TEMA IV
EL COMANDO SELECT

Introducción

Tal vez la sentencia de mayor repercusión entre la comunidad de usuarios y desarrolladores, es la *SELECT* a través de la cual se realizan las consultas a los datos mantenidos en la base. La sentencia *SELECT* consta de varias cláusulas que utilizadas en conjunto, permiten obtener complicadas salidas a partir de simples instrucciones. En el tema se desarrollarán las cláusulas y sus formas de utilización.

LAS CLÁUSULAS

Las expresiones que soporta el SQL indicándose además su finalidad son:

Cláusula	Finalidad
FROM	Nombres de las tablas de las que se seleccionan filas
WHERE	Especifica las condiciones que deben satisfacer las filas seleccionadas
GROUP BY	Agrupa las filas seleccionadas en grupos especificados
HAVING	Indica la condición que ha de satisfacer cada grupo mostrado en el **GROUP BY**
ORDER BY	Especifica el orden en que se mostrarán las filas seleccionadas

Estas cláusulas, son expresiones de las tablas que se utilizan para obtener tablas. Los predicados que utilizan en SQL para indicar una condición específica son los ya enumerados de comparación (=, <, >, <=, >=,<> ó !=), **BETWEEN**, **IN**, **LIKE**, **NULL**, los cuantificadores (**ALL**, **SOME**, **ANY**) y **EXISTS**.

Las expresiones de las tablas se utilizan para especificar una tabla o una tabla agrupada. Estas son cláusulas que se usan para obtener tablas que son el resultado de la última cláusula especificada. Las expresiones de las tablas son las indicadas anteriormente, y excepto la cláusula **FROM**, las demás son todas optativas. Las columnas especificadas luego de la palabra SELECT se denominará *select_list*.

La cláusula FROM

La cláusula **FROM**, especifica una o varias tablas a partir de las cuales se recuperarán las filas que se desean. Debe recordarse que una consulta SQL devuelve siempre una tabla. Si no hay expresiones optativas en la consulta (**where, group by, having** u **order by**), entonces la tabla que se ha recuperado es la tabla compuesta por las columnas de la lista objeto solamente. Si la

tabla indicada en la cláusula **FROM** es una vista agrupada, entonces la consulta no contiene ninguna expresión optativa de tabla, tal como **where**, **group by** o **having**. Si la lista objeto contiene columnas de más de una tabla, entonces la cláusula **FROM** deberá nombrar todas las tablas en cualquier orden, independientemente del orden de las columnas en la lista objeto.

La cláusula WHERE

La cláusula **WHERE** especifica una tabla obtenida por la aplicación de una condición de búsqueda a las tablas que se listan en la cláusula **FROM**. En otras palabras, el resultado de la cláusula **WHERE** es esa o esas filas recuperadas de las tablas nombradas en la cláusula **FROM** que satisfacen la especificación de la cláusula **WHERE**. La sintaxis es:

```
SELECT columna1, columna2, ....., columnaN
    FROM nombre_de_la_tabla
    WHERE condición;
```

Esta cláusula puede contener una o más subconsultas. Si es así, cada subconsulta que sigue a la cláusula **WHERE** se ejecuta para cada fila recuperada por la cláusula **FROM**.

La Cláusula GROUP BY

La cláusula **GROUP BY** especifica una tabla agrupada que resulta de la aplicación de la cláusula **GROUP BY** al resultado de cualquier cláusula especificada previamente. Esta cláusula, hace referencia de manera específica a una columna o varias columnas de la tabla nombrada en la cláusula **FROM** y agrupa las filas sobre la base de los valores de esas columnas. El resultado de una cláusula **GROUP BY** divide el resultado de la cláusula **FROM** en un conjunto de grupos, de forma que para cada grupo de más de una fila, los valores de la columna agrupada son idénticos. La sintaxis es:

```
SELECT columna1, columna2, ....., columnaN
    FROM      nombre_de_la_tabla
    GROUP BY columna_de_agrupación1, …
              Columna_de_agrupacionN
```

TODAS LA COLUMNAS de la select_list deben estar en GROUP BY. Si el comando no contiene una cláusula **WHERE**, entonces la cláusula **GROUP BY** se coloca inmediatamente detrás de la cláusula **FROM**. Si el comando tiene la cláusula **WHERE**, entonces la cláusula **GROUP BY** va detrás de aquella. Las filas devueltas, estarán ordenadas al azar dentro de cada grupo, porque esta cláusula no realiza ningún tipo de ordenamiento.

