama sintáctico de la sentencia SELECT.

La sentencia SELECT recupera datos de una base de datos y los devuelve en forma de resultados de la consulta. El diagrama sintáctico de SELECT puede verse en la figura 3.5.1.1. Las cláusulas *SELECT* y *FROM* de la sentencia son necesarias. Las cuatro cláusulas restantes son opcionales.

La cláusula *SELECT* que inicia cada sentencia SELECT especifica los ítems de datos a recuperar por la consulta. Los ítems se especifican generalmente mediante una lista 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 puede ser:

- 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 la

base de datos y lo coloca en la fila correspondiente de los resultados de la consulta.

- El valor *, indicando todas las columnas que forman la tabla.

- Una constante, especificando que el 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.

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.

3.5.2 Resultados de consultas.

El resultado de una consulta SQL es siempre una tabla de datos, semejante a las tablas de base de datos. Generalmente los resultados de la consulta formarán una tabla con varias columnas y filas. Por ejemplo, la consulta siguiente forma una tabla de tres columnas (ya que se piden tres ítems de datos) y diez filas (ya que hay diez vendedores):

```
SELECT nombre,oficina_rep,contrato FROM repventas
```

NOMBRE	*OFICINA_REP*	*CONTRATO*
Jose Maldonado	11	14/06/1998
Carlos Martinez	12	19/05/1997
Belen Aguirre	13	12/02/1998
Maria Garcia	11	12/10/1999
Lorenzo Fernandez	21	12/10/1999
Soledad Martinez	21	10/12/1996
Daniel Gutierrez	12	20/10/1996
Antonio Valle	NULL	13/01/2000
Pedro Cruz	12	01/03/1997
Natalia Martin	22	14/11/1998

Tabla 3.5.2.1: Listado de los nombres, oficinas y fechas de contrato de todos los vendedores.

En contraste, la consulta siguiente produce una única fila, ya que sólo hay un vendedor que tenga el número de empleado solicitado. Aún cuando esta única fila de resultados tenga un aspecto menos “tabular” que los resultados multifíla, SQL sigue considerándolo una tabla de tres columnas y una fila.

```
SELECT nombre,cuota,ventas FROM repventas WHERE num_empl=107
```

NOMBRE	*CUOTA*	*VENTAS*
Natalia Martin	27500	34432

Tabla 3.5.2.2: Nombre, cuota y ventas del empleado número 107.

En algunos casos los resultados de la consulta pueden ser un único valor, como el siguiente ejemplo:

```
SELECT AVG(ventas) FROM repventas
```

AVG(ventas)
24758.20

Tabla 3.5.2.3: Promedio de ventas de los vendedores.

E incluso en otros puede producir cero filas de resultados como en este ejemplo:

```
SELECT nombre,contrato FROM repventas WHERE ventas>75000
```

NOMBRE	*CONTRATO*

Tabla 3.5.2.4: Nombre y fecha de contrato de cualquier vendedor con ventas superiores a 75000.

Aún en esta situación, el resultado sigue siendo una tabla. Se trata de una tabla vacía con dos columnas y cero filas.

3.5.3 Consultas sencillas.

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

```
SELECT ciudad,region,ventas FROM oficinas
```

CIUDAD	*REGION*	*VENTAS*
Toledo	Centro	34.432
Valencia	Este	40.063
Barcelona	Este	29.328
Alicante	Este	39.327
Madrid	Centro	81.309

Tabla 3.5.3.1: Lista de la población, región y ventas de cada oficina de ventas.

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.

```
SELECT ciudad,region,(ventas-objetivo) FROM oficinas
```

CIUDAD	*REGION*	*VENTAS-OBJETIVO*
Toledo	Centro	6.932
Valencia	Este	-12.437
Barcelona	Este	-40.672
Alicante	Este	9.327
Madrid	Centro	21.309

Tabla 3.5.3.2: Lista de la ciudad, región y el resultado de ventas para cada oficina.

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. La tercera columna se calcula, fila a fila, utilizando

los valores de datos de la fila actual de la tabla OFICINAS. Veamos otros ejemplos de columnas calculadas.

```
SELECT id_fab,id_producto,descripcion,(existencias*precio) FROM productos
```

ID_FAB	*ID_PRODUCTO*	*DESCRIPCION*	*EXISTENCIAS*PRECIO*
REI	2A45C	V Stago Trinquete	16.590
ACI	4100Y	Extractor	68.750
QSA	XK47	Reductor	13.490
BIC	41672	Plate	0
IMM	779C	Riostra 2-Tm	16.875
ACI	41003	Artículo Tipo 3	22.149
ACI	41004	Artículo Tipo 4	16.263
BIC	41003	Manivela	1.956
IMM	887P	Perno Riostra	6.000
QSA	XK48	Reductor	27.202
REI	2A44L	Bisagra Izqda.	54.000
FEA	112	Cubierta	17.020
IMM	887H	Soporte Riostra	12.042
BIC	41089	Reten	17.550
ACI	41001	Artículo Tipo 1	15.235
IMM	775C	Riostra 1-Tm	7.125
ACI	4100Z	Montador	14.000
QSA	XK48A	Reductor	4.329
ACI	41002	Artículo Tipo 2	12.692
REI	2A44R	Bisagra Dcha.	54.000
IMM	773C	Riostra 1/2-Tm	27.300
ACI	4100X	Ajustador	925
FEA	114	Bancada Motor	3.645
IMM	887X	Retenedor Riostra	15.200
REI	2A44G	Pasador Bisagra	4.900

Tabla 3.5.3.3: Valor del inventario de cada producto.

```
SELECT nombre,cuota,(cuota+(0.03*ventas)) FROM repventas
```

NOMBRE	*CUOTA*	*CUOTA+(.03*VENTAS)*
Jose Maldonado	25.000	25.988,74
Carlos Martinez	17.500	17.500,00
Belen Aguirre	30.000	31.179,81
Maria Garcia	27.500	27.713,15
Lorenzo Fernandez	30.000	31.755,99
Soledad Martinez	30.000	30.683,28
Daniel Gutierrez	27.500	28.298,84
Antonio Valle	NULL	NULL
Pedro Cruz	25.000	25.081,00
Natalia Martin	27.500	28.532,96

Tabla 3.5.3.4: Resultado de elevar la cuota de ventas un 3% a cada vendedor.

```
SELECT nombre,MONTH(contrato),YEAR(contrato) FROM repventas
```

NOMBRE	*MONTH(contrato)*	*YEAR(contrato)*
Jose Maldonado	6	1998
Carlos Martinez	5	1997
Belen Aguirre	2	1998
Maria Garcia	10	1999
Lorenzo Fernandez	10	1999
Soledad Martinez	12	1996
Daniel Gutierrez	10	1996

Antonio Valle	1	2000
Pedro Cruz	3	1997
Natalia Martin	11	1998

Tabla 3.5.3.5: Mes y año de contrato de cada vendedor.

También se pueden utilizar constantes de 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, como por ejemplo:

```
SELECT ciudad,'tiene ventas de',ventas FROM oficinas
```

CIUDAD	*'tiene ventas de'*	*VENTAS*
Toledo	tiene ventas de	34.432
Valencia	tiene ventas de	40.063
Barcelona	tiene ventas de	29.328
Alicante	tiene ventas de	39.327
Madrid	tiene ventas de	81.309

Tabla 3.5.3.6: Lista de ventas para cada ciudad.

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. SQL permite utilizar un asterisco (*) en lugar de la lista de selección como abreviatura de "todas las columnas":

```
SELECT * FROM oficinas
```

OFICINA	*CIUDAD*	*REGION*	*DIR*	*OBJETIVO*	*VENTAS*
22	Toledo	Centro	108	27.500	34.432
11	Valencia	Este	106	52.500	40.063
12	Barcelona	Este	104	70.000	29.328
13	Alicante	Este	105	30.000	39.327
21	Madrid	Centro	108	60.000	81.309

Tabla 3.5.3.7: Todos los datos de la tabla oficinas.

3.5.4 Filas duplicadas.

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 tiene un valor diferente en cada fila). Si no se incluye la clave primaria en los resultados pueden producirse filas duplicadas. Por ejemplo, supongamos la siguiente petición:

```
SELECT region FROM oficinas
```

REGION
Centro
Este
Este
Este
Centro

Tabla 3.5.4.1: Lista de las regiones en que se encuentran las oficinas.

Los resultados tienen cinco filas (uno por oficina), pero se encuentran filas duplicadas una de la otra. Estos resultados no son probablemente los que uno pretende

con la consulta, si existen dos regiones diferentes, cabría esperar que aparecieran solo las dos regiones.

Se pueden eliminar filas duplicadas de los resultados de la consulta insertando la palabra clave *DISTINCT* en la sentencia SELECT justa antes de la lista de selección. Veamos la consulta anterior modificada:

```
SELECT DISTINCT region FROM oficinas
```

REGION
Este
Centro

Tabla 3.5.4.2: Lista de las regiones en que se encuentran las 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. Si se omite la palabra *DISTINCT*, SQL no elimina fila duplicadas. También se puede especificar la palabra clave *ALL* para indicar que las filas duplicadas sean mostradas, pero es innecesario ya que éste es el comportamiento por defecto.

3.5.5 Selección de fila.

Las consultas SQL que recuperan todas las filas de una tabla únicamente son útiles para generar informes sobre la base de datos. Generalmente se desea seleccionar solamente parte de las filas de una tabla, y sólo se incluirán esas filas en los resultados. La cláusula *WHERE* se emplea para especificar las filas que se desean recuperar. Veamos unos ejemplos simples:

```
SELECT ciudad,ventas,objetivo FROM oficinas WHERE ventas>objetivo
```

CIUDAD	*VENTAS*	*OBJETIVO*
Toledo	34.432	27.500
Alicante	39.327	30.000
Madrid	81.309	60.000

Tabla 3.5.5.1: Oficinas donde las ventas exceden al objetivo.

```
SELECT nombre,ventas,cuota FROM repventas WHERE num_empl=105
```

NOMBRE	*VENTAS*	*CUOTA*
Belen Aguirre	39.327	30.000

Tabla 3.5.5.2: Nombre, ventas y cuota del empleado número 105.

```
SELECT nombre,ventas FROM repventas WHERE director=104
```

NOMBRE	*VENTAS*
Belen Aguirre	39.327
Daniel Gutierrez	26.628
Pedro Cruz	2.700

Tabla 3.5.5.3: Empleados dirigidos por el empleado número 104.

