TEMA 5: LENGUAJE SQL. CONSULTAS

David Bataller Signes

ÍNDICE

1. Introducción

Las aplicaciones informáticas utilizadas en la actualidad para la gestión de cualquier organización mueven una cantidad considerable de datos que se almacenan en bases de datos gestionadas por sistemas gestores de bases de datos (SGBD).

Hay bases de datos ofimáticas que dan bastante prestaciones para almacenar información que podríamos llamar doméstica. Pero las bases de datos ofimáticas suelen no tener suficientes recursos cuando se trata de gestionar grandes volúmenes de información a la que deben poder acceder muchos usuarios simultáneamente desde la red local y también desde puestos de trabajo remotos y, por este motivo, aparecen las bases de datos corporativas.

Todos los SGBD (ofimáticos y corporativos) incorporan el lenguaje SQL (structured query language) para poder dar instrucciones al sistema gestor y así poder efectuar altas, bajas, consultas y modificaciones, y crear las estructuras de datos (tablas e índices), y los usuarios para que puedan acceder a las bases de datos, concederse y revocarlos los permisos de acceso, etc. El trabajo en SGBD ofimáticos acostumbra efectuar sin utilizar este lenguaje, ya que la interfaz gráfica que aporta el entorno suele permitir cualquier tipo de operación. Los SGBD corporativos también aportan (cada vez más) interfaces gráficas potentes que permiten efectuar muchas operaciones pero, aún así, se hace necesario el conocimiento del lenguaje SQL para efectuar múltiples tareas.

En esta unidad, haremos los primeros pasos en el diseño de consultas sencillas en la base de datos, para pasar a ampliar nuestro conocimiento sobre el lenguaje SQL para aprovechar toda la potencia que da en el ámbito de la consulta de información. Así, por ejemplo, aprenderemos a ordenar la información, a agruparla efectuando filtrados para reducir el número de resultados, combinar resultados de diferentes consultas ... Es decir, que este lenguaje es una maravilla que posibilita efectuar cualquier tipo de consulta sobre una base de datos para obtener la información deseada.

2. Consultas simples

Una vez ya conocemos los diferentes tipos de datos que nos podemos encontrar almacenados en una base de datos (nosotros nos hemos centrado en MySQL, pero en el resto de SGBD es similar), estamos en condiciones de iniciar la explotación de la base de datos, es decir, de comenzar la gestión de los datos. Evidentemente, para poder gestionar datos, previamente se debe haber definido las tablas que deben contener los datos, y para poder consultar datos hay que haberlos introducido antes.

El aprendizaje del lenguaje SQL se efectúa, sin embargo, en sentido inverso; es decir, empezaremos conociendo las posibilidades de consulta de datos sobre tablas ya

creadas y con datos ya introducidas. Necesitamos, sin embargo, conocer la estructura de las tablas que se deben gestionar y las relaciones existentes entre ellas.

Las tablas que gestionaremos forman parte de dos diseños diferentes, es decir, son tablas de temas disjuntos. Las podéis ver en el archivo ejemplos de base de datos que esta en “aules”:

Así pues, ya estamos en condiciones de introducirnos en el aprendizaje de las instrucciones de consulta SQL, que practicaremos en las tablas de los temas empresa y sanidad presentados.

Todas las consultas en el lenguaje SQL se hacen con una única sentencia, llamada SELECT, que se puede utilizar con diferentes niveles de complejidad. Y todas las instrucciones SQL finalizan, obligatoriamente, con un punto y coma.

Tal como lo indica su nombre, esta sentencia permite seleccionar lo que el usuario pide, el cual no debe indicar dónde lo tiene que ir a buscar ni como lo ha de hacer.

La sentencia SELECT consta de diferentes apartados que se suelen denominar cláusulas. Dos de estos apartados son siempre obligatorios y son los primeros que presentaremos. El resto de cláusulas deben utilizarse según los resultados que se quieran obtener.

2.1 Cláusulas SELECT y FROM

La sintaxis más simple de la sentencia SELECT utiliza estas dos cláusulas de manera obligatoria:

select <expresión/columna>, <expresión/columna>, … from <tabla>, <tabla>, ...;

La cláusula SELECT permite elegir columnas y/o valores (resultados de las expresiones) derivados de estas.

La cláusula FROM permite especificar las tablas en las que hay que ir a buscar las columnas o sobre las que se calcularán los valores resultantes de las expresiones.

Una sentencia SQL se puede escribir en una única línea, pero para hacer la sentencia más legible suelen utilizar diferentes líneas para las diferentes cláusulas.

Ejemplo de utilización simple de las cláusulas select y from.