La cláusula HAVING

La cláusula **HAVING** especifica una restricción en la tabla agrupada que resulta de la cláusula anterior **GROUP BY** y elimina los grupos que no satisfagan la condición especificada. Si se especifica **HAVING** en una consulta, entonces también se tendrá que haber especificado

GROUP BY. **HAVING** se utiliza para especificar la cualidad que debe poseer un grupo para que éste sea devuelto. Compara una propiedad del grupo con un valor constante. Realiza la misma función con los grupos que **WHERE** realiza con las filas individuales. En otras palabras, elimina los grupos que no poseen la cualidad, de la misma manera que **WHERE** elimina las filas que no poseen esa cualidad. Por lo tanto, **HAVING** se utiliza siempre con **GROUP BY**. La sintaxis es la siguiente:

```
SELECT columna1, columna2, ....., columnaN
    FROM nombre_de_la_tabla
    GROUP BY columna/s_de_agrupación
    HAVING propiedad_especificada_del_grupo;
```

Por ejemplo, suponga la tabla **empleados** con la siguiente estructura: **(documento, nombre_empleado, nombre_puesto, sueldo)**, y se desea encontrar el sueldo anual medio de todos los puestos de trabajo en que hay más de un empleado, entonces la consulta será:

```
SELECT nombre_puesto, COUNT(*), 12*AVG(sueldo)  FROM  empleados
    GROUP BY  nombre_puesto
    HAVING  COUNT(*) > 1;
```

La cláusula ORDER BY

La cláusula **ORDER BY**, permite especificar el orden en que aparecerán las filas en la recuperación. Mientras que **GROUP BY** simplemente coloca juntas todas las filas que tienen el mismo valor en una columna especificada, **ORDER BY** hace una lista de las filas que hay en un grupo especificado de acuerdo con el valor creciente o decreciente. Si se usa la cláusula **ORDER BY**, ésta deberá ser la última cláusula del comando **SELECT**. Puede especificarse el orden ascendente (**ASC**) o descendente (**DESC**), teniendo en cuenta que el ascendente es el orden por defecto. Por consiguiente, sólo es necesario especificar la dirección si el orden que se desea es DESC.

Si la columna ordenada consta de letras en vez de números, SQL utilizará el orden alfabético ascendente (comenzando con la A) si no se especifica DESC.

LOS PREDICADOS

Los predicados son condiciones que se indican en la cláusula **WHERE** de una consulta SQL. Los predicados que permite SQL son la **comparación, BETWEEN, IN, LIKE, NULL, los cuantificadores y los existenciales**.

El predicado de Comparación

Un predicado de comparación especifica la comparación de dos valores. Consta de una expresión de valor, seguida de un operador de comparación, seguido, a su vez, ya sea de otra expresión de valor o de una subconsulta de valor único. Los tipos de datos de las dos expresiones

de valores, o la expresión de valor y la subconsulta, deben ser comparables. Los operadores de comparación incluidos en SQL son =, <> ó !=,<, >, <=, >=. Si los valores que hay a ambos lados del operador de la comparación no son NULL, entonces el predicado de la comparación es verdadero o falso. Si alguna de las dos expresiones de valores es un valor NULL, o si la subconsulta está vacía, entonces el resultado del predicado de la comparación es desconocido; sin embargo, cuando se usa **GROUP BY, ORDER BY** o **DISTINCT** junto con un predicado de comparación, un valor NULL es idéntico a otro valor NULL o es un duplicado del mismo.

Las cadenas de caracteres se pueden comparar por medio de los operadores de comparación mencionados antes. Esto se consigue comparando los caracteres que se encuentran en las mismas posiciones ordinales de la cadena. Así, dos cadenas de caracteres son iguales si todos los caracteres con la misma posición ordinal, son iguales.

El predicado BETWEEN

El predicado **BETWEEN** especifica la comparación dentro de un intervalo. La sintaxis es:

. . . BETWEEN . . . AND . . . o
. . . NOT BETWEEN . . . AND . . .

donde cada grupo de puntos suspensivos contienen un valor. Los tipos de datos de los valores, deben ser comparables. Para seleccionar los productos cuyos precios estén entre $ 5 y $ 10 de la tabla **producto**, se escribirá:

```
SELECT Cod_prod, Nom_prod, Precio
    FROM producto
    WHERE precio BETWEEN 5 AND 10;
```

La respuesta que se obtendrá, viene dada en cualquier orden, puesto que no se ha especificado el requerimiento del mismo. Si se efectúa nuevamente la misma consulta, puede ser que el orden, sea totalmente distinto al obtenido en el primer intento.