La cláusula WHERE consta de la palabra clave *WHERE* seguida de una condición de búsqueda que especifica las filas a recuperar. Conceptualmente, SQL recorre cada fila de la tabla seleccionada, una a una, y aplica la condición de búsqueda. Esta condición de búsqueda puede producir uno de los tres resultados:

- Si la condición de búsqueda es cierta, la fila se incluye en los resultados de la consulta.
- Si la condición de búsqueda es falsa, la fila se excluye de los resultados de la consulta.
- Si la condición de búsqueda tiene un valor NULL (desconocido), la fila se excluye de los resultados de la consulta.

3.5.6 Condiciones de búsqueda.

SQL ofrece un rico conjunto de condiciones de búsqueda que permiten especificar muchos tipos diferentes de consultas eficaz y naturalmente. Las condiciones de búsqueda son cinco según el estándar ANSI/ISO:

- 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).

Test de comparación.

La condición de búsqueda más común utilizada en una consulta SQL es la 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. SQL ofrece seis modos de comparar dos expresiones (=, <>, <, <=, >, >=). Veamos algunos ejemplos:

```
SELECT nombre FROM repventas WHERE contrato<'01/01/1998'
```

NOMBRE
Carlos Martinez
Soledad Martinez
Daniel Gutierrez
Pedro Cruz

Tabla 3.5.6.1: Lista los vendedores contratados antes de 1998.

SELECT ciudad,ventas,objetivo FROM oficinas WHERE ventas<(0.75*objetivo)

CIUDAD	*VENTAS*	*OBJETIVO*
Barcelona	29.328	70.000

Tabla 3.5.6.2: Lista las oficinas cuyas ventas están por debajo del 75% del objetivo.

SELECT ciudad,region FROM oficinas WHERE region<>'Centro'

CIUDAD	*REGION*
Valencia	Este
Barcelona	Este
Alicante	Este

Tabla 3.5.6.3: Lista de las oficinas que no se encuentran en la región Centro.

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 clave primaria, el test aísla una sola fila de la tabla, produciendo una sola fila de resultados, como en el ejemplo siguiente:

SELECT empresa,limite_credito FROM clientes WHERE num_clie=2107

EMPRESA	*LIMITE_CREDITO*
Distribuciones Sur S.A.	35.000

Tabla 3.5.6.4: Nombre y limite de crédito del cliente número 2107.

Observe que la sentencia SQL que recupera un cliente específico por su número y la que recupera todos los clientes que tienen una cierta característica es exactamente de la misma forma. Estos dos tipos de consulta serían operaciones muy diferentes en una base de datos no relacional. Considere además, que SQL maneja lógica trivaluada, por lo cual es necesario considerar el manejo del valor NULL como en el ejemplo siguiente:

SELECT nombre FROM repventas WHERE ventas>cuota

NOMBRE
Jose Maldonado
Belen Aguirre
Lorenzo Fernandez
Natalia Martin

Tabla 3.5.6.5: Lista de vendedores que superan sus cuotas.

SELECT nombre FROM repventas WHERE ventas<=cuota

NOMBRE
Carlos Martinez
Maria Garcia
Soledad Martinez
Daniel Gutierrez
Pedro Cruz

Tabla 3.5.6.6: Lista de vendedores que no superan sus cuotas.

Como puede observarse, solo se recuperan nueve filas, pues una de las filas originales no posee una cuota establecida (valor NULL).

Test de rango.

SQL proporciona una forma diferente de condición de búsqueda con el test de rango (*BETWEEN*). El test de rango comprueba si el valor del dato se encuentre entre dos valores especificados. Un ejemplo de test de rango es el siguiente:

```
SELECT num_pedido,fecha_pedido,producto,importe FROM pedidos WHERE
fecha_pedido BETWEEN '01/10/1999' AND '31/12/1999'
```

NUM_PEDIDO	*FECHA_PEDIDO*	*PRODUCTO*	*IMPORTE*
112961	17/12/1999	2A44L	31500
112968	12/10/1999	41004	3.978
112963	17/12/1999	41004	3.276
112983	27/12/1999	41004	702
112979	12/10/1999	4100Z	15.000
112992	04/11/1999	41002	760
112975	12/10/1999	2A44G	2.100
112987	31/12/1999	4100Y	27500

Tabla 3.5.6.7: Relación de pedidos entre dos fechas.

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. Veamos otro ejemplo de test de rango:

```
SELECT num_pedido,importe FROM pedidos WHERE importe BETWEEN 20000.00
AND 29999.99
```

NUM_PEDIDO	*IMPORTE*
110036	22.500
112987	27.500
113042	22.500

Tabla 3.5.6.8: Pedidos que caen en un determinado rango de importe.

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

```
SELECT nombre,ventas,cuota FROM repventas WHERE ventas NOT BETWEEN
(0.75*cuota) AND (1.50*cuota)
```

NOMBRE	*VENTAS*	*CUOTA*
Carlos Martinez	0	17.500
Maria Garcia	7.105	27.500
Lorenzo Fernadez	58.533	30.000
Pedro Cruz	2.700	25.000

Tabla 3.5.6.9: Lista de vendedores cuyas ventas no están entre el 75 y el 150% de su cuota.

Es necesario señalar que la sentencia BETWEEN no añade potencia expresiva a SQL, pues *A BETWEEN B AND C* puede ser expresado como *(A>=B) AND (A<=C)*.

Test de pertenencia a conjunto.

Otra condición habitual es el test de pertenencia a conjunto (*IN*). Examina si un valor de dato coincide con uno de una lista de valores objetivo. Veamos varios ejemplos:

```
SELECT nombre,cuota,ventas FROM repventas WHERE oficina_rep IN (11,13,22)
```

NOMBRE	*CUOTA*	*VENTAS*
Jose Maldonado	25.000	32.958
Belen Aguirre	30.000	39.327
Maria Garcia	27.500	7.105
Natalia Martin	27.500	34.432

Tabla 3.5.6.10: Lista de los vendedores de las oficinas 11, 13 y 22.

```
SELECT num_pedido,rep,importe FROM pedidos WHERE rep IN (101,103,107,109)
```

NUM_PEDIDO	*REP*	*IMPORTE*
112968	101	3.978
113058	109	1.480
112997	107	652
113062	107	2.430
113069	107	31.350
112975	103	2.100
113055	101	150
113003	109	5.625
113057	103	600
113042	101	22.500

Tabla 3.5.6.11: Pedidos obtenidos por cuatro vendedores específicos.

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 equivalente a *(X=A) OR (X=B) OR (X=C)*.

Test de correspondencia con patrón.

Se puede utilizar un test de comparación simple para recuperar las filas en donde el contenido de una columna de texto se corresponde con un cierto texto particular. Por ejemplo, esta consulta recupera la fila de la tabla clientes por nombre:

```
SELECT empresa,limite_credito FROM clientes WHERE empresa='AFS S.A.'
```

EMPRESA	*LIMITE_CREDITO*
AFS S.A.	20.000

Tabla 3.5.6.12: Limite de crédito de la empresa AFS S.A.

Sin embargo, uno puede olvidar fácilmente el nombre exacto de la empresa. El test de correspondencia con patrón (*LIKE*) comprueba si el valor de dato de una columna se ajusta a un patrón especificado. El patrón es una cadena que puede incluir uno o más caracteres comodines. Estos caracteres comodines son el signo de porcentaje (%) y el subrayado (_).

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

Así por ejemplo, uno puede consultar todos los vendedores de apellido Martinez de la siguiente forma:

```
SELECT num_empl,nombre FROM repventas WHERE nombre LIKE '% Martinez'
```

NUM_EMPL	*NOMBRE*
104	Carlos Martinez
102	Soledad Martinez

Tabla 3.5.6.13: Empleados apellidados Martinez.

Uno de los problemas de la correspondencia con patrones de cadenas es cómo hacer corresponder los propios caracteres comodines como caracteres literales. Para comprobar la presencia de un carácter comodín en una cadena se precede de un carácter de escape que hace que el carácter que le sigue inmediatamente se trate como un literal en lugar de cómo un carácter comodín. El carácter escape en SQL es un signo de dólar ($).

Test de valor nulo.

En algunas ocasiones es útil comprobar explícitamente los valores NULL en una condición de búsqueda y manejarlos directamente. SQL proporciona un test especial de valor nulo (*IS NULL*) para tratar este caso. Por ejemplo, la consulta siguiente utiliza el test de valor nulo para hallar los vendedores a los que no se les ha asignado una oficina:

```
SELECT nombre FROM repventas WHERE oficina_rep IS NULL
```

NOMBRE
Antonio Valle

Tabla 3.5.6.14: Vendedores que aún no tienen asignada una oficina.

La forma negada del test de valor nulo (*IS NOT NULL*) encuentra las filas que no contienen un valor NULL:

```
SELECT nombre FROM repventas WHERE oficina_rep IS NOT NULL
```

NOMBRE
Jose Maldonado
Carlos Martinez
Belen Aguirre
Maria Garcia
Lorenzo Fernandez
Soledad Martinez
Daniel Gutierrez
Pedro Cruz
Natalia Martin

Tabla 3.5.6.15: Vendedores que tienen asignada una oficina.

3.5.7 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 trivaluada, se pueden combinar estas condiciones de búsqueda para formar otras más complejas. Las condiciones de búsqueda compuesta se crean con las palabras clave *AND*, *OR* y *NOT*.

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

```
SELECT nombre,cuota,ventas FROM repventas WHERE ventas<cuota OR
ventas<30000.00
```

NOMBRE	*CUOTA*	*VENTAS*
Carlos Martinez	17.500	0
Maria Garcia	27.500	7.105
Soledad Martinez	30.000	22.776
Daniel Gutierrez	27.500	26.628
Antonio Valle	NULL	23.123
Pedro Cruz	25.000	2.700

Tabla 3.5.7.1: Vendedores por debajo de la cuota o con ventas inferiores a 30000.

La palabra clave *AND* se utiliza para combinar dos condiciones de búsqueda que deban ser ciertas simultáneamente:

```
SELECT nombre,cuotas,ventas FROM repventas WHERE ventas<cuota AND
ventas<20000.00
```

NOMBRE	*CUOTA*	*VENTAS*
Carlos Martinez	17.500	0
Maria Garcia	27.500	7.105
Pedro Cruz	25.000	2.700

Figura 3.5.7.2: Vendedores por debajo de la cuota y con ventas inferiores a 20000.

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

```
SELECT nombre,cuota,ventas FROM repventas WHERE ventas<cuota AND NOT
ventas>5000.00
```

NOMBRE	*CUOTA*	*VENTAS*
Carlos Martinez	17.500	0
Pedro Cruz	25.000	2.700

Tabla 3.5.7.3: Vendedores que están por debajo de la cuota y con ventas no superiores a 5000.