En el tema empresa, se quieren mostrar los códigos, apellidos y oficios de los empleados. Este es un ejemplo claro de las consultas más simples: hay que indicar las columnas a visualizar y la tabla de donde visualizarlas. La sentencia es la siguiente:

select emp_no, apellidos, oficio from emp;

El resultado que se obtiene es el siguiente:

EMP_NO	APELLIDOS	OFICIO
7369	SÁNCHEZ	EMPLEADO
7499	ARROYO	VENDEDOR
7521	SALA	VENDEDOR
7566	JIMÉNEZ	DIRECTOR
7654	MARTÍN	VENDEDOR
7698	NEGRO	DIRECTOR
7782	CEREZO	DIRECTOR
7788	GIL	ANALISTA
7839	REY	PRESIDENTE
7844	TOVAR	VENDEDOR
7876	ALONSO	EMPLEADO
7900	JIMENO	EMPLEADO
7902	FERNÁNDEZ	ANALISTA
7934	MUÑOZ	EMPLEADO

14 rows selected

Ejemplo de utilización de expresiones en la cláusula select

En el tema empresa, se quieren mostrar los códigos, apellidos y salario anual de los empleados.
Como saben que en la tabla EMP consta el salario mensual de cada empleado, sabemos calcular, mediante el producto por el número de pagas mensuales en un año (12, 14, 15 ...?), su salario anual. Supondremos que el empleado tiene catorce pagas mensuales. Por tanto, en este caso, alguna de las columnas a visualizar es el resultado de una expresión:

*select emp_no, apellido, salario * 14 from emp;*

El resultado que se obtiene es el siguiente:

EMP_NO	APELLIDOS	SALARIO*14
7369	SÁNCHEZ	1456000
7499	ARROYO	2912000
7521	SALA	2275000
7566	JIMÉNEZ	5414500
7654	MARTÍN	2275000
7698	NEGRO	5187000
7782	CEREZO	4459000
7788	GIL	5460000
7839	REY	9100000
7844	TOVAR	2730000
7876	ALONSO	2002000

7900	JIMENO	1729000
7902	FERNÁNDEZ	5460000
7934	MUÑOZ	2366000

14 rows selected

Fijémonos que el lenguaje SQL utiliza los nombres reales de las columnas como títulos en la presentación del resultado y, en caso de columnas que correspondan a expresiones, nos muestra la expresión como título.

El lenguaje SQL permite dar un nombre alternativo (llamado alias) a cada columna. Para ello, se puede emplear la siguiente sintaxis:

select <expresión / columna> [as alias], <expresión / columna> [as alias], ...
from <tabla>, <tabla>, ...;

Tenga en cuenta lo siguiente:
• Si el alias es formado por varias palabras, hay que cerrarlo entre comillas dobles.
• Hay algunos SGBD que permiten la no utilización de la partícula as (como Oracle y MySQL) pero en otros es obligatoria (como el MS-Access).

Ejemplo de utilización de alias en la cláusula select

En el tema empresa, se quieren mostrar los códigos, apellidos y salario anual de los empleados. La instrucción para alcanzar el objetivo puede ser, con la utilización de alias:

*select emp_no, apellido, salario * 14 as "Salario Anual" from emp;*

Obtendríamos el mismo resultado sin la partícula as:

*select emp_no, apellidos, salario * 14 "Salario Anual" from emp;*

El resultado que se obtiene en este caso es el siguiente:

EMP_NO	APELLIDOS	SALARIO ANUAL
7369	SÁNCHEZ	1456000
7499	ARROYO	2912000
7521	SALA	2275000
7566	JIMÉNEZ	5414500
7654	MARTÍN	2275000
7698	NEGRO	5187000
7782	CEREZO	4459000
7788	GIL	5460000
7839	REY	9100000
7844	TOVAR	2730000
7876	ALONSO	2002000
7900	JIMENO	1729000

7902	FERNÁNDEZ	5460000
7934	MUÑOZ	2366000

14 rows selected

El lenguaje SQL facilita una manera sencilla de mostrar todas las columnas de las tablas seleccionadas en la cláusula from (y pierde la posibilidad de emplear un alias) y consiste en utilizar un asterisco en la cláusula select.

Ejemplo de utilización de asterisco en la cláusula select

Se nos pide que mostrar, en el tema empresa, toda la información que hay en la tabla que contiene los departamentos.
La instrucción que nos permite alcanzar el objetivo es la siguiente (fijémonos que la instrucción SELECT con asterisco nos muestra los datos de todas las columnas de la tabla):

*select * from dept;*

Y obtenemos el resultado esperado:

DEPT_NO	DNOMBRE	LOC
10	CONTABILIDAD	SEVILLA
20	INVESTIGACIÓN	MADRID
30	VENTAS	BARCELONA
40	PRODUCCIÓN	BILBAO

Aunque disponemos del asterisco para visualizar todas las columnas de las tablas de la cláusula from, a veces nos interesará conocer las columnas de una tabla para diseñar una sentencia SELECT adecuada a las necesidades y no utilizar el asterisco para visualizar todas las columnas.

Los SGBD suelen facilitar mecanismos para visualizar un descriptor breve de una tabla. En MySQL (y también en Oracle), disponemos del orden desc (no es sentencia del lenguaje SQL) para ello. Hay que emplearla acompañando el nombre de la tabla.