El término **NOT** se puede utilizar con **BETWEEN** para recuperar la información que hay fuera de un intervalo en lugar de la que hay dentro del mismo. Por ejemplo, para recuperar la información de los productos cuyo precio es inferior a $ 3 y superior a $ 6, será:

```
SELECT Cod_prod, Nom_prod, Precio
    FROM producto
    WHERE precio NOT BETWEEN 3 AND 6;
```

El predicado IN (o NOT IN)

El predicado **IN** (o **NOT IN**) - equivalente al uso de **OR** -, especifica una comparación cuantificada. Hace una lista de un conjunto de valores y prueba si un valor está en esa lista. La lista debe ir entre paréntesis. Por ejemplo, para recuperar los productos cuyo precio es alguno de

los siguientes: $ 4, $ 5, $ 7, será:

SELECT Cod_prod, Nom_prod, Precio FROM producto WHERE Precio IN (4,5,7);

Es de destacar que el mismo efecto se podría haber logrado utilizando **OR**.

SELECT Cod_prod, Nom_prod, Precio FROM producto
WHERE Precio = 4 OR Precio = 5 OR Precio = 7;

La consulta también se puede escribir usando **ANY**:

SELECT Cod_prod, Nom_prod, Precio FROM producto
WHERE precio = ANY(4,5,7);

El orden en que los elementos de la lista se especifican (select_list) en la consulta determina el orden en que aparecerán las columnas en la recuperación. *No determina el orden en que aparecerán las filas*. En caso de desear un ordenamiento específico, debe emplearse a continuación del **WHERE** la cláusula **ORDER BY**. Como en el caso anterior, también es válido el uso de **NOT IN**.

El predicado LIKE (o NOT LIKE)

El predicado **LIKE** especifica una comparación de caracteres pudiendo utilizarse *substrings*, utilizándose para ello el *guión bajo* ("_") para representar un único carácter y el signo de porcentaje ("%") para representar una cadena de caracteres de longitud arbitraria (incluyendo cero caracteres). Estos símbolos reciben el nombre de comodines (*wild cards*). Otro elemento más que se utiliza ya propio del motor en cuestión, es el símbolo arroba ("@") usado como carácter de escape para poder utilizar símbolos especiales (caso de "_" o "%" o ").

El predicado NULL

El predicado **NULL**, especifica la prueba que se lleva a cabo con un valor **NULL**. Para consultar la base de datos en busca de un valor **NULL** se necesita un enfoque ligeramente distinto a la consulta en busca de cualquier otro valor, debido a las propiedades especiales que caracterizan al concepto de **NULL**. Por ejemplo, si se busca el nombre de un producto, en el que debido a la falta de información se ha puesto **NULL** al precio, no se puede especificar:

WHERE precio = NULL

porque nada, ni incluso el mismo **NULL**, es igual al valor de **NULL**. **SQL** tampoco permite utilizar **NULL** en la cláusula **SELECT**.

No se puede encontrar el valor **NULL** por exclusión, como por ejemplo, estipulando que el precio está por encima o por debajo de cualquier precio conocido de la lista. Por ejemplo, si $ 10 es el mayor precio especificado de la lista, la consulta:

SELECT Nom_prod, Precio FROM producto WHERE precio > 10;

no devolverá el nombre del producto con el valor **NULL**, ni tampoco lo devolvería con el predicado:

SELECT Nom_prod, Precio FROM producto WHERE precio < 0

El único predicado que se puede utilizar cuando se busca un valor **NULL** es:

WHERE especificación_de_columna IS NULL (o **IS NOT NULL**)

Los predicados cuantificadores: ALL, SOME y ANY (todos, algún, cualquier)

Los predicados cuantificadores **ALL, SOME y ANY** exigen que se use el predicado de la comparación aplicado a los resultados de una *subconsulta*. Un predicado de comparación es un predicado que contiene un operador de comparación tal como =, <> ó !=, <, >, >=, <=, según se explicara anteriormente.

Estos predicados, permiten probar un valor único frente a todos los elementos de un conjunto. Por ejemplo, podría desearse encontrar los proveedores que no pertenecen a la ciudad de Santa Fe y cuya fecha de inicio de actividades sea inferior a la de todos los proveedores que son de la ciudad de Santa Fe. Para hacerlo se escribirá:

```
SELECT Nom_provee FROM proveedor
    WHERE Ciudad <> "Santa Fe" AND fecha_inicio < ALL
            (SELECT  fecha_inicio  FROM  proveedor
                    WHERE Ciudad = "Santa Fe");
```

De manera análoga, todos los otros operadores de comparación se pueden combinar con **SOME, ANY y ALL**. Cuando el operador de comparación que se está usando es igual, el término **ANY** se puede intercambiar con **IN**, e incluso algunas veces puede parecer más lógico usar el **IN**.

En muchos casos, **ANY** tiene el mismo significado que **SOME**. Si se desea ahora encontrar los proveedores que no pertenecen a la ciudad de Santa Fe y cuya fecha de inicio de actividades sea inferior a la de algún proveedor de la ciudad de Santa Fe, se puede escribir de la siguiente forma:

```
SELECT Nom_provee  FROM proveedor
    WHERE  Ciudad <> "Santa Fe" AND fecha_inicio < ANY
            (SELECT fecha_inicio FROM    proveedor
                WHERE Ciudad = "Santa Fe");
```

La comparación **< ANY** de la cláusula **WHERE** del **SELECT** exterior es verdadera, si el precio es menor que al menos un elemento del conjunto formado por todos los proveedores de Santa Fe.