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:

```
SELECT nombre,edad FROM repventas WHERE (ventas<cuota AND edad>45) OR
ventas<5000.00
```

NOMBRE	*EDAD*
Carlos Martinez	33

Soledad Martinez	48
Pedro Cruz	29

Tabla 3.5.7.4: Vendedores con ventas inferiores a la cuota y edad mayor de 45 años o con ventas inferiores a 5000.

3.5.8 Ordenación 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 clave *ORDER BY* seguidas de una lista de especificaciones de ordenación separadas por comas. Por ejemplo, los resultados de la consulta siguiente están ordenados por las columnas región y ciudad.

```
SELECT ciudad,region,ventas FROM oficinas ORDER BY region,ciudad
```

CIUDAD	*REGION*	*VENTAS*
Madrid	Centro	81.309
Toledo	Centro	34.432
Alicante	Este	39.327
Barcelona	Este	29.328
Valencia	Este	40.063

Tabla 3.5.8.1: Ventas de cada oficina, ordenadas en orden alfabético por región, y dentro de cada región por ciudad.

La primera especificación de ordenación (región) es la clave de 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.

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

```
SELECT ciudad,region,ventas FROM oficinas ORDER BY ventas DESC
```

CIUDAD	*REGION*	*VENTAS*
Madrid	Centro	81.309
Valencia	Este	40.063
Alicante	Este	39.327
Toledo	Centro	34.432
Barcelona	Este	29.328

Figura 3.5.8.2: Lista de las oficinas clasificadas en orden descendente de ventas.

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 el siguiente ejemplo:

```
SELECT ciudad,region,(ventas-objetivo) FROM oficinas ORDER BY 3 DESC
```

CIUDAD	*REGION*	*VENTAS-OBJETIVO*

Madrid	Centro	21.309
Alicante	Este	9.327
Toledo	Centro	6.932
Valencia	Este	-12.437
Barcelona	Este	-40.672

Figura 3.5.8.3: Lista de las oficinas clasificadas en orden descendente de rendimiento de ventas.

3.5.9 Combinación de resultados de consulta.

Ocasionalmente, es conveniente combinar los resultados de dos o más consultas en una única tabla de resultados totales. SQL soporta esta capacidad gracias a la característica *UNION* de la sentencia SELECT. Por ejemplo, supongamos que queremos resolver la petición: "Lista todos los productos en donde el precio del producto exceda de 2000 o en donde más de 30000 del producto hayan sido ordenados en un solo pedido".

La primera parte de la petición puede satisfacerse con la consulta:

```
SELECT id_fab,id_producto FROM productos WHERE precio>2000
```

ID_FAB	***ID_PRODUCTO***
ACI	4100Y
REI	2A44L
ACI	4100Z
REI	2A44R

Tabla 3.5.9.1: Productos cuyo precio excede de 2000.

Análogamente, la segunda parte de la petición puede satisfacerse con la consulta:

```
SELECT DISTINCT fab,producto FROM pedidos WHERE importe>30000
```

FAB	***PRODUCTO***
IMM	775C
REI	2A44L
REI	2A44R

Tabla 3.5.9.2: Productos de los cuales se ha hecho un pedido mayor de 30000.

La unión de ambas consultas se realiza de la siguiente forma:

```
SELECT id_fab,id_producto FROM productos WHERE precio>2000 UNION SELECT
DISTINCT fab,producto FROM pedidos WHERE importe>30000
```

ID_FAB	***ID_PRODUCTO***
ACI	4100Y
ACI	4100Z
IMM	775C
REI	2A44L
REI	2A44R

Tabla 3.5.9.3: Resultado de la unión de las dos consultas anteriores.

Para poder realizar una unión existen varias restricciones:

- 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 en la columna correspondiente en la segunda tabla.

- Ninguna de las dos tablas puede estar ordenadas con la cláusula ORDER BY.

Puede observarse que la *UNION* combina las filas de los dos conjuntos de resultados eliminando por defecto las filas duplicadas. Si se desean mantener las filas duplicadas en una operación *UNION*, se puede especificar la palabra clave *ALL* a continuación de la palabra *UNION*. Por ejemplo:

```
SELECT id_fab,id_producto FROM productos WHERE precio>2000 UNION ALL
SELECT DISTINCT fab,producto FROM pedidos WHERE importe>30000
```

ID_FAB	*ID_PRODUCTO*
ACI	4100Y
REI	2A44L
ACI	4100Z
REI	2A44R
IMM	775C
REI	2A44L
REI	2A44R

Tabla 3.5.9.4: Resultado de la unión de las consultas anteriores conservando las filas repetidas.

Como hemos dicho, las sentencias SELECT combinadas no pueden estar ordenadas. Sin embargo, el resultado de la *UNION* puede ordenarse mediante una cláusula *ORDER BY* que ordene el resultado. Por ejemplo, la consulta anterior puede ordenarse de la siguiente forma:

```
SELECT id_fab,id_producto FROM productos WHERE precio>2000 UNION SELECT
DISTINCT fab,producto FROM pedidos WHERE importe>30000 ORDER BY 2
```

ID_FAB	*ID_PRODUCTO*
REI	2A44L
REI	2A44R
ACI	4100Y
ACI	4100Z
IMM	775C

Tabla 3.5.9.5: Resultado de la unión de las consultas ordenado por la identificación del producto.

3.6 Consultas multitabla (composiciones).

Muchas consultas útiles solicitan datos procedentes de dos o más tablas en la base de datos. Por ejemplo, las siguientes consultas extraen los datos de varias tablas:

- Lista los vendedores y las oficinas en donde trabajan (tablas REPVENTAS y OFICINAS).

- Lista los pedidos mostrando el importe del pedido, el nombre del cliente que lo ordenó y el nombre del producto solicitado (tablas PEDIDOS, CLIENTES y REPVENTAS).

- Muestra todas las órdenes aceptadas por vendedores de la región Este, mostrando la descripción del producto y el vendedor (tablas PEDIDOS, REPVENTAS, OFICINAS y PRODUCTOS).

SQL permite recuperar datos que responden a estas peticiones mediante consultas multitabla que componen (join) datos procedentes de dos o más tablas. Por ejemplo, la consulta "lista todos los pedidos, mostrando su número, importe, nombre del cliente y el límite de crédito del cliente" se realiza de la siguiente forma:

```
SELECT num_pedido,importe,empresa,limite_credito FROM pedidos,clientes WHERE
clie=num_clie
```

NUM_PEDIDO	*IMPORTE*	*EMPRESA*	*LIMITE_CREDITO*
112961	31.500	Hnos. Ramon S.L.	35.000
113012	3.745	EVBE S.A.	50.000
112989	1.458	Exclusivas Soriano S.A.	65.000
113051	1.420	Exclusivas Norte S.A.	60.000
112968	3.978	Exclusivas del Este S.L.	65.000
110036	22.500	Distribuciones Sur S.A.	35.000
113045	45.000	Lopez Asociados S.L.	50.000
112963	3.276	Pino S.L.	50.000
113013	652	Exclusivas Norte S.A.	60.000
113058	1.480	Zapater Importaciones S.A.	55.000
112997	652	Domingo S.L.	40.000
112983	702	Pino S.L.	50.000
113024	7.100	Componentes Fernandez S.A.	20.000
113062	2.430	Domingo S.L.	40.000
112979	15.000	Componentes Fernandez S.A.	20.000
113027	4.104	Pino S.L.	50.000
113007	2.825	Lopez Asociados S.L.	50.000
113069	31.350	Roda & Castedo S.L.	25.000
113034	632	Distribuciones Sur S.A.	35.000
112992	760	Exclusivas Norte S.A.	60.000
112975	2.100	EVBE S.A.	50.000
113055	150	Zapater Importaciones S.A.	55.000
113048	3.750	Distribuciones Montiel S.L.	50.000
112993	1.896	Construcciones Leon S.A.	65.000
113065	2.130	Construcciones Leon S.A.	65.000
113003	5.625	Zapater Importaciones S.A.	55.000
113049	776	Exclusivas Norte S.A.	60.000
112987	27.500	Pino S.L.	50.000
113057	600	EVBE S.A.	50.000
113042	22.500	Importaciones Martin S.L.	20.000

Tabla 3.6.1: Ejemplo de consulta de dos tablas.

Los cuatro datos de la consulta anterior están almacenados en dos tablas:

- 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 no la información referente a los pedidos.

Sin embargo, existe 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.

El proceso de formar parejas de filas haciendo coincidir los contenidos de las columnas relacionadas se denomina componer (joining) las tablas. La tabla resultante se denomina composición entre las tablas. Las composiciones son el fundamento del procesamiento de consultas multitabla en SQL. Todos los datos de la base relacional están almacenados en sus columnas con valores explícitos, de modo que todas las relaciones posibles entre tablas puedan formarse comparando los contenidos de las columnas relacionadas.

3.6.1 Consultas padre/hijo.

Las consultas multitabla más comunes implican a dos tablas que tienen la relación natural padre/hijo. La consulta referente a pedidos y clientes anterior 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 padre/hijo.

Las claves foráneas y las claves primarias crean relaciones padre/hijo en una base de datos SQL. La tabla que contiene la clave foránea 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 foránea y la clave primaria. Otro ejemplo de una consulta padre/hijo es la siguiente:

```
SELECT nombre,ciudad,region FROM repventas,oficinas WHERE oficina_rep=oficina
```

NOMBRE	*CIUDAD*	*REGION*
Jose Maldonado	Valencia	Este
Carlos Martinez	Barcelona	Este
Belen Aguirre	Alicante	Este
Maria Garcia	Valencia	Este
Lorenzo Fernandez	Madrid	Centro
Soledad Martinez	Madrid	Centro
Daniel Gutierrez	Barcelona	Este
Pedro Cruz	Barcelona	Este
Natalia Martin	Toledo	Centro

Tabla 3.6.1.1: Lista de los vendedores y la ciudad y región donde trabajan.

Como puede observarse, SQL no requiere que las columnas de emparejamiento sean incluidas en los resultados de la consulta multitabla. Con frecuencia son omitidas pues las claves primarias y las claves foráneas suelen ser números de identificación difíciles de recordar y poco descriptivos.

3.6.2 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. Por ejemplo, supongamos la consulta anterior que se desea restringirla mostrando únicamente los vendedores de las oficinas con objetivos de ventas iguales o mayores de 60000.

```
SELECT nombre,ciudad,region FROM repventas,oficinas WHERE oficina_rep=oficina
AND objetivo>=60000
```

NOMBRE	*CIUDAD*	*REGION*
Carlos Martinez	Barcelona	Este
Lorenzo Fernandez	Madrid	Centro
Soledad Martinez	Madrid	Centro
Daniel Gutierrez	Barcelona	Este
Pedro Cruz	Barcelona	Este

Tabla 3.6.2.1: Lista de los vendedores de las oficinas con objetivo igual o superior a 60000.