Ejemplo de obtención del descriptor de una tabla

Si necesitamos conocer las columnas que forman una tabla determinada (y sus características básicas), podemos obtener el descriptor: desc

SQL> desc dept;

Atributo	Null	Tipo
DEPT_NO	NOT NULL	INT(2)
DNOMBRE	NOT NULL	VARCHAR(14)
LOC		VARCHAR(14)

El orden DESC no es exactamente una orden SQL, sino una orden que facilitan los SGBD para visualizar la estructura o el diccionario de los datos almacenados en una BD concreta. Por lo tanto, como no se trata estrictamente de una orden SQL, admite la no utilización del punto y coma (;) al final de la sentencia. De este modo, los códigos siguientes son equivalentes:

• *SQL> desc dept;*
• *SQL> desc dept*

Supongamos que estamos conectados con el SGBD en la base de datos o el esquema por defecto que contiene las tablas correspondientes al tema empresa y que necesitamos acceder a tablas de una base de datos que contiene las tablas correspondientes al esquema sanidad. Lo podemos conseguir?

La cláusula from puede hacer referencia a tablas de otra base de datos. En esta situación, hay que anotar la tabla como <nombre_esquema>. <nombre_tabla>.

El acceso a objetos de otros esquemas (bases de datos) para un usuario conectado a un esquema sólo es posible si tiene concedidos los permisos de acceso correspondientes.

Ejemplo de acceso a tablas de otros esquemas

Si, estando conectados al esquema (base de datos) empresa, queremos mostrar los hospitales existentes en el esquema sanidad, habrá que hacer lo siguiente:

*select * from sanitat.hospital;*

El resultado que se obtiene es el siguiente:

HOSPITAL_COD	NOMBRE	DIRECCIÓN	TELÉFONO	CDAD_CAMAS
13	Provincial	Donella 50	9644264	88
18	General	Atocha s/n	5953111	63
22	La Paz	Castellana 100	9235411	162
45	San Carlos	Ciudad Universitaria	5971500	92

4 rows selected

El lenguaje SQL efectúa el producto cartesiano de todas las tablas que encuentra en la cláusula from. En este caso, puede haber columnas con el mismo nombre en diferentes tablas y, si es así y hay que seleccionar una, hay que utilizar obligatoriamente la sintaxis <nombre_tabla>. <nombre_columna> y, incluso, la sintaxis
<Nombre_esquema>. <Nombre_tabla>. <Nombre_columna> si se accede a una tabla de otro esquema.

Ejemplo de sentencia "SELECT" con varias tablas y coincidencia en nombres de columnas

Si desde el esquema empresa queremos mostrar el producto cartesiano de todas las filas de la tabla DEPT con todas las filas de la tabla SALA del esquema sanidad (visualización que no tiene ningún sentido, pero que hacemos a modo de ejemplo), mostrando únicamente las columnas que forman las claves primarias respectivas, ejecutaríamos lo siguiente:

select dept.dept_no, sanidad.sala.hospital_cod, sanidad.sala.sala_cod from dept, sanidad.sala;

El resultado obtenido es formado por cuarenta filas. Mostramos sólo algunas:

DEPT_NO	HOSPITAL_COD	SALA_COD
10	13	3
10	13	6
10	18	3
...	...	...
40	45	1
40	45	2
40	45	4

40 rows selected

En este caso, la sentencia se hubiera podido escribir sin utilizar el prefijo sanidad en las columnas de la tabla SALA en la cláusula select, ya que en la cláusula from no aparece más de una tabla llamada SALA y, por tanto, no hay problemas de ambigüedad. Pudimos escribir lo siguiente:

select dept.dept_no, sala.hospital_cod, sala.sala_cod from dept, sanitat.sala;
El lenguaje SQL permite definir alias para una tabla. Para conseguirlo, hay que escribir el alias en la cláusula from después del nombre de la tabla y antes de la coma que la separa de la tabla siguiente (si existe) de la cláusula from.

Ejemplo de utilización de alias para nombres de tablas

Si estamos conectados al esquema empresa, para obtener el producto cartesiano de todas las filas de la tabla DEPT con todas las filas de la tabla SALA del esquema sanidad, que muestre únicamente las columnas que forman las claves primarias respectivas, podríamos ejecutar la instrucción siguiente:

select d.dept_no, s.hospital_cod, s.sala_cod from dept d, sanitat.sala s;

El lenguaje SQL permite utilizar el resultado de una sentencia SELECT como tabla en la cláusula from de otra sentencia SELECT.

Ejemplo de sentencia "SELECT" como tabla en una cláusula from

Así, pues, otra manera de obtener, estando conectados al esquema empresa, el producto cartesiano de todas las filas de la tabla DEPT con todas las filas de la tabla SALA del esquema sanidad, que muestre únicamente las columnas que forman las claves primarias respectivas, sería la siguiente:
select d.dept_no, h.hospital_cod, h.sala_cod from dept d, (select hospital_cod, sala_cod from sanitat.sala) h;

MySQL (igual que Oracle) incorpora una tabla ficticia, llamada DUAL, para efectuar cálculos independientes de cualquier tabla de la base de datos aprovechando la potencia de la sentencia SELECT.