El predicado EXISTS

El predicado **EXISTS** indica las condiciones de un conjunto vacío. Contiene una subconsulta que, junto con la estipulación de **EXISTS**, se puede evaluar como verdadera o falsa. Si el resultado de la subconsulta no existe, entonces el conjunto descrito por la subconsulta estará vacío. De ello resulta evidente que el predicado **EXISTS** representa el *cuantificador existencial de la lógica formal*. El cuantificador universal de la lógica formal, **FORALL** (para todo), no está incluido directamente en el lenguaje SQL; sin embargo se verá que un predicado equivalente que utiliza **EXISTS** proporciona el mismo resultado.

El predicado **EXISTS** se puede usar siempre que se pueda usar una consulta con el predicado **IN** aunque no siempre se puede usar **IN** en lugar de **EXISTS**. Puesto que las consultas que utilizan **EXISTS** son a veces más fáciles de formular que las consultas que usan **IN**, el uso de **EXISTS** adquiere una importancia adicional en SQL. La sintaxis del predicado **EXISTS** es la siguiente:

```
SELECT nombre_de_la_columna
    FROM nombre_de_la_tabla
    WHERE EXISTS(subconsulta);
```

Una consulta se puede realizar usando **NOT EXISTS** con la misma sintaxis.

LAS FUNCIONES AGREGADAS

Las funciones agregadas de SQL no forman parte del álgebra relacional, por lo que se considera que en este sentido, SQL es más potente que el álgebra relacional.

SQL soporta las funciones agregadas o funciones grupo siguientes:

AVERAGE (AVG):
Calcula el valor promedio de los valores especificados de una columna. El argumento puede ir precedido de **DISTINCT**, con el fin de eliminar los valores duplicados. Los valores **NULL** no se consideran en la suma.

COUNT:
Devuelve el número de valores de la columna; el tipo de datos es numérico con la precisión definida por el usuario. Puede especificarse **DISTINCT**.

COUNT(*):
Es igual que **COUNT**, sólo que no se permite **DISTINCT**. Cuenta todas las filas considerando además las duplicadas.

MINIMUM (MIN):
Devuelve el valor menor de una columna.

MAXIMUM (MAX):

Devuelve el valor mayor de una columna.

SUM:

Devuelve la suma de los valores de una columna. La columna debe contener valores numéricos. El argumento puede ir precedido de **DISTINCT** para eliminar valores duplicados. Los **NULL** no se consideran en la suma.

Estas funciones agregadas, se distinguen de otros términos tales como **GROUP BY**, por el hecho que cada una de ellas, devuelve un único valor del grupo de filas con las que se opera, deducidos de aplicar tal función, al grupo de valores especificados en un argumento. En algunos sistemas comerciales, las funciones agregadas se las conoce como funciones de grupo o funciones incorporadas, mientras que las normas ANSI las llama funciones de conjunto, y algunos autores las llaman funciones de columna. La función agregada va en la instrucción **SELECT** de una consulta y seguida de la columna a la que se aplica. Para evitar confusiones, se incluye entre paréntesis el nombre de la columna. Se puede utilizar más de una función agregada en la misma instrucción **SELECT**. La sintaxis para utilizar funciones agregadas se presenta a continuación:

```
SELECT    funcion_agregada1  (nombre_de_la_columna), ....
          funcion_agregadaN  (nombre_de_la_columna)
    FROM            tabla(s)
    WHERE     condición(es);
```

LOS ALIAS

Aun cuando los alias tienen muchos usos potenciales, los usos principales de sustituir nombres de las tablas y las columnas son los siguientes:

1. Hacer el nombre cifrado de una columna más significativo cuando se la muestra
2. Abreviar un nombre que se usa a menudo en una tabla o de una columna
3. Hacer más clara una instrucción complicada de SQL
4. Distinguir entre dos ocurrencias del mismo nombre de columna o nombre de tabla, en cualquier instrucción SELECT

El uso indicado antes en cuarto lugar, puede implicar la unión natural de una tabla consigo misma, por lo que este punto será considerado posteriormente.