Con la condición de búsqueda adicional, las filas que aparecen en los resultados están más restringidas. El primer test (oficina_rep=oficina) selecciona solamente pares de filas REPVENTAS y OFICINAS que tienen la relación padre/hijo. El segundo test selecciona adicionalmente sólo aquellos pares de filas donde la oficina está por encima del objetivo.

3.6.3 Múltiples columnas de emparejamiento.

La tabla PEDIDOS y la tabla PRODUCTOS de la base de datos ejemplo están relacionadas por un par de claves foránea/primaria. Las columnas FAB y PRODUCTO de la tabla PEDIDOS forman juntas una clave foránea para la tabla PRODUCTOS, que se emparejan con las columnas ID_FAB e ID_PRODUCTO, respectivamente. Para componer las tablas basándose en la relación padre/hijo, deben especificarse ambos pares de columnas de emparejamiento. Por ejemplo:

```
SELECT num_pedido,importe,descripcion FROM pedidos,productos WHERE
fab=id_fab AND producto=id_producto
```

NUM_PEDIDO	*IMPORTE*	*DESCRIPCION*
112961	31.500	Bisagra Izqda.
113012	3.745	Articulo Tipo 3
112989	1.458	Bancada Motor
113051	1.420	Reductor
112968	3.978	Articulo Tipo 4
110036	22.500	Montador
113045	45.000	Bisagra Dcha.
112963	3.276	Articulo Tipo 4
113013	652	Manivela
113058	1.480	Cubierta
112997	652	Manivela
112983	702	Articulo Tipo 4
113024	7.100	Reductor
113062	2.430	Bancada Motor
112979	15.000	Montador
113027	4.104	Articulo Tipo 2
113007	2.825	Riostra 1/2-Tm

113069	31.350	Riostra 1-Tm
113034	632	V Stago Trinquete
112992	760	Articulo Tipo 2
112975	2.100	Pasador Bisagra
113055	150	Ajustador
113048	3.750	Riostra 2-Tm
112993	1.896	V Stago Trinquete
113065	2.130	Reductor
113003	5.625	Risotra 2-Tm
113049	776	Reductor
112987	27.500	Extractor
113057	600	Ajustador
113042	22.500	Bisagra Dcha.

Tabla 3.6.3.1: Lista todos los pedidos, mostrando los importes y las descripciones del producto.

Las composiciones multicolumna son menos habituales y se encuentran generalmente en consultas que afectan a claves foráneas compuestas.

3.6.4 Consultas de tres o más tablas.

SQL puede combinar datos de tres o más tablas utilizando las mismas técnicas básicas utilizadas para las consultas de dos tablas. Un sencillo ejemplo es el siguiente:

```
SELECT num_pedido,importe,empresa,nombre FROM pedidos,clientes,repventas
WHERE clie=num_clie AND rep=num_empl AND importe>25000
```

NUM_PEDIDO	*IMPORTE*	*EMPRESA*	*NOMBRE*
112961	31.500	Hnos. Ramon S.L.	Jose Maldonado
113045	45.000	Lopez Asociados S.L.	Lorenzo Fernandez
113069	31.350	Roda & Castedo S.L.	Natalia Martin
112987	27.500	Pino S.L.	Belen Aguirre

Tabla 3.6.4.1: Lista de los pedidos superiores a 25000, incluyendo el nombre del vendedor que tomo el pedido y el nombre del cliente que lo solicitó.

Esta consulta utiliza dos claves foráneas de la tabla PEDIDOS. La columna CLIE es una clave foránea para la tabla CLIENTES, que enlaza cada pedido con el cliente que lo remitió. La columna REP es una clave foránea para la tabla REPVENTAS, que enlaza cada pedido con el vendedor que lo aceptó. Otra consulta de tres tablas que utiliza una disposición diferente de las relaciones padre/hijo es:

```
SELECT num_pedido,importe,empresa,nombre FROM pedidos,clientes,repventas
WHERE clie=num_clie AND rep_clie=num_empl AND importe>25000
```

NUM_PEDIDO	*IMPORTE*	*EMPRESA*	*NOMBRE*
112961	31.500	Hnos. Ramon S.L.	Jose Maldonado
113045	45.000	Lopez Asociados S.L.	Lorenzo Fernandez
113069	31.350	Roda & Castedo S.L.	Pedro Cruz
112987	27.500	Pino S.L.	Belen Aguirre

Tabla 3.6.4.2: Lista de los pedidos superiores a 25000, incluyendo el nombre del cliente que remitió el pedido y el nombre del vendedor asignado a ese cliente.

No es infrecuente encontrar consultas de tres o más tablas en aplicaciones SQL de producción. Por ejemplo, dentro de la sencilla base de datos ejemplo que contiene cinco tablas, es posible hallar una consulta de cuatro tablas que tiene sentido:

```
SELECT num_pedido,importe,empresa,nombre,ciudad FROM
pedidos,clientes,repventas,oficinas WHERE clie=num_clie AND rep_clie=num_empl
AND oficina_rep=oficina AND importe>25000
```

NUM_PEDIDO	*IMPORTE*	*EMPRESA*	*NOMBRE*	*CIUDAD*
112961	31.500	Hnos. Ramon S.L.	Jose Maldonado	Valencia
113045	45.000	Lopez Asociados S.L.	Lorenzo Fernandez	Madrid
113069	31.350	Roda & Castedo S.L.	Pedro Cruz	Barcelona
112987	27.500	Pino S.L.	Belen Aguirre	Alicante

Tabla 3.6.4.3: Lista de los pedidos superiores a 25000, incluyendo el nombre del cliente que remitió el pedido, el vendedor asociado al cliente y la oficina donde trabaja.

3.6.5 Otras equicomposiciones.

La mayoría de las consultas multitabla se basan en relaciones padre/hijo, pero SQL no exige que las columnas de emparejamiento estén relacionadas como clave primaria y clave foránea. Cualquier par de columnas de dos tablas pueden servir de columnas de emparejamiento, siempre que tengan tipos de datos comparables. Veamos un ejemplo:

```
SELECT num_pedido,importe,fecha_pedido,nombre FROM pedidos,repventas
WHERE fecha_pedido=contrato
```

NUM_PEDIDO	*IMPORTE*	*FECHA_PEDIDO*	*NOMBRE*
112968	3.978	12/10/1999	Maria Garcia
112979	15.000	12/10/1999	Maria Garcia
112975	2.100	12/10/1999	Maria Garcia
112968	3.978	12/10/1999	Lorenzo Fernandez
112979	15.000	12/10/1999	Lorenzo Fernandez
112975	2.100	12/10/1999	Lorenzo Fernandez

Tabla 3.6.5.1: Pedidos recibidos en los días en que un vendedor fue contratado.

Los resultados de esta consulta provienen de los pares de filas de las tablas PEDIDOS y REPVENTAS donde el valor en la columna FECHA_PEDIDO coincide con el valor de la columna CONTRATO para el vendedor. Ninguna de estas columnas es una clave foránea o una clave primaria, pero SQL es capaz de componer las tablas.

3.6.6 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 las tablas. Aunque las composiciones basadas en la igualdad entre columnas correspondientes (equicomposiciones) son con mucho las composiciones más habituales, SQL también permite componer tablas basándose en otros operadores de comparación. Por ejemplo:

```
SELECT nombre,cuota,ciudad,objetivo FROM repventas,oficinas WHERE
oficina_rep=oficina and cuota>(0.5*objetivo)
```

NOMBRE	*CUOTA*	*CIUDAD*	*OBJETIVO*
Belen Aguirre	30.000	Alicante	30.000
Maria Garcia	27.500	Valencia	52.500
Natalia Martin	27.500	Toledo	27.500

Tabla 3.6.6.1: Lista los vendedores y oficinas donde la cuota del vendedor es superior al 50% del objetivo de la oficina.

3.6.7 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, contienen ambas una columna de nombre VENTAS. 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 dos ejemplos:

```
SELECT ciudad,ventas FROM oficinas WHERE ventas>objetivo
```

CIUDAD	*VENTAS*
Toledo	34.432
Alicante	39.327
Madrid	81.309

Tabla 3.6.7.1: Lista de las ciudades donde las ventas superan el objetivo.

```
SELECT nombre,ventas FROM repventas WHERE ventas>30000
```

NOMBRE	*VENTAS*
Jose Maldonado	32.958
Belen Aguirre	39.327
Lorenzo Fernandez	58.533
Natalia Martin	34.432

Tabla 3.6.7.2: Lista de los vendedores con ventas superiores a 30000.

Sin embargo, en la siguiente consulta los nombres duplicados provocan un problema:

```
SELECT nombre,ventas,ciudad FROM repventas,oficinas WHERE oficina_rep=oficina
```

Error: Columna 'ventas' se encuentra en mas de una tabla.

Para eliminar la ambigüedad, debe utilizarse un nombre de columna cualificado para identificar la columna. El nombre de columna cualificado se construye con el nombre de la tabla y el nombre de la columna. Por ejemplo, los nombres cualificados de las dos columnas VENTAS son OFICINAS.VENTAS y REPVENTAS.VENTAS, esto es, el nombre de la tabla seguido por el nombre de la columna.

Un nombre de columna cualificado puede ser utilizado en una sentencia SELECT en cualquier lugar donde se permite usar un nombre de columna. La tabla especificada en el nombre de columna cualificado debe, obviamente, corresponder a una de las tablas especificadas en la lista FROM. Veamos la versión corregida de la consulta anterior:

```
SELECT nombre,repventas.ventas,ciudad FROM repventas,oficinas WHERE
oficina_rep=oficina
```

NOMBRE	*VENTAS*	*CIUDAD*
Jose Maldonado	32.958	Valencia
Carlos Martinez	0	Barcelona
Belen Aguirre	39.327	Alicante
Maria Garcia	7.105	Valencia
Lorenzo Fernandez	58.533	Madrid
Soledad Martinez	22.776	Madrid
Daniel Gutierrez	26.628	Barcelona
Pedro Cruz	2.700	Barcelona
Natalia Martin	34.432	Toledo

Tabla 3.6.7.3: Lista del nombre, las ventas y la oficina de cada vendedor.

Utilizar nombres de columna cualificados en una consulta multitabla es siempre una buena medida. La desventaja es que hacen que el texto de la consulta sea mayor.