Así pues, podemos usar esta tabla para hacer lo siguiente:

1. Realizar cálculos matemáticos

*SQL> select 4 * 3 - 8/2 as "Resultado" from dual;*

RESULTADO 8
1 rows selected

2. Obtener la fecha del sistema, sabiendo que proporciona la función SYSDATE ()

SQL> select SYSDATE () from dual;

SYSDATE () 09/02/08
1 rows selected

La tabla DUAL también puede ser elidida. De este modo, las siguientes sentencias serían equivalentes a las expuestas anteriormente:

select 4 * 3 - 8/2 as "Resultado";

select sysdate();

2.2 Cláusula ORDER BY

La sentencia SELECT tiene más cláusulas aparte de las conocidas select y from. Así, tiene una cláusula order by que permite ordenar el resultado de la consulta.

Ejemplo de ordenación de los datos utilizando la cláusula ORDER BY

Si se quieren obtener todos los datos de la tabla departamentos, ordenadas por el nombre de la localidad, podemos ejecutar la siguiente sentencia:

*SELECT * FROM DEPT ORDER BY loc;*

Y obtendremos el siguiente resultado:

DEPT_NO	DNOMBRE	LOC
30	VENTAS	BARCELONA
40	PRODUCCIÓN	BILBAO
20	INVESTIGACIÓN	MADRID
10	CONTABILIDAD	SEVILLA

2.3 Cláusula Where

La cláusula where añade detrás de la cláusula from lo que ampliamos la sintaxis de la sentencia SELECT:

select <expresión / columna>, <expresión / columna>, ...
from <tabla>, <tabla>, ...
[where <condición_de_búsqueda>];

La cláusula where permite establecer los criterios de búsqueda sobre las filas generadas por la cláusula from.
La complejidad de la cláusula where es prácticamente ilimitada gracias a la abundancia de operadores disponibles para efectuar operaciones.

1. Operadores aritméticos

Son los típicos operadores +, -, *, / utilizables para formar expresiones con constantes, valores de columnas y funciones de valores de columnas.

2. Operadores de fechas

Con el fin de obtener la diferencia entre dos fechas:

Operador -, para restar dos fechas y obtener el número de días que las separan.

3. Operadores de comparación

Disponemos de diferentes operadores para efectuar comparaciones:
• Los típicos operadores =,! =,>, <,> =, <=, Para efectuar comparaciones entre datos y obtener un resultado booleano: verdadero o falso.

• El operador [NOT] LIKE, para comparar una cadena (parte izquierda del operador) con una cadena patrón (parte derecha del operador) que puede contener los caracteres especiales:
-% para indicar cualquier cadena de cero o más caracteres.
- _ para indicar cualquier carácter.
Así:

LIKE 'Torres' compara con la cadena 'Torres'.
LIKE 'Torr%' compara con cualquier cadena iniciada por 'Torr'.
LIKE '% S%' compara con cualquier cadena que contenga 'S'.
LIKE '_U%' compara con cualquier cadena que tenga por segundo carácter una 'U'.
LIKE '%% __ %%' compara con cualquier cadena de dos caracteres.

Un último conjunto de operadores lógicos:

[NOT] BETWEEN valor_1 AND valor_2 que permite efectuar la comparación entre dos valores.

[NOT] IN (lista_valores_separados_por_comas) que permite comparar con una lista de valores.

IS [NOT] NULL que permite reconocer si nos encontramos ante un valor null.

<comparador genérico> AÑO (lista_valores) que permite efectuar una comparación genérica (=,! =,>, <,> =, <=) con cualquiera de los valores de la derecha. Los valores de la derecha serán el resultado de ejecución de otra consulta (SELECT), por ejemplo:

*select * from emp where apellidos! = Año (select 'Alonso' from dual);*

<comparador genérico> ALL (lista_valores) que permite efectuar una comparación genérica (=,! =,>, <,> =, <=) con todos los valores de la derecha. Los valores de la derecha serán el resultado de ejecución de otra consulta (SELECT), por ejemplo:

*select * from emp where apellidos!= all (select 'Alonso' from dual);*

Ejemplo de filtrado simple en la cláusula where

En el tema empresa, se quieren mostrar los empleados (código y apellido) que tienen un salario mensual igual o superior a 200.000 y también su salario anual (suponemos que en un año hay catorce pagas mensuales).

La instrucción que permite alcanzar el objetivo es ésta:

*select emp_no as "Código", apellidos as "Empleado", salario * 14 as "Salario anual" from emp where salario> = 200000;*

El resultado obtenido es el siguiente:

Código	Empleado	Salario anual
7499	ARROYO	2912000
7566	JIMENEZ	5414500
7698	NEGRO	5187000
7782	CEREZO	4459000
7788	GIL	5460000
7839	REY	9100000
7902	FERNANDEZ	5460000

7 rows selected

Ejemplo de filtrado de fechas utilizando la especificación ANSI para indicar una fecha

En el tema empresa, se quieren mostrar los empleados (código, apellido y fecha de contratación) contratados a partir del mes de junio de 1981.