Los alias de columna

Para crear el alias de una columna, se introduce el alias después del nombre de la columna en la instrucción SELECT. Por ejemplo, para hacer que la columna AVG(sueldo), una vez calculada, sea más significativa cuando se la ve, se escribirá:

```
SELECT nom_funcion, AVG(sueldo) “Sueldo Medio” FROM  empleado
    GROUP BY nom_funcion
    HAVING  AVG(sueldo) >
```

```
(SELECT AVG(sueldo) FROM empleado
        WHERE nom_funcion = "secretaria");
```

La consulta anterior, dará una lista del promedio de sueldo de todas las funciones de empleados para los cuales, esa media es mayor que la media del sueldo de los empleados con el puesto de secretaria, y la cabecera de la columna será "Sueldo Medio", en vez de AVG(sueldo). Obsérvese que sólo se hace referencia al alias de la columna en la instrucción **SELECT**. Se le incluye entre comillas porque consta de dos palabras que habrán de aparecer juntas sobre los resultados que se muestren. Si el alias de la columna fuese una única palabra, no serían necesarias las comillas.

Los alias de tabla

Para crear el alias de una tabla, se la define en la cláusula **FROM**. Después se usa el alias como un calificador tanto en la cláusula **SELECT** como en la **WHERE**. Por ejemplo, si se desea abreviar con E el nombre de la tabla de Empleados, y C a la tabla de Clientes con el fin de combinar ciertos empleados con ciertos clientes, se escribirá:

```
SELECT E.*, C.*
        FROM   empleado E, cliente C
        WHERE E.id_zona = C.id_zona AND E.comision  >  0.15;
```

El uso del calificativo **E.** en la cláusula **AND** no es necesario si la tabla de empleados es la única que contiene la columna *comision*; sin embargo, siempre es bueno utilizar la especificación completa por una cuestión de claridad de la expresión. En el ejemplo anterior, se obtienen todos los campos de las tablas empleado y cliente recuperando los datos de aquellos clientes que viven en la misma zona asignada a los empleados (E.id_zona = C.id_zona) pero considerando solamente las de aquellos empleados que obtengan una comisión por ventas mayor a un 15% (E.comision > 0.15).

LAS SUBCONSULTAS

SQL permite el uso de subconsultas. En algunos sistemas de base de datos, a las subconsultas se las llama "selecciones anidadas". Estas se pueden usar para obtener la información que se necesita para completar la consulta principal. El uso de una subconsulta produce la escritura de una consulta compuesta en vez de dos o más consultas sencillas, y por lo tanto, proporciona un método para aumentar la eficacia del usuario.

En el procesamiento de una instrucción SQL compuesta, se evalúa primero la subconsulta, y a continuación, se aplican los resultados a la consulta principal. La mayor potencia de SQL se pueden lograr con el uso subconsultas. SQL no impone límites al número de subconsultas que se pueden anidar dentro de una consulta, aun cuando el implementador puede imponer un límite.

Si se sabe que la subconsulta deberá devolver como máximo un valor, o si se quiere estar seguro de que el resultado es único, entonces la sintaxis es:

```
SELECT      columnas
FROM        tablas
WHERE       condición (predicado de la comparación) (subconsulta);
```

La sintaxis anterior, devolverá un mensaje de error si hay más de una fila que cumpla tal condición.

Por otra parte, si la subconsulta puede devolver (o devolverá) más de un valor, entonces la sintaxis exige el predicado **IN** en la forma siguiente:

```
SELECT      columna
FROM        tablas
WHERE       nombre_de_la_columna    IN    (subconsulta);
```

A medida que quien escriba consultas SQL vaya adquiriendo habilidad, llegará hasta puntos en los que resolverá consultas muy complejas en un solo comando ayudándose con múltiples consultas anidadas. Esto resulta muy elegante y demostrará las grandes capacidades que tiene el SQL, pero, para el motor de la base de datos el gasto o consumo de recursos requeridos será muy grande. Se aconseja en consecuencia, y sobre todo en un ambiente de desarrollo en el que las instrucciones SQL estarán embebidas en los programas de aplicación y previendo que serán muchos los usuarios que los estarán utilizando simultáneamente, utilizar las subconsultas complejas solamente en los casos en los que sea imprescindible, para evitar de esta manera, una gran pérdida de *performance* en la respuesta del motor de base de datos.

Subconsultas correlacionadas

Sea la siguiente tabla con su estructura: **item** (nro_fac, cod_prod, precio_item). Se desea obtener un listado que contenga el número de factura y el precio total de la misma, pero solamente de aquellas en las que el precio total sea al menos el doble del precio del producto más barato dentro de la misma factura.

```
SELECT nro_fac, SUM(precio_item)
    FROM item A
    GROUP BY nro_fac
    HAVING SUM(precio_item) >
        (SELECT 2 * MIN(precio_item)
            FROM item
            WHERE nro_fac = A.nro_fac)
```

En el ejemplo mostrado, la tabla **item** está ligada consigo misma a través de una subconsulta correlacionada. Una subconsulta que contiene una referencia a una consulta externa es llamada subconsulta correlacionada (*correlated subquery*) porque su resultado está correlacionado

con cada fila individual de la consulta principal. Una subconsulta está correlacionada cuando sus valores dependen del valor de una variable recibida desde el SELECT externo.