3.6.8 Selecciones de todas las columnas.

Como se vio con anterioridad, “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. Por ejemplo:

```
SELECT * FROM repventas,oficinas WHERE oficina_rep=oficina
```

NUM_EMPL	*NOMBRE*	*EDAD*	*OFICINA_REP*	*TITULO*	*CONTRATO*	*DIRECTOR*
106	Jose Maldonado	52	11	VP Ventas	14/06/1998	NULL
104	Carlos Martinez	33	12	Dir. Ventas	19/05/1997	106
105	Belen Aguirre	37	13	Dir. Ventas	12/02/1998	104
109	Maria Garcia	31	11	Rep. Ventas	12/10/1999	106
108	Lorenzo Fernandez	62	21	Dir. Ventas	12/10/1999	106
102	Soledad Martinez	48	21	Rep. Ventas	10/12/1995	108
101	Daniel Gutierrez	45	12	Rep. Ventas	20/10/1996	104
103	Pedro Cruz	29	12	Rep. Ventas	01/03/1997	104
107	Natalia Martin	49	22	Rep. Ventas	14/11/1998	108

CUOTA	*VENTAS*	*OFICINA*	*CIUDAD*	*REGION*	*DIR*	*OBJETIVO*	*VENTAS*
25.000	32.958	11	Valencia	Este	106	52.500	40.063
17.500	0	12	Barcelona	Este	104	70.000	29.328
30.000	39.327	13	Alicante	Este	105	30.000	39.327
27.500	7.105	11	Valencia	Este	106	52.500	40.063
30.000	58.533	21	Madrid	Centro	108	60.000	81.309
30.000	22.776	21	Madrid	Centro	108	60.000	81.309
27.500	26.628	12	Barcelona	Este	104	70.000	29.328
25.000	2.700	12	Barcelona	Este	104	70.000	29.328
27.500	34.432	22	Toledo	Centro	108	27.500	34.432

Tabla 3.6.8.1: Informe sobre todos los vendedores y las oficinas en que trabajan.

Obviamente, la forma SELECT * en una consulta resulta ser de menos práctica cuando existen dos o más tablas en la cláusula FROM.

3.6.9 Autocomposiciones.

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. Por ello, la columna DIRECTOR es una clave foránea para la propia tabla REPVENTAS.

Si se tratara de expresar esta consulta como cualquier otra consulta de dos tablas implicando una coincidencia clave foránea/clave primaria, aparecería tal como ésta:

```
SELECT nombre,nombre FROM repventas,repventas WHERE director=num_empl
```

Error: Referencia duplicada a la tabla 'repventas'.

También podría intentarse la pregunta eliminando la segunda referencia a la tabla REPVENTAS:

```
SELECT nombre,nombre FROM repventas WHERE director=num_empl
```

NOMBRE	***NOMBRE***

Tabla 3.6.9.1: Empleados que son sus propios directores.

Obteniendo una tabla vacía, pues la condición director=num_empl se aplica fila a fila, buscando empleados que sean sus propios directores, no existiendo tales filas.

Para resolver el problema, imaginemos que SQL tuviera dos copias idénticas de la tabla REPVENTAS, una llamada EMPS, que contuviera los empleados, y otra llamada DIRS, que contuviera los directores. Entonces la columna director de la tabla EMPS sería una clave foránea para la tabla DIRS y la consulta se realizaría como:

```
SELECT emps.nombre,dirs.nombre FROM emps, dirs WHERE
emps.director=dirs.num_empl
```

SQL realmente no tiene dos tablas diferentes, pero es capaz de asignar a una tabla un nombre diferente, llamado *alias de tabla*. La consulta, escrita utilizando los alias EMPS y DIRS para la tabla REPVENTAS sería:

```
SELECT emps.nombre,dirs.nombre FROM repventas emps,repventas dirs WHERE
emps.director=dirs.num_empl
```

NOMBRE	***NOMBRE***
Carlos Martinez	Jose Maldonado
Belen Aguirre	Carlos Martinez
Maria Garcia	Jose Maldonado
Lorenzo Fernandez	Jose Maldonado
Soledad Martinez	Lorenzo Fernandez
Daniel Gutierrez	Carlos Martinez
Antonio Valle	Daniel Gutierrez
Pedro Cruz	Carlos Martinez
Natalia Martin	Lorenzo Fernandez

Tabla 3.6.9.2: Lista de los nombres de los vendedores y sus directores.

La cláusula FROM asigna una alias a cada "copia" de la tabla REPVENTAS especificando el nombre del alias inmediatamente después del nombre real de la tabla. Cuando la cláusula FROM contiene un alias, el alias debe ser utilizado para identificar la tabla en referencias de columna cualificadas.

Naturalmente, en la pregunta anterior solo es realmente necesario utilizar un alias para una de las dos ocurrencias de la tabla de la consulta, por lo cual podíamos haber escrito la pregunta como:

```
SELECT repventas.nombre,dirs.nombre FROM repventas,repventas dirs WHERE
repventas.director=dirs.num_empl
```

Otro ejemplo de autocomposición puede verse a continuación:

```
SELECT repventas.nombre,repventas.cuota,dirs.cuota FROM repventas, repventas
dirs WHERE repventas.director=dirs.num_empl AND repventas.cuota>dirs.cuota
```

NOMBRE	*CUOTA*	*CUOTA*
Belen Aguirre	30.000	17.500
Maria Garcia	27.500	25.000
Lorenzo Fernandez	30.000	25.000
Daniel Gutierrez	27.500	17.500
Pedro Cruz	25.000	17.500

Tabla 3.6.9.3: Lista de los vendedores con una cuota superior a la de su director.

Los alias de tablas se requieren en consultas que afectan a autocomposiciones, pero pueden utilizarse alias en cualquier consulta. Así, por ejemplo la consulta:

```
SELECT nombre,repventas.ventas,ciudad FROM repventas,oficinas WHERE
oficina_rep=oficina
```

Puede escribirse como:

```
SELECT r.nombre,r.ventas,ciudad FROM repventas r,oficinas WHERE
r.oficina_rep=oficina
```

3.7 Consultas sumarias.

Muchas peticiones de información no requieren el nivel de detalle proporcionado por las consultas SQL. Por ejemplo, podemos solicitar de la base de datos:

- ¿Cuál es la cuota total para todos los vendedores?.
- ¿Cuáles son las cuotas mínima y máxima?.
- ¿Cuántos vendedores han superado su cuota?.
- ¿Cuál es el tamaño del pedido medio para cada oficina?.

SQL soporta estas peticiones 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 sumariza la columna. SQL ofrece seis funciones de columna diferentes:

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

3.7.1 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 o coma flotante). El resultado de la función SUM() tiene el mismo tipo de dato básico que los datos de la columna, pero el resultado puede tener una precisión superior. Veamos algunos ejemplos que utilizan la función de columna SUM():

```
SELECT SUM(cuota),SUM(ventas) FROM repventas
```

SUM(CUOTA)	*SUM(VENTAS)*
240.000	247.582

Tabla 3.7.1.1: Cuotas y ventas totales.

```
SELECT SUM(importe) FROM pedidos,repventas WHERE nombre='Belen Aguirre'
AND rep=num_empl
```

SUM(IMPORTE)
39.327

Tabla 3.7.1.2: Total de pedidos aceptados por el empleado Belen Aguirre.

3.7.2 Cálculo del promedio de una columna (AVG).

La función de columna AVG() calcula el promedio de una columna de valores de datos. Al igual que en la función SUM(), los datos de la columna deben tener tipo numérico. El resultado de la función AVG() puede tener un tipo de datos diferente al de los valores de la columna. Por ejemplo, si se aplica AVG() a una columna de enteros, el resultado será un número decimal o un número de coma flotante. Veamos algunos ejemplos de uso de la función de columna AVG():

```
SELECT AVG(precio) FROM productos WHERE id_fab='ACI'
```

AVG(PRECIO)
804,29

Tabla 3.7.2.1: Precio promedio de los productos del fabricante 'ACI'.

```
SELECT AVG(importe) FROM pedidos WHERE clie=2103
```

AVG(IMPORTE)
8.895,50

Tabla 3.7.2.2: Precio medio del pedido ordenado por el cliente 2103.

3.7.3 Determinación de los 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. Veamos algunos ejemplos:

```
SELECT MIN(cuota),MAX(cuota) FROM repventas
```

MIN(CUOTA)	*MAX(CUOTA)*
17.500	30.000

Tabla 3.7.3.1: Cuotas mínima y máxima asignadas a los vendedores.

```
SELECT MIN(fecha_pedido) FROM pedidos
```

MIN(FECHA_PEDIDO)
12/10/1999

Tabla 3.7.3.2: Fecha del pedido más antiguo en la base de datos.

```
SELECT MAX(100*ventas/cuota) FROM repventas
```

*MAX(100*VENTAS/CUOTA)*
195

Tabla 3.7.3.3: Mejor rendimiento de todos los vendedores.

```
SELECT MIN(nombre),MAX(nombre) FROM repventas
```

MIN(NOMBRE)	*MAX(NOMBRE)*
Antonio Valle	Soledad Martinez

Tabla 3.7.3.4: Primer y último vendedor por orden alfabético.

3.7.4 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() devuelve siempre un entero, independientemente del tipo de datos de la columna. Algunos ejemplos son:

```
SELECT COUNT(num_clie) FROM clientes
```

COUNT(NUM_CLIE)
21

Tabla 3.7.4.1: Número de clientes.

SELECT COUNT(nombre) FROM repventas WHERE ventas>cuota

COUNT(NOMBRE)
4

Tabla 3.7.4.2:Número de vendedores que superan su cuota.

SELECT COUNT(importe) FROM pedidos WHERE importe>25000

COUNT(IMPORTE)
4

Tabla 3.7.4.3: Número de pedidos de más de 25000.

Resaltar que la función COUNT() ignora los valores de los datos en la columna, simplemente cuenta cuántos datos hay. En consecuencia, no importa realmente qué columna se especifica como argumento de la función COUNT(). Por ello, el SQL soporta una función de columna especial, COUNT(*) que cuenta filas en lugar de valores de datos. Por ello la última consulta podría haberse escrito como:

SELECT COUNT(*) FROM pedidos WHERE importe>25000

COUNT(*)
4

Tabla 3.7.4.4: Número de pedidos de más de 25000.

3.7.5 Funciones de columna en la lista de selección.

Las consultas simples con una función de columna en una lista de selección son fáciles de entender. Sin embargo, cuando la lista de selección incluye varias funciones de columna es difícil entender su proceso.

Uno de los mejores modos de imaginar las consultas sumarias y las funciones de columna es pensar en el procesamiento de la consulta dividido en dos pasos. Primero, imaginamos cómo funcionaria la consulta sin las funciones de columna, produciendo muchas filas de resultados de consulta detallados. Luego imaginamos a SQL aplicando las funciones de columna a los resultados de consulta detallados, produciendo una sola fila sumaria. Por ejemplo:

SELECT importe,importe,100*importe/limite_credito,100*importe/cuota FROM pedidos,clientes,repventas WHERE clie=num_clie AND rep=num_empl

Y a continuación se aplican sobre los resultados las funciones de columna:

SELECT AVG(importe),SUM(importe),AVG(100*importe/limite_credito), AVG(100*importe/cuota) FROM pedidos,clientes,repventas WHERE clie=num_clie AND rep=num_empl

AVG(IMPORTE)	*SUM(IMPORTE)*	*AVG(100*IMPORTE/LIMITE_CREDITO)*	*AVG(100*IMPORTE/CUOTA)*
8.253,03	247.591	24.10	27,86

Tabla 3.7.5.1: Diversos porcentajes de la base de datos.