La instrucción que permite alcanzar el objetivo es ésta:

select emp_no as "Código", apellido as "Empleado", fecha_alta as "Contrato" from emp where data_alta> = '1981-06-01';

El resultado obtenido es el siguiente:

Codigo	Empleado	Contrato
7654	MARTIN	29/09/81
7782	CEREZO	09/06/81
7788	GIL	09/11/81
7839	REY	17/11/81
7844	TOVAR	08/09/81
7876	ALONSO	23/09/81
7900	JIMENO	03/12/81
7902	FERNANDEZ	03/12/81
7934	MUÑOZ	23/01/82

9 rows selected

Ejemplo de utilización de operaciones lógicas en la cláusula where

En el tema empresa, se quieren mostrar los empleados (código, apellido) resultado de la intersección de los dos últimos ejemplos, es decir, empleados que tienen un sueldo mensual igual o superior a 200.000 y contratados a partir del mes de junio de 1981.

La instrucción para conseguir lo que se nos pide es ésta:

select emp_no as "Código", apellido as "Empleado" from emp where fecha_alta> = '1981 06 01' and salario> = 200000;

El resultado obtenido es el siguiente:

Código	Empleado
7782	CEREZO
7788	GIL
7839	REY
7902	FERNANDEZ

4 rows selected

Ejemplo 1 de utilización del operador like

En el tema empresa, se quieren mostrar los empleados que tienen como inicial del apellido una 'S'.

La instrucción solicitada puede ser esta:

select apellido as "Empleado" from emp where apellido like 'S%';

Esta instrucción muestra los empleados con el apellido comenzado por la letra 'S' mayúscula, y se supone que los apellidos están introducidos con la inicial en mayúscula, pero, a fin de asegurar la solución, en el enunciado se puede utilizar la función incorporada upper (), que devuelve una cadena en mayúsculas:

select apellido as "Empleado" from emp where upper (apellido) like 'S%';

Ejemplo 2 de utilización del operador like

En el tema empresa, se quieren mostrar los empleados que tienen alguna S en el apellido.

La instrucción solicitada puede ser esta:

select apellido as "Empleado" from emp where upper (apellido) like '%S%';

Ejemplo 3 de utilización del operador like

En el tema empresa, se quieren mostrar los empleados que no tienen la R como tercera letra del apellido.

La instrucción solicitada puede ser esta:

select apellido as "Empleado" from emp where upper (apellido) not like '__R%';

Ejemplo de utilización del operador between

En el tema empresa, se quieren mostrar los empleados que tienen un salario mensual entre 100.000 y 200.000.
La instrucción solicitada puede ser esta:

select emp_no as "Código", apellido as "Empleado", salario as "Salario" from emp where salario> = 100000 and salario <= 200000;

Sin embargo, podemos utilizar el operador between:

select emp_no as "Código", apellido as "Empleado", salario as "Salario" from emp where salario between 100000 and 200000;

Ejemplo de utilización de los operadores in o =any

En el tema empresa, se quieren mostrar los empleados de los departamentos 10 y 30.
La instrucción solicitada puede ser esta:

select emp_no as "Código", apellido as "Empleado", dept_no as "Departamento" from emp where dept_no = 10 or dept_no = 30;

Sin embargo, podemos utilizar el operador in:

select emp_no as "Código", apellido as "Empleado", dept_no as "Departamento" from emp where dept_no in (10,30);

Ejemplo de utilización del operador "not in"

En el tema empresa, se quieren mostrar los empleados que no trabajan en los departamentos 10 y 30.
La instrucción solicitada puede ser esta:

select emp_no as "Código", apellido as "Empleado", dept_no as "Departamento" from emp where dept_no !=10 and dept_no !=30;

3. Consultas de selección complejas

Una vez conocemos los tipos de datos que facilita el SGBD y sabemos utilizar la sentencia SELECT para la obtención de información de la base de datos con la utilización de las cláusulas select (selección de columnas y / o expresiones), from (Selección de las tablas correspondientes) y where (filtrado adecuado de las filas que

interesan), estamos en condiciones de profundizar en las posibilidades que tiene la sentencia SELECT, ya que sólo conocemos las cláusulas básicas.

A la hora de obtener información de la base de datos, nos interesa poder incorporar, en las expresiones de las cláusulas select y where, cálculos genéricos que los SGBD facilitan con funciones incorporadas (cálculos como el valor absoluto, redondeos, truncamientos, extracción de subcadenas en cadenas de caracteres, extracción del año, mes o día en fechas ...), así como poder efectuar consultas más complejas que permitan clasificar la información, efectuar agrupamientos de filas, realizar combinaciones entre diferentes tablas e incluir los resultados de consultas dentro de otras consultas.

3.1 Funciones incorporadas a MySQL

Los SGBD suelen incorporar funciones utilizables desde el lenguaje SQL.
El lenguaje SQL, en sí mismo, no incorpora funciones genéricas, a excepción de las llamadas funciones de agrupamiento.

Las funciones incorporadas proporcionadas por los SGBD se pueden utilizar dentro de expresiones y actúan con los valores de las columnas, variables o constantes en las cláusulas select, where y order by.

También es posible utilizar el resultado de una función como valor para utilizar en otra función.