LAS UNIONES (JOIN)

La capacidad de formar uniones, es probablemente una de las características más potentes del SQL. También es una de las propiedades más importantes que distinguen a una base de datos relacional de otros tipos de arquitectura. El formar uniones, es por consiguiente, un aspecto muy desarrollado del lenguaje SQL.

Una unión en SQL es una consulta en la que los datos se recuperan de dos o más tablas. La finalidad de unir, es recuperar información que no está en una única tabla. A continuación se estudiarán las uniones equidistantes, las uniones NO equidistantes, las uniones naturales, las uniones externas, las uniones de más de dos tablas y la unión de una tabla consigo misma.

Las uniones equidistantes

La unión equidistante en SQL se representa por:

```
SELECT  tabla1.*, tabla2.*
        FROM  tabla1, tabla2  WHERE (conjunto de condiciones);
```

donde el conjunto de condiciones es el grupo de ***comparaciones de igualdad*** entre las columnas de la tabla1 y las columnas de la tabla2. Las columnas comparadas, deben ser del mismo tipo y tamaño.

Si el conjunto de condiciones de la unión equidistante está vacío, vale decir que no exista cláusula **WHERE**, entonces el resultado, es el producto cartesiano de las tablas. En otras palabras, si se yuxtaponen cada fila de la tabla1 con cada fila de la tabla2 (sin eliminar ninguna fila de ninguna tabla), entonces el resultado es el producto cartesiano de la tabla1 por la tabla2. No es común que se ocupe una consulta de éste tipo por el gran costo que requiere. El comando para hacer esto es:

```
SELECT  *  FROM  tabla1, tabla2;
```

Por ejemplo, si se considera que la tabla1 contiene las columnas A, B y C, y la tabla2, que contiene las columnas C, D y E siguientes:

Tabla1			Tabla2		
A	**B**	**C**	**C**	**D**	**E**
1	1	1	1	4	5
1	2	1	2	6	7
2	2	1	3	8	9
1	2	2			

El producto cartesiano de las tablas 1 y 2 (que es la unión equidistante de las tablas 1 y 2), es la tabla3 siguiente, que contiene las columnas A, B, C, C, D y E:

TABLA3					
T1A	**T1B**	**T1C**	**T2C**	**T2D**	**T2E**
1	1	1	1	4	5
1	1	1	2	6	7
1	1	1	3	8	9
1	2	1	1	4	5
1	2	1	2	6	7
1	2	1	3	8	9
2	2	1	1	4	5
2	2	1	2	6	7
2	2	1	3	8	9
1	2	2	1	4	5
1	2	2	2	6	7
1	2	2	3	8	9

Si las dos tablas que se unen contienen dos o más nombres de columnas que son iguales, a esas se les llama columnas comunes. Las columnas T1C y T2C del ejemplo anterior, son un caso de ellas.

Uniones equidistantes con una condición

Si el conjunto de condiciones consta sólo de una comparación, entonces se tiene:

SELECT tabla1.*, tabla2.* FROM tabla1, tabla2 WHERE (condición);

en donde el operador de comparación de la condición es el signo igual (=).

Lo normal, es que se desee obtener una unión equidistante, cuando existen dos tablas que tienen una columna en común, y el conjunto de condiciones de la cláusula **WHERE** indicará que los valores de las columnas en común son iguales. A la columna en común, se la denomina columna de unión. La sintaxis para obtener tal resultado es:

```
SELECT tabla1.*, tabla2.* FROM tabla1, tabla2
    WHERE tabla1.columna_de_union = tabla2.columna_de_union;
```

Es de destacar que en este caso, es necesario usar el nombre de identificación de la tabla con cada una de las columnas de unión para especificar claramente que proceden de tablas distintas. Los resultados podrían ser los siguientes:

TABLA4					
T1A	**T1B**	**T1C**	**T2C**	**T2D**	**T2E**
1	1	1	1	4	5
1	2	1	1	4	5
2	2	1	1	4	5
1	2	2	2	6	7

La unión natural

La unión natural, a la que se conoce normalmente como unión, se la puede obtener, formado una unión equidistante con la columna común de dos tablas, y suprimiendo a continuación el duplicado de la columna en común. El resultado será:

TABLA 5				
A	**B**	**C**	**D**	**E**
1	1	1	4	5
1	2	1	4	5
2	2	1	4	5
1	2	2	6	7

que es similar a la tabla4 anterior a la que se le ha suprimido la columna duplicada C.

En SQL se puede obtener la unión natural de las tablas1 y 2, introduciendo:

```
SELECT tabla1.*, D, E
        FROM  tabla1, tabla2
        WHERE tabla1.C = tabla2.C;
```

No es necesario usar la identificación del nombre de la tabla para las columnas D y E en la instrucción **SELECT** porque esas columnas sólo aparecen en la tabla2 anterior.