Una función columna puede aparecer en la lista de selección en cualquier lugar en el que puede aparecer un nombre de columna. Sin embargo, el argumento de una función de columna no puede contener a otra función de columna y también es ilegal mezclar funciones de columna y nombres de columnas ordinarios en una lista de selección, pues la consulta carece de sentido, ya que un nombre de columna genera una tabla de resultados con cierto número de filas y una función de columna genera una columna de una fila con los resultados sumarios.

3.7.6 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. Sin embargo, dicha columna puede contener valores NULL. En tal caso, los valores NULL son ignorados por las funciones de columna. Veamos un ejemplo:

```
SELECT COUNT(*),COUNT(ventas),COUNT(cuota) FROM repventas
```

COUNT(*)	*COUNT(VENTAS)*	*COUNT(CUOTA)*
10	10	9

Tabla 3.7.6.1: Ejemplo de funciones de columna con valores NULL.

La tabla REPVENTAS contiene diez filas, por lo que COUNT(*) devuelve diez. Las columnas VENTAS contienen diez valores no NULL, por lo que la función COUNT(ventas) devuelve también diez. Sin embargo, la columna CUOTA es NULL para un vendedor, por lo cual la función COUNT(cuota) ignora ese valor de NULL y devuelve el valor nueve.

La ignorancia de los valores NULL para las funciones MIN() y MAX() carece de importancia. Sin embargo para las funciones SUM() y AVG() puede producir sutiles problemas. Supongamos la siguiente consulta:

```
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)*
247.582	240.000	7.582	-15.541

Tabla 3.7.6.2: Uso de la función SUM con valores NULL.

Como se observa, la expresión SUM(ventas)-SUM(cuota) funciona correctamente, mientras que la expresión SUM(ventas-cuota) proporciona un resultado aparentemente incorrecto. Esto es así pues existe un valor NULL en la columna CUOTA. Entonces, SUM(ventas)-SUM(cuota) realiza la resta entre la suma de los diez valores de ventas y la suma de los nueve valores de cuota, mientras que SUM(ventas-cuota) realiza primero la resta, con lo cual solo aparecen nueve valores validos (pues la resta de un número menos NULL produce NULL), con lo cual solo se suman nueve restas.

3.7.7 Eliminación de filas duplicadas (DISTINCT).

Igual que en la selección de una lista, se puede pedir a SQL que elimine valores duplicados de una columna antes de aplicarle la 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. Veamos algunos ejemplos:

```
SELECT COUNT(DISTINCT titulo) FROM repventas
```

COUNT(DISTINCT TITULO)
3

Tabla 3.7.7.1: Número de títulos diferentes de los vendedores.

```
SELECT COUNT(DISTINCT oficina_rep) FROM repventas WHERE ventas>cuota
```

COUNT(DISTINCT OFICINA_REP)
4

Tabla 3.7.7.2: Número de oficinas con vendedores que superan sus cuotas.

La palabra clave DISTINCT puede usarse con las funciones de columna COUNT(), SUM() y AVG(). Cuando se usa la palabra clave DISTINCT el argumento de la función columna debe ser un nombre de columna único, no puede ser una expresión; además, la palabra clave DISTINCT sólo puede ser especificada una vez en una consulta.

3.7.8 Consultas agrupadas (cláusula GROUP BY).

Las consultas sumarias descritas hasta ahora son como los totales al final de un informe. Condensan todos los datos detallados del informe en una única fila sumaria de datos. Sin embargo, en algunos casos es útil obtener subtotales. La cláusula GROUP BY de la sentencia SELECT proporciona esta capacidad. Veamos un ejemplo:

```
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	7.865,40
106	16.479,00
107	11.477,33
108	8.361,86
109	3.552,50
110	11.566,00

Tabla 3.7.8.1: Tamaño medio de los pedidos para cada vendedor.

SQL lleva a cabo la consulta de la siguiente forma:

1. 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.

2. Por cada grupo, SQL calcula el valor medio de la columna IMPORTE para todas las filas del grupo y genera una única fila sumario de resultados.

Veamos algunos ejemplos adicionales de consultas agrupadas:

```
SELECT oficina_rep,MIN(cuota),MAX(cuota) FROM repventas GROUP BY oficina_rep
```

OFICINA_REP	*MIN(CUOTA)*	*MAX(CUOTA)*
NULL	NULL	NULL
11	25.000	27.500
12	17.500	27.500
13	30.000	30.000
21	30.000	30.000
22	27.500	27.500

Tabla 3.7.8.2: Rango de cuotas asignadas a cada oficina.

```
SELECT oficina_rep,COUNT(*) FROM repventas GROUP BY oficina_rep
```

OFICINA_REP	*COUNT(*)*
NULL	1
11	2
12	3
13	1
21	2
22	1

Tabla 3.7.8.3: Número de vendedores asignados a cada oficina.

```
SELECT COUNT(DISTINCT num_clie),rep_clie FROM clientes GROUP BY rep_clie
```

COUNT(DISTINCT NUM_CLIE)	*REP_CLIE*
3	101
4	102
3	103
1	104
2	105
2	106
1	107
2	108
2	109
1	110

Tabla 3.7.8.4: Número de clientes diferentes atendidos por cada vendedor.

SQL también puede agrupar resultados de consulta en base a contenidos de dos o más columnas. Por ejemplo:

```
SELECT rep,clie,SUM(importe) FROM pedidos GROUP BY rep,clie
```

REP	*CLIE*	*SUM(IMPORTE)*
101	2102	3.978
101	2108	150
101	2113	22.500
102	2106	4.026
102	2114	15.000

102	2120	3.750
103	2111	2.700
105	2103	35.582
105	2111	3.745
106	2101	1.458
106	2117	31.500
107	2109	31.350
107	2124	3.082
108	2112	47.825
108	2114	7.100
108	2118	3.608
109	2108	7.105
110	2107	23.132

Tabla 3.7.8.5: Pedidos totales por cliente y vendedor.

Una limitación de las consultas agrupadas es que SQL ignora información referente a claves primarias y foráneas cuando analiza la validez de una consulta agrupada. Por ello, la consulta:

```
SELECT num_empl,nombre,SUM(importe) FROM pedidos,repventas WHERE
rep=num_empl GROUP BY num_empl
```

Da como resultado:

Error: La columna 'nombre' no es una expresión GROUP BY

La consulta tiene perfecto sentido, ya que la agrupación por el número de empleado del vendedor es, en efecto, igual que la agrupación sobre el nombre del vendedor. No obstante, SQL requiere que las columnas de SELECT estén especificadas como columna de agrupación, por ello el problema se corrige sencillamente realizando la consulta de la forma siguiente:

```
SELECT num_empl,nombre,SUM(importe) FROM pedidos,repventas WHERE
rep=num_empl GROUP BY num_empl,nombre
```

NUM_EMPL	*NOMBRE*	*SUM(IMPORTE)*
101	Daniel Gutierrez	26.628
102	Soledad Martinez	22.776
103	Pedro Cruz	2.700
105	Belen Aguirre	39.327
106	Jose Maldonado	32.958
107	Natalia Martin	34.432
108	Lorenzo Fernandez	58.533
109	Maria Garcia	7.105
110	Antonio Valle	23.132

Tabla 3.7.8.6: Pedidos totales por cada vendedor.

3.7.9 Condiciones de búsqueda en 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 la condición de búsqueda. Por tanto, la cláusula HAVING especifica una condición de búsqueda por grupos. Veamos un ejemplo:

```
SELECT rep,AVG(importe) FROM pedidos GROUP BY rep HAVING
SUM(importe)>30000
```

REP	*AVG(IMPORTE)*
105	7.865,40
106	16.479,00
107	11.477,33
108	8.361,86

Tabla 3.7.9.1: Tamaño de pedido promedio por vendedor para vendedores con pedidos totales de más de 30000.

SQL efectúa la consulta de la siguiente forma:

1. La cláusula GROUP BY dispone los pedidos en grupos por vendedor.

2. La cláusula HAVING elimina entonces los grupos donde el total de pedidos no excede de 30000.

3. Finalmente la cláusula SELECT calcula el tamaño de pedido medio para cada uno de los grupos restantes y genera los resultados de la consulta.

Veamos otro ejemplo:

```
SELECT ciudad,SUM(cuota),SUM(repventas.ventas) FROM oficinas,repventas
WHERE oficina=oficina_rep GROUP BY ciudad HAVING COUNT(*)>=2
```

CIUDAD	*SUM(CUOTA)*	*SUM(REPVENTAS.VENTAS)*
Barcelona	70.000	29.328
Madrid	60.000	81.309
Valencia	52.500	40.063

Tabla 3.7.9.2: Cuota total y ventas totales para todos los vendedores de una oficina en oficinas con dos o más personas.