Las funciones principales facilitados por MySQL son las de matemáticas, de cadenas de caracteres, de gestión de momentos temporales, de control de flujo, pero disponemos de más funciones. Siempre será necesario consultar la documentación de MySQL.

3.1.1 Funciones matemáticas

Podéis encontrar una copia del manual de MySQL en español de estas funciones en la página http://download.nust.na/pub6/mysql/doc/refman/5.0/es/mathematical-functions.html

3.1.2 Funciones de cadenas de caracteres

Podéis encontrarlas en http://download.nust.na/pub6/mysql/doc/refman/5.0/es/string-functions.html

3.1.3 Funciones de gestión de fechas

Podéis encontrarlas en http://download.nust.na/pub6/mysql/doc/refman/5.0/es/date-and-time-functions.html

Notaciones para formatear las fechas en MySQL

Especificador	Descripción
%a	Día de semana abreviado (Sun..Sat)
%b	Mes abreviado (Jan..Dec)
%c	Mes, numérico (0..12)
%D	Día del mes con sufijo inglés (0th, 1st, 2nd, 3rd, ...)
%d	Día del mes numérico (00..31)
%e	Día del mes numérico (0..31)
%f	Microsegundos (000000..999999)
%H	Hora (00..23)
%h	Hora (01..12)
%I	Hora (01..12)
%i	Minutos, numérico (00..59)
%j	Día del año (001..366)
%k	Hora (0..23)
%l	Hora (1..12)
%M	Nombre mes (January..December)
%m	Mes, numérico (00..12)
%p	AM o PM
%r	Hora, 12 horas (hh:mm:ss seguido de AM o PM)
%S	Segundos (00..59)
%s	Segundos (00..59)
%T	Hora, 24 horas (hh:mm:ss)
%U	Semana (00..53), donde domingo es el primer día de la semana
%u	Semana (00..53), donde lunes es el primer día de la semana
%V	Semana (01..53), donde domingo es el primer día; usado con %X
%v	Semana (01..53), donde lunes es el primer día; usado con %x
%W	Nombre día semana (Sunday..Saturday)
%w	Día de la semana (0=Sunday..6=Saturday)
%X	Año para la semana donde domingo es el primer día de la semana, numérico, cuatro dígitos; usado con %V
%x	Año para la semana, donde lunes es el primer día de la semana, numérico, cuatro dígitos; usado con %v
%Y	Año, numérico, cuatro dígitos
%y	Año, numérico (dos dígitos)
%%	Carácter '%' literal

Así, por ejemplo, se puede mostrar la fecha y/o la hora utilizando expresiones como las siguientes, obteniendo los resultados indicados:

SELECT DATE_FORMAT ('1997 10 04 22:23:00', '% W% M% Y');
Devuelve: 'Saturday October 1997'

SELECT DATE_FORMAT ('1997 10 04 22:23:00', '% H:% y:% s');
Devuelve: '22:23:00 '

SELECT DATE_FORMAT ('1997 10 04 22:23:00', '% D% y% a% d% m% b% j');
Devuelve: '4th 97 Sat 04 10 Oct 277'

SELECT DATE_FORMAT ('1997 10 04 22:23:00', '% H% k% Y% r% T% S% w');
Devuelve: '22 22 10 10:23:00 PM 22:23:00 00 6'

SELECT DATE_FORMAT ('1999 01 01', '% X% V');
Devuelve: '1998 52'

3.1.4 Funciones de control de flujo

Aunque el lenguaje SQL no es un lenguaje de programación de aplicaciones estrictamente, sí, en algunas operaciones sobre la base de datos es último realizar una operación o considerar unos valores u otros en función de un estado inicial o de partida. Por este motivo MySQL ofrece la posibilidad de incluir, dentro de la sintaxis de las sentencias SQL, unos operadores y unas funciones que permitan realizar acciones diferentes en función de unos estados.

Estas 4 funciones son CASE, IF, IFNULL y NULLIF. Podéis encontrar una explicación de su funcionamiento en:
http://download.nust.na/pub6/mysql/doc/refman/5.0/es/control-flow-functions.html

Otras funciones de MySQL

MySQL proporciona otras funciones propias que enriquecen el lenguaje. Se proporcionan funciones para acceder a código XML y obtener datos, soportando el lenguaje XPath 1.0 de acceso a datos XML.

También proporciona operadores que permiten trabajar a nivel de operaciones de bits (inversión de bits, operaciones de AND, OR o XOR o desplazamientos de bits, por ejemplo).

MySQL dispone de funciones para comprimir (ENCODE ()) y descomprimir (DECODE ()) datos, y también para encriptar su (ENCRYPT ()) y descifrar (DES_DECRYPT ()).

Evidentemente, MySQL también ofrece una serie de operaciones diversas de administración del SGBD que permiten acceder al diccionario de datos.

Para todas estas otras operaciones será necesario consultar la guía de referencia del lenguaje que se puede encontrar en el sitio:
http://download.nust.na/pub6/mysql/doc/refman/5.0/es/other-functions.html

3.2 Clasificación de filas. Cláusula ORDER BY

La cláusula select permite decidir qué columnas se seleccionarán del producto cartesiano de las tablas especificadas en la cláusula from, y la cláusula where filtra las filas correspondientes. No se puede asegurar, sin embargo, el orden en que el SGBD dará el resultado.