La unión de columnas especificadas

Si no se desea mostrar todas las columnas que aparecen en la unión natural, puede formase en este caso, una unión de las columnas especificadas, haciendo un listado de las columnas que se desean obtener, después de **SELECT**:

```
SELECT tabla1.columnas, tabla2.columnas  FROM  tabla1, tabla2
                        WHERE columna1_de_union = columna2_de_union;
```

esto significa que la recuperación, constará de las columnas que se desean que aparezcan de la tabla1 y de la tabla2, siempre que el valor de la fila de la columna de unión de la tabla1, sea igual que el valor de la fila de la columna de unión de la tabla2. Como ejemplo, si se desea asignar clientes a los vendedores que viven en la misma zona que el cliente, y si se tiene una tabla de

|vendedores que contiene sólo los vendedores y sus zonas, y una tabla de clientes que muestra sólo los clientes y sus zonas, entonces, se puede unir las tablas de vendedores con la de clientes, allí donde la zona del vendedor se corresponde con la misma zona del cliente de la siguiente manera:

```
SELECT vendedores.*, clientes.*
        FROM  vendedores, clientes
        WHERE  vendedores.zona = clientes.zona
        ORDER BY  zona;
```

Como puede observarse, se califica a los nombres dentro de la columna dentro de la cláusula **WHERE** mediante la colocación delante de los mismos el nombre de la tabla. Esto es así porque estas columnas tienen el mismo nombre en cada una de las dos tablas, y el nombre de la columna por sí sólo, es ambiguo. Por otra parte, debe hacerse notar, que las columnas de unión, no tienen por qué tener el mismo nombre.

La unión equidistante, recuperará las columnas pedidas, y entre éstas habrá dos columnas idénticas (es decir, columna1_de_union y columna2_de_union - zona en este caso -). Los resultados entonces, tendrán el siguiente aspecto:

Vendedores	
Vendedor	**Zona**
Baudino	Sur
Scrutiño	Guadalupe
Gauderio	Costanera
Deyro	Guadalupe

Clientes	
Cliente	**Zona**
Cortese	Guadalupe
Anchaval	Costanera
Ingaramo	Sur
Rossi	Guadalupe
Ortiz	Sur
López	Sur

Vendedor	**Zona**	**Zona**	**Cliente**
Baudino	Sur	Sur	Ortiz
Baudino	Sur	Sur	Ingaramo
Baudino	Sur	Sur	López
Scrutiño	Guadalupe	Guadalupe	Cortese
Deyro	Guadalupe	Guadalupe	Rossi
Gauderio	Costanera	Costanera	Anchaval

Para obtener esta unión sin la columna duplicada, se deben seleccionar todas las columnas de una de las tablas, y luego seleccionar sólo la columna necesaria de la otra tabla, prescindiendo de la columna que sería la duplicada. En la consulta siguiente, se seleccionan todas las columnas de la tabla de vendedores y sólo la del nombre de la tabla de clientes:

```
SELECT vendedores.*, clientes.cliente  FROM  vendedores, clientes
        WHERE  vendedores.zona = clientes.zona
        ORDER BY  zona;
```

En este caso, se mostrará sólo una de las dos columnas de las Zonas.

Las uniones NO equidistantes (uniones *theta*)

Se llamará unión NO equidistante a la unión de dos tablas, en las que la columna de unión de una tabla, no es igual a la columna de unión correspondiente de la otra tabla. Algunos autores les denominan uniones theta. La sintaxis general de una unión no equidistante es:

```
SELECT      tabla1.columnas, tabla2.columnas
    FROM    tabla1, tabla2
    WHERE   columna1_de_union (cualquier operador de comparación excepto =)
            columna2_de_union;
```

en donde el operador de comparación, puede ser: mayor que (>), menor que (<) y distinto de (<>). Como ejemplo, en vez de asignar vendedores a los clientes en las mismas zonas que aquéllas en que viven los vendedores, se podría asignar vendedores a los clientes que no viven en la misma zona que aquellos. Para esto debe hacerse:

```
SELECT vendedores.*, clientes.*  FROM  vendedores, clientes
    WHERE  NOT (vendedores.zonas = clientes.zonas);
```

La cláusula **WHERE** también se podría expresar:

```
WHERE vendedores.zonas <> clientes.zonas;
```

El resultado haría que se correspondiesen los vendedores con los clientes de forma impredecible, con la excepción de que no habría filas en las que la zona del vendedor fuese la misma que la zona del cliente.