En este caso el proceso que realiza SQL es el siguiente:

1. Compone las tablas OFICINAS y REPVENTAS para hallar la ciudad donde trabaja cada vendedor.

2. Agrupa las filas resultantes por oficina.

3. Elimina los grupos con menos de dos filas.

4. Calcula la cuota total y las ventas totales para cada grupo.

Por último, veamos otro ejemplo que utiliza todas las cláusulas de la sentencia SELECT:

```
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	38	32
Ajustador	25	37	30
Bancada Motor	243	15	16
Bisagra Dcha.	4.500	12	15
Riostra 1-Tm	1.425	5	22

Tabla 3.7.9.3: Precio, existencias y cantidad total de los pedidos de cada producto para los cuales la cantidad total pedida es superior al 75% de las existencias.

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

1. Agrupa las filas resultantes por fabricante e id de producto.

2. Elimina los grupos en donde la cantidad pedida (el total de la columna CANT para todos los pedidos del grupo) es menos del 75% de las existencias.

3. Calcula la cantidad total pedida para cada grupo.

4. Genera una fila sumaria de resultados por cada grupo.

5. Ordena los resultados para que los productos con el mayor valor de existencias aparezcan en primer lugar.

Las columnas DESCRIPCION, PRECIO y EXISTENCIAS aparecen especificadas como columnas de agrupación en esta consulta únicamente porque aparecen en la lista de selección. Realmente no contribuyen en nada al proceso de agrupación, ya que ID_FAB e ID_PRODUCTO especifican completamente una única fila en la tabla PRODUCTOS, haciendo automáticamente que las otras tres columnas tengan un único valor por grupo.

3.8 Subconsultas.

La característica de subconsulta de SQL permite utilizar los resultados de una consulta como parte de otra. La capacidad de utilizar una consulta dentro de otra fue la razón original para la palabra “estructurada” en el nombre Lenguaje de Consultas Estructuradas (Structured Query Language-SQL). Esta característica juega un papel importante por tres razones:

1. Una sentencia SQL con una subconsulta es frecuentemente el modo más natural de expresar una consulta, ya que se asemeja más estrechamente a la descripción de la consulta en lenguaje natural.

2. 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”.

3. Hay algunas consultas que no pueden ser expresadas en el lenguaje SQL sin utilizar una subconsulta.

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 manejar las peticiones de consultas que se expresan en términos de los resultados de otras consultas. Por ejemplo, analicemos la petición: "Lista las oficinas en donde las ventas de la oficina son inferiores al 50% de la suma de las cuotas de los vendedores de la oficina". La petición solicita una lista de oficinas de la tabla OFICINAS, en donde el valor de la columna VENTAS satisface cierta condición. Parece razonable que la sentencia SELECT que expresa la consulta deba ser semejante a esta:

```
SELECT ciudad FROM oficinas WHERE ventas<???
```

El valor "???" necesita ser sustituido, y debería ser igual a "al 50% de la suma de las cuotas de los vendedores asignados a la oficina en cuestión". Sabemos que la menor de las cuotas para una oficina específica puede obtenerse como:

```
SELECT 0.5*SUM(cuota) FROM repventas WHERE oficina_rep=XX
```

Donde "XX" representa el número de oficina.

Parece entonces razonable comenzar con la primera consulta y reemplazar los "???" con la segunda consulta del modo siguiente:

```
SELECT ciudad FROM oficinas WHERE ventas<(SELECT 0.5*SUM(cuota) FROM
repventas WHERE oficina_rep=oficina)
```

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. El diagrama sintáctico de una subconsulta puede verse en la figura 3.8.1.

Como puede observarse en dicho diagrama sintáctico:

- Una subconsulta debe producir una única columna de datos como resultados. Esto significa que una subconsulta siempre tiene un único elemento de selección de la 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 carece de sentido ordenarlos.

- Una subconsulta no puede ser la UNION de varias sentencias SELECT diferentes; sólo se permite una única SELECT.

- Los nombre de columna que aparecen en una subconsulta pueden referirse a columnas de tablas en la consulta principal.

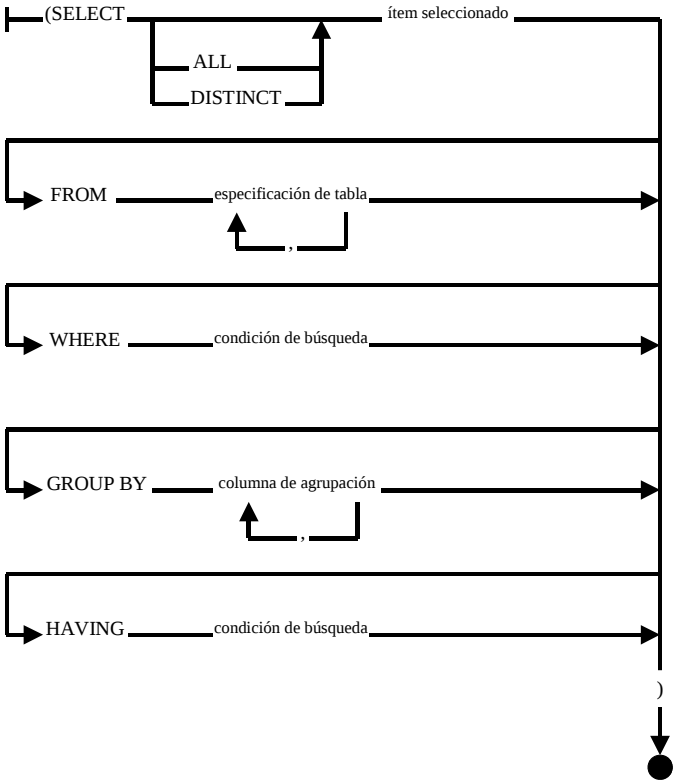


Figura 3.8.1: Diagrama sintáctico de una subconsulta.

3.8.1 Subconsultas en la cláusula 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. Por ejemplo, la consulta anterior:

```
SELECT ciudad FROM oficinas WHERE ventas<(SELECT 0.5*SUM(cuota) FROM
repventas WHERE oficina_rep=oficina)
```

CIUDAD
Barcelona

Tabla 3.8.1.1: Ciudades con ventas inferiores al 50% de la suma de las cuotas de los vendedores asignados a dicha ciudad.

Conceptualmente, SQL lleva a cabo la consulta de la siguiente forma: 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. Para ello 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 VENTAS de la fila actual con el valor producido por la subconsulta. Para examinar el valor VENTAS, 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 VENTAS, seleccionando o rechazando la oficina actual en base a la comparación.

Dentro del cuerpo de una subconsulta, con frecuencia es necesario referirse al valor de una columna en la fila “actual” de la consulta principal, tal y como sucede en la

consulta anterior. En estos casos, la columna de la consulta principal que se utiliza en la subconsulta se conoce como *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.

3.8.2 Condiciones de búsqueda en subconsultas.

Una subconsulta forma parte siempre de una condición de búsqueda en la cláusula WHERE o HAVING. Además de las condiciones de búsqueda simples que pueden ser utilizadas en estas cláusulas, SQL ofrece la siguientes condiciones de búsqueda en subconsultas:

- Test de comparación subconsulta. Compara el valor de una expresión con un valor único 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.

Test de comparación subconsulta.

El test de comparación subconsulta es una forma modificada del test de comparación simple. Compara el valor de una expresión con el valor 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 el ejemplo:

```
SELECT nombre FROM repventas WHERE cuota>=(SELECT objetivo FROM oficinas
WHERE ciudad='Alicante')
```

NOMBRE
Belen Aguirre
Lorenzo Fernandez
Soledad Martinez

Tabla 3.8.2.1: Lista los vendedores con cuota igual o superior al objetivo de Alicante.

El test de comparación subconsulta ofrece los mismos seis operadores de comparación (=, <>, <, <=, >, >=) disponibles en el test de comparación simple. Veamos algunos ejemplos adicionales:

```
SELECT empresa FROM clientes WHERE rep_clie=(SELECT num_empl FROM
repventas WHERE nombre='Belen Aguirre')
```

EMPRESA
Pino S.L.
JPF S.L.

Tabla 3.8.2.2: Lista de los clientes atendidos por Belen Aguirre.

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*
Articulo Tipo 1	277
Articulo Tipo 2	167
Articulo Tipo 3	207

Tabla 3.8.2.3: Lista de los productos del fabricante ACI para los cuales las existencias superan a las existencias del producto ACI-41004.

Test de pertenencia a conjunto.

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. Este test se utiliza cuando se necesita comparar un valor de la fila que está siendo examinada con un conjunto de valores producidos por una subconsulta, como se muestra en los ejemplos siguientes:

SELECT nombre FROM repventas WHERE oficina_rep IN (SELECT oficina FROM oficinas WHERE ventas>objetivo)

NOMBRE
Belen Aguirre
Lorenzo Fernandez
Soledad Martinez
Natalia Martin

Tabla 3.8.2.4: Lista de los vendedores que trabajan en oficinas que superan su objetivo de ventas.

SELECT nombre FROM repventas WHERE num_empl NOT IN (SELECT rep_clie FROM clientes WHERE limite_credito<50000)

NOMBRE
Lorenzo Fernandez

Tabla 3.8.2.5: Lista de empleados que tienen todos sus clientes con limite de credito igual o superior a 50000.

SELECT empresa FROM clientes WHERE num_clie IN (SELECT DISTINCT clie FROM pedidos WHERE fab='ACI' AND producto LIKE '4100%' AND fecha_pedido BETWEEN '01/01/2000' AND '30/06/2000')

EMPRESA
EVBE S.A.
Pino S.L.
Distribuciones Sur S.A.
Zapater Importaciones S.A.

Tabla 3.8.2.6: Lista de los clientes que han remitido pedidos del fabricante ACI con productos que empiezan por 4100 entre Enero y Junio de 2000.

Test de existencia.

El test de existencia (EXISTS) comprueba si una subconsulta produce alguna fila de resultados. No existe test de comparación simple que se asemeje al de existencia; solamente se utiliza con subconsultas. Veamos algunos ejemplos:

```
SELECT DISTINCT descripcion FROM productos WHERE EXISTS (SELECT
num_pedido FROM pedidos WHERE fab=id_fab AND producto=id_producto AND
importe>=25000)
```

DESCRIPCION
Bisagra Dcha.
Bisagra Izqda.
Extractor
Riostra 1-Tm

Tabla 3.8.2.7: Lista de productos para los cuales se ha recibido un pedido mayor o igual a 25000.

```
SELECT empresa FROM clientes WHERE rep_clie=(SELECT num_empl FROM
repventas WHERE nombre='Soledad Martinez') AND NOT EXISTS (SELECT * FROM
pedidos WHERE clie=num_clie AND importe>3000)
```

EMPRESA
Hnos. Martinez S.A.
Construcciones Leon S.A.

Tabla 3.8.2.8: Lista de los clientes asignados a 'Soledad Martinez' y que no han remitido ningún pedido superior a 3000.

```
SELECT ciudad FROM oficinas WHERE EXISTS (SELECT * FROM repventas
WHERE oficina_rep=oficina AND cuota>(0.55*objetivo))
```

CIUDAD
Toledo
Alicante

Tabla 3.8.2.9: Oficinas donde la cuota de un vendedor representa más del 55% del objetivo de la oficina.

Como podemos ver, el test EXISTS devuelve TRUE si la subconsulta produce filas o FALSE si no las produce. Como la condición de búsqueda EXISTS no utiliza realmente los resultados de la subconsulta, es posible relajar la regla de que "las subconsultas deben devolver una única columna de datos" y permitir utilizar la forma SELECT * en la subconsulta de un test EXISTS.

Test cuantificados.

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 otros dos test cuantificados, ANY y ALL, que extienden esta noción a otros operadores de comparación.

El test ANY se utiliza conjuntamente con uno de los seis operadores de comparación de SQL (=, <>, <, <=, >, >=) para comparar un único valor de test con una columna de valores producidos en una subconsulta. Para efectuar el test, SQL utiliza el operador de comparación especificado para comparar el valor de test con cada valor de datos en la columna, uno cada vez. Si alguna de las comparaciones individuales produce un resultado TRUE, el test ANY devuelve un resultado TRUE.

El test ALL se utiliza, al igual que el test ANY, 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 TRUE, el test ALL devuelve un resultado TRUE. Veamos unos ejemplos de test ANY y ALL:

```
SELECT nombre FROM repventas WHERE cuota<ANY (SELECT importe FROM
pedidos WHERE rep=num_empl)
```

NOMBRE
Jose Maldonado
Lorenzo Fernandez
Natalia Martin

Tabla 3.8.2.10: Lista de los vendedores que han aceptado un pedido que representa más de su cuota.