La cláusula order by permite especificar el criterio de clasificación del resultado de la consulta.

Esta cláusula se añade detrás de la cláusula where si las hay, de manera que ampliamos la sintaxis de la sentencia SELECT:

select <expresión / columna>, <expresión / columna>, ...
from <tabla>, <tabla>, ...
[where <condición_de_busqueda>]
[order by <expresión / columna> [asc | desc], <expresión / columna> [asc | desc], ...];

Como se puede ver, la cláusula order by permite ordenar la salida según diferentes expresiones y/o columnas, que deben ser calculables a partir de los valores de las columnas de las tablas de la cláusula from aunque no aparezcan en las columnas de la cláusula select.

Las expresiones y/o columnas de la cláusula order by que aparecen en la cláusula select se pueden referenciar por el número ordinal de la posición que ocupan en la cláusula select en lugar de escribir su nombre.

El criterio de ordenación depende del tipo de dato de la expresión o columna y, por tanto, podrá ser numérico o lexicográfico.

Cuando hay más de un criterio de ordenación (varias expresiones y/o columnas), se clasifican de izquierda a derecha.

La secuencia de ordenación predeterminado es ascendente para cada criterio. Se puede, sin embargo, especificar que la secuencia de ordenación para un criterio sea descendente con la partícula desc después del criterio correspondiente. También se puede especificar la partícula asc para indicar una secuencia de ordenación ascendente, pero es innecesario porque es la secuencia de ordenación por defecto.

Ejemplo de clasificación de resultados según varias columnas

En el esquema empresa, se quieren mostrar los empleados ordenados de manera ascendente por su salario mensual, y ordenados por el apellido cuando tengan el mismo salario.

La instrucción para alcanzar el objetivo es esta:

select emp_no as "Código", apellido as "Empleado", salario as "Salario"
from emp
order by salario, apellido;

En caso de que haya criterios de ordenación que también aparecen en la cláusula select y tengan un alias definido, se puede utilizar este alias en la cláusula order by.

Así, pues, tendríamos el siguiente:

select emp_no as "Código", apellido as "Empleado", salario as "Salario"
from emp
order by "Salario", "Empleado";

Y, como hemos dicho más arriba, también podríamos utilizar el ordinal:

select emp_no as "Código", apellido as "Empleado", salario as "Salario"
from emp
order by 3, 2;

Ejemplo de clasificación de resultados según expresiones

En el esquema empresa, se quieren mostrar los empleados con su salario y comisión ordenados, de manera descendente, por el sueldo total mensual (salario + comisión).

La instrucción para alcanzar el objetivo es esta:

select emp_no as "Código", apellido as "Empleado", salario as "Salario", comision as "Comisión"
from emp
order by salario + IFNULL (comisión, 0) desc;

Tenga en cuenta que si no utilizamos la función IFNULL también aparecen todos los empleados, pero todos los que no tienen comisión asignada aparecen agrupados al inicio, ya que el valor NULL se considera superior a todos los valores y el resultado de la suma salario + comision es NULL en las filas que tienen NULL en la columna comisión.

3.3 Exclusión de filas repetidas. Opción DISTINCT

La cláusula select permite decidir qué columnas se seleccionarán del producto cartesiano de las tablas especificadas en la cláusula from, y la cláusula where filtra las filas correspondientes. El resultado, sin embargo, puede tener filas repetidas, para las que puede interesar tener sólo un ejemplar.

La opción distinct acompañando la cláusula select permite especificar que se quiere un único ejemplar para las filas repetidas.

La sintaxis es la siguiente:

select [distinct] <expresión / columna>, <expresión / columna>, ...
from <tabla>, <tabla>, ...
[where <condición_de_busqueda>]
[order by <expresión / columna> [asc | desc], <expresión / columna> [asc | desc], ...];

La utilización de la opción distinct implica que el SGBD ejecute obligatoriamente una order by sobre todas las columnas seleccionadas (aunque no se especifique la cláusula order by), lo que implica un coste de ejecución adicional. Por lo tanto, la opción distinct debería utilizarse en caso de que pueda haber filas repetidas y interese un único ejemplar, y al estar seguro debería evitarse que no puede haber filas repetidas.

Ejemplo de la necesidad de utilizar la opción distinct

Como ejemplo en el que hay que utilizar la opción distinct veamos como se muestran en el esquema empresa, los departamentos (sólo el código) en los que hay algún empleado.

La instrucción para alcanzar el objetivo es esta:

select distinct dept_no as "Código"
from emp;

Evidentemente, la consulta no se puede efectuar sobre la tabla de los departamentos, ya que puede haber algún departamento que no tenga ningún empleado. Por este motivo, la ejecutamos sobre la tabla de los empleados y tenemos que utilizar la opción distinct, pues de otro modo un mismo departamento aparecería tantas veces como empleados tuviera asignados.