Condiciones adicionales en las consultas de uniones

Se puede usar la cláusula **WHERE** en la consulta de una unión para especificar cualquier número de condiciones. Para ello, se usa un **AND** que especificará la condición o condiciones adicionales. Si los vendedores de los ejemplos, estuvieran realmente listados en la tabla que contiene todos los empleados, entonces, tal vez, se querría apartar a los empleados cuya función no tiene relación con la venta, como los administrativos contables, los programadores, los analistas, etc., que de hecho, no están relacionados con los clientes de la manera en que interesa. Se puede lograr una consulta de este tipo, añadiendo otra condición a la consulta de la unión:

```
SELECT empleado.nombre, cliente.nombre   FROM   empleado, cliente
    WHERE empleado.zona = cliente.zona AND empleado.nom_funcion = "vendedor";
```

Esta consulta, unirá vendedores con clientes, de acuerdo con las zonas que se correspondan, sin establecer correspondencias entre el personal que no sea vendedor con los clientes. Se puede añadir cualquier número de condiciones a la cláusula **WHERE**, usando el operador **AND**.

La unión de más de dos tablas

Se puede unir cualquier número de tablas. Esto se logra indicando las tablas que hay que unir en la cláusula **FROM**, y usando el **AND** para añadir cualquier condición que pueda ser necesaria.

```
SELECT      nombres_de_las_columnas
      FROM      tabla1, tabla2, . . . , tablaN
      WHERE     (condicion1) AND
                (condicion2) AND
                . . . . . .     . . .
                (condicionN);
```

No existe límite teórico respecto al número posible de uniones, aun cuando el implemetador puede imponer un límite. En una consulta que trabaje con **n** tablas, se requieren al menos **n-1** condiciones de join.

La unión de una tabla consigo misma

Algunas veces, puede ser deseable unir filas de la misma tabla. Para facilitar esta situación se pueden usar alias. Por ejemplo, se podría desear obtener una lista de pares de vendedores que están en la misma ciudad. Para hacer esto, se podría listar la tabla dos veces en la cláusula **FROM** y después distinguir los dos listados dando a cada uno un alias. Así, para la consulta: "hacer una lista de todos los pares de vendedores que están en la misma ciudad", la sintaxis será:

```
SELECT primero.nombre, segundo.nombre
      FROM  vendedor primero, vendedor segundo
      WHERE primero.ciudad = segundo.ciudad  AND
            primero.cod_empleado  >  segundo.cod_empleado;
```

La cláusula **FROM**, asigna el alias primero a la tabla de vendedores y el alias segundo a la misma tabla de vendedores. La especificación que va detrás de AND, que indica que el código de empleado debe ser mayor que el segundo, evita que se produzca el resultado que un vendedor se empareje consigo mismo (Scrutiño, Scrutiño), y evita que dé el mismo emparejamiento dos veces (Scrutiño, Deyro) y (Deyro, Scrutiño). Para algunas consultas, hay otras formas de combinar una tabla consigo misma, pero el uso de los alias, suele aclarar mucho más el significado. Un caso muy particular ya visto en las subconsultas, es el constituido por las subconsultas correlacionadas.

Las uniones externas

En algunas situaciones, puede ser útil recuperar filas que satisfacen una de las condiciones de la unión, pero que no satisfacen ambas. A estos casos, se les llama uniones externas. Las uniones externas, son importantes porque pueden recuperar datos que, en otro caso, podrían per-

derse si la condición de la unión sola, se usa para la recuperación. Por ejemplo, una columna de la unión puede contener filas que se ajustaría a la condición de unión, excepto que tienen valores NULL en la segunda columna de la unión. Podría ocurrir que si esos valores NULL se rellenasen con valores que no sean NULL, la condición de la unión quedase completamente satisfecha. Con la recuperación de la unión, no se obtendrían estas filas. La recuperación de la unión externa devolvería estas filas; después se las puede examinar para determinar si el proporcionar los valores que faltan satisfarían la condición de la unión. Puede resultar útil para comprender las uniones externas el darse cuenta que las uniones son análogas a la intersección de conjuntos (la unión incluye todas las filas que son miembros de ambos conjuntos) y las uniones externas son análogas a la unión (exclusiva) de conjuntos (la unión externa incluye las filas que pertenecen a uno, pero no a los dos conjuntos especificados.)

Las uniones externas no están incluidas directamente en el lenguaje SQL estándar. De todas maneras, la mayoría de las bases de datos comerciales de la actualidad contemplan esta posibilidad, a través de la cláusula opcional OUTER especificado en la SELECT.

Un ejemplo de la necesidad de recuperar una unión externa, se puede ver en la correspondencia entre las zonas de los vendedores y las zonas de los clientes. Si hubiese un cliente en una zona en que no viviese ningún vendedor, no habría una correspondencia con este cliente. Por consiguiente, no habría un vendedor que atendiese el cliente. De forma análoga, si hubiese un vendedor que viviese en una zona en la que no hubiere clientes, este vendedor no tendría clientela asignada.