```
SELECT ciudad,objetivo FROM oficinas WHERE objetivo<ALL (SELECT ventas
FROM repventas WHERE oficina_rep=oficina)
```

CIUDAD	*OBJETIVO*
Toledo	27.500
Alicante	30.000

Tabla 3.8.2.11: Lista de las oficinas y sus objetivos en donde todos los vendedores tienen ventas que igualan o superan el objetivo de la oficina.

3.8.3 Subconsultas anidadas.

Todas las consultas descritas hasta ahora han sido consultas de “dos niveles”, afectando a la consulta principal y una subconsulta. Del mismo modo que se puede utilizar una subconsulta dentro de una consulta principal, se puede utilizar una subconsulta dentro de otra subconsulta. Esto se puede extender hasta el nivel de subconsulta que se desee. Veamos un ejemplo:

```
SELECT empresa FROM clientes WHERE rep_clie IN (SELECT num_empl FROM
repventas WHERE oficina_rep IN (SELECT oficina FROM oficinas WHERE
region='Centro'))
```

EMPRESA
Hnos. Martinez S.A.
Lopez Asociados S.L.
Componentes Fernandez S.A.
Domingo S.L.
Distribuciones Montiel S.L.

Construcciones Leon S.A.
Exclusivas Norte S.A.

Tabla 3.8.3.1: Lista de clientes cuyos vendedores están asignados a oficinas de la región de ventas Centro.

En este ejemplo, la subconsulta más interna produce una columna que contiene los números de oficina de las oficinas de la región Centro. La siguiente subconsulta produce una columna que contiene los números de empleado de los vendedores que trabajan en las oficinas seleccionadas. Finalmente la consulta externa encuentra los clientes cuyos vendedores tienen uno de los números de empleado seleccionados.

3.8.4 Subconsultas en la cláusula 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 la siguiente consulta:

```
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)
```

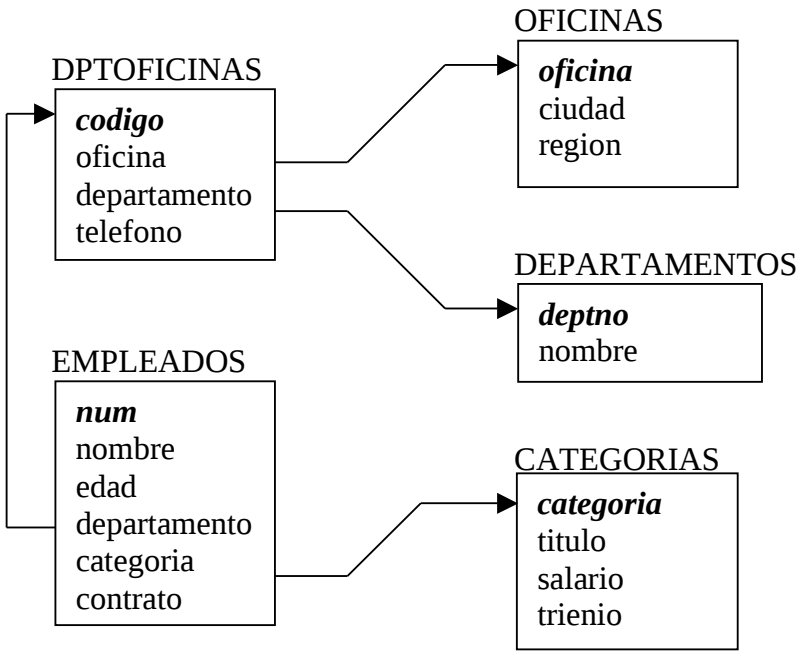
NOMBRE	*AVG(IMPORTE)*
Antonio Valle	22.500,00
Soledad Martinez	15.000,00

Tabla 3.8.4.1: Lista de los vendedores cuyo tamaño de pedido medio para productos fabricados por "ACI" es superior al tamaño de pedido medio global.

La consulta funciona calculando en primer lugar el "tamaño de pedido medio global". Luego la cláusula HAVING comprueba cada grupo de filas para ver si el tamaño medio pedido de ese grupo es superior al promedio de todos los pedidos. Si es así, el grupo de filas es retenido; si no, el grupo de filas es descartado. Finalmente la cláusula SELECT produce una fila sumaria por cada grupo, mostrando el nombre del vendedor y el tamaño de pedido medio para cada uno.

3.9 Ejercicios de consultas en SQL.

Dado el siguiente diagrama de una base de datos y las relaciones mostradas a continuación:



OFICINAS		
OFICINA	*CIUDAD*	*REGION*
11	Valencia	Este
12	Barcelona	Este
13	Alicante	Este
21	Madrid	Centro
22	Toledo	Centro

CATEGORÍAS			
CATEGORIA	TITULO	SALARIO	TRIENIO
1	Director	50.000	1.000
2	Jefe Sección	40.000	800
3	Administrativo	35.000	700
4	Comercial	35.000	700
5	Empl. almacén	25.000	500

DEPARTAMENTOS	
DEPTNO	*NOMBRE*
1	Dirección
2	Administración
3	Comercial
4	Almacén
5	Informática

DPTOFICINAS			
CODIGO	OFICINA	DEPARTAMENTO	TELEFONO
100	11	1	963.981.000
101	11	2	963.981.100
102	11	3	963.981.200
103	11	4	963.981.300
104	11	5	963.981.400
105	12	2	933.551.000
106	12	3	933.551.100
107	12	4	933.551.200
108	13	3	961.671.000
109	21	2	913.641.000
110	21	3	913.641.100
111	21	4	913.641.200
112	22	3	925.871.000

EMPLEADOS					
NUM	*NOMBRE*	*EDAD*	*DEPARTAMENTO*	*CATEGORÍA*	*CONTRATO*
1000	Antonio Gutierrez	45	100	1	12/01/1989
1001	Paloma Blanco	56	100	2	17/03/1992
1002	Antonio Pazos	52	100	3	14/03/1986
1003	Ana Garcia	27	101	2	23/10/1995
1004	Amparo Beltran	28	101	3	04/02/1998
1005	Enrique Gomez	36	101	3	05/07/2000
1006	Nieves Soler	25	105	2	13/03/1996
1007	Juan Jose Velasco	34	105	3	14/02/1997
1008	Isidro Perez	22	109	2	06/05/2000
1009	Ignacio Lopez	38	109	3	07/04/1990

1010	Vicente Salvador	29	109	3	08/07/1995
1011	Carmen Hernandez	44	102	2	16/07/1990
1012	Juan Pons	50	102	4	14/04/1994
1013	Pedro Fernandez	23	102	4	16/09/1999
1014	Silvia Blasco	33	102	4	23/02/1992
1015	Jose Alegre	26	106	2	26/08/1997
1016	Cristina Prats	46	106	4	18/11/1984
1017	Carlos Gimenez	35	106	4	15/05/1995
1018	Maria Gonzalez	37	108	4	16/06/1996
1019	Manuel Torres	24	108	4	19/01/1998
1020	Jose Perez	28	110	2	22/03/1996
1021	Alejandro Martos	34	110	4	17/10/1994
1022	Veronica Muelas	25	110	4	05/07/1997
1023	Elena Lopez	29	112	4	09/07/1994
1024	Isabel Fernandez	22	112	4	12/10/2000
1025	Jose Mujica	49	103	2	04/09/1987
1026	Pedro Bledos	26	103	5	06/02/1998
1027	Pablo Costas	35	107	5	03/07/1995
1028	Ester Castro	27	111	2	18/07/1996
1029	Gregorio Mas	33	111	5	14/03/1997
1030	Jose Medina	34	104	2	14/06/1995
1031	Maria Utrillas	27	104	3	19/08/1997
1032	Marina Gilabert	24	104	3	01/12/1998

Contestar a las siguientes preguntas, utilizando para ello el lenguaje SQL.

1) Nombre y edad de los empleados.

2) Salario y trienios de cada categoría si suponemos un aumento del 2%.

3) Año de contratación de cada empleado.

4) Edades de los empleados.

5) Categorías profesionales que superan las 35.000 de salario.

6) Datos del empleado número 1014.

7) Empleados del departamento 106.

8) Empleados cuya contratación se produjo en el año 2000.

9) Empleados que no sean comerciales (código de categoría 4).

10) Empleados contratados entre los años 1990 y 1994.

11) Categorías que tienen un salario inferior a 35.000 o superior a 40.000.

12) Empleados cuya categoría es director o jefe de sección (códigos 1 y 2).

13) Empleados de nombre 'Jose'.

14) Empleados que pertenecen a la categoría de administrativo (código 3) y que tienen una edad mayor de 35 años.

15) Empleados que no pertenecen al departamento 110..

16) Nombre y edad de los empleados ordenados por edad.

17) Nombre y edad de los empleados ordenados por nombre de forma descendente.

18) Nombre del empleado y de la categoría en el que trabaja.

19) Código y teléfonos de los departamentos de las oficinas de la región 'Centro'.

20) Nombre del empleado y ciudad en la que trabaja.

21) Sueldo de cada empleado incluyendo trienios.

22) Nombre de los empleados y de sus jefes de sección.

23) Suma del sueldo de los empleados, sin contar trienios.

24) Promedio del sueldo, sin contar trienios, de la oficina de 'Barcelona'.

25) Salarios máximo y mínimo de los empleados, incluyendo trienios.

26) Número de empleados que superan los 40 años.

27) Número de edades diferentes que tienen los empleados.

28) Categoría y suma de los sueldos de los empleados, contando trienios, de cada una de las categorías.

29) Nombre y suma de los sueldos de los empleados, contando trienios, de cada oficina.

30) Titulo y suma de trienios de las categorías cuya suma supera las 10000

31) Nombre del departamento y número de empleados de los departamentos que tienen más de 5 empleados.

32) Nombre y sueldo, contando trienios, de los empleados cuyos sueldos son inferiores a la media de sueldos de la empresa.

33) Título de las categorías donde existe un empleado con contrato anterior a 1990.

34) Nombre de los empleados que tiene un contrato más antiguo que cualquier empleado del departamento de 'Informática'.

35) Ciudad y número de empleados de la oficina que tiene un número de empleados superior a la media de la empresa.