3.4 Agrupamientos de filas. Cláusulas GROUP BY y HAVING

Sabiendo cómo se seleccionan filas de una tabla o de un producto cartesiano de tablas (Cláusula where) y como quedarnos con las columnas interesantes (cláusula select), hay que ver cómo agrupar las filas seleccionadas y como filtrar por condiciones sobre los grupos.

La cláusula group by permite agrupar las filas resultado de las cláusulas select, from y where según una o más de las columnas seleccionadas.

La cláusula having permite especificar condiciones de filtrado sobre los grupos alcanzados por la cláusula group by.

Las cláusulas group by y having se añaden detrás la cláusula where (si las hay) y antes de la cláusula order by (si las hay), por lo que ampliamos la sintaxis de la sentencia SELECT:

select [distinct] <expresión / columna>, <expresión / columna>, ...
from <tabla>, <tabla>, ...
[where <condicion_de_busqueda>]
[group by <alias / columna>, <alias / columna>, ...]
[having <condicion_sobre_grupos>]
[order by <expresión / columna> [asc | desc], <expresión / columna> [asc | desc], ...];

A continuación podemos ver las funciones de agrupamiento más importantes que se pueden utilizar en las sentencias SELECT de selección de conjuntos.

AVG (n): Devuelve el valor medio de la columna n ignorando los valores nulos. Ejemplo: AVG (salario) devuelve el salario medio de todos los empleados seleccionados que tienen salario (los nulos se ignoran).

COUNT ([* o expr]): Devuelve el número de veces que expr evalúa algún dato con valor no nulo. La opción * contabiliza todas las filas seleccionadas. Ejemplo: COUNT (dept_no) (sobre la tabla de empleados) cuenta cuántos empleados están asignados a algún departamento.

MAX (expr): Devuelve el valor máximo de expr. Ejemplo: MAX (salario) devuelve el salario más alto.

MIN (expr): Devuelve el valor mínimo de expr. Ejemplo: MIN (salario) devuelve el salario más bajo.

STDDEV (expr): Devuelve la desviación típica de expr sin tener en cuenta los valores nulos. Ejemplo: STDDEV (salario) devuelve la desviación típica de los salarios.

SUM (expr): Devuelve la suma de los valores de expr sin tener en cuenta los valores nulos. Ejemplo: SUM (salario) devuelve la suma de todos los salarios.

VARIANCE (expr): Devuelve la varianza de expr sin tener en cuenta los valores nulos. Ejemplo: VARIANCE (salario) devuelve la varianza de los salarios.

La expresión sobre la que se calculan las funciones de agrupamiento puede ir precedida de la opción distinct para indicar que se evalúe sobre los valores distintos de la expresión, o de la opción all para indicar que se evalúe sobre todos los valores de la expresión. El valor por defecto es all.

Una sentencia SELECT es una sentencia de selección de conjuntos cuando aparece la cláusula group by o la cláusula having o una función de agrupamiento; es decir, una sentencia SELECT puede ser sentencia de selección de conjuntos aunque no haya cláusula group by, en cuyo caso se considera que hay un único conjunto formado por todas las filas seleccionadas.

Las columnas o expresiones que no son funciones de agrupamiento y que aparecen en una cláusula select de una sentencia SELECT de selección de conjuntos han de aparecer obligatoriamente en la cláusula group by de la sentencia. Ahora bien, no todas las columnas y expresiones de la cláusula group by deben aparecer necesariamente en la cláusula select.

Ejemplo de utilización de la función count () sobre toda la consulta

En el esquema empresa, se quiere contar cuántos empleados hay. La instrucción para alcanzar el objetivo es esta:

select count (*) as "¿Cuántos empleados" from emp;

Esta sentencia SELECT es una sentencia de selección de conjuntos aunque no aparezca la cláusula group by. En este caso, el SGBD ha agrupado todas las filas en un único conjunto para poderlas contar.

Ejemplo de utilización de la opción distinct en una función de agrupamiento

En el esquema empresa, se quiere contar cuántos oficios diferentes hay. La instrucción para alcanzar el objetivo es esta:

select count (distinct oficio) as "Cuántos oficios" from emp;

En este caso es necesario indicar la opción distinct, pues de lo contrario contaría todas las filas que tienen algún valor en la columna oficio, sin descartar los valores repetidos.

Ejemplo de utilización de la función count () sobre una consulta con grupos

En el esquema empresa, se quiere mostrar cuántos empleados hay de cada oficio. La instrucción para alcanzar el objetivo es esta:

select oficio as "Oficio", count (*) as "¿Cuántos empleados" from emp group by oficio;

El resultado obtenido es el siguiente:

Oficio	Cuántos empleados
EMPLEADO	4
VENDEDOR	4
ANALISTA	2
PRESIDENTE	1
DIRECTOR	3

5 rows selected

Ejemplo de coexistencia de las cláusulas group by y order by

En el esquema empresa, se quieren mostrar los departamentos que tienen empleados, acompañados del salario más alto de sus empleados y ordenados de manera ascendente por salario máximo. La instrucción para alcanzar el objetivo es esta:

select dept_no, max (salario) from emp group by dept_no order by max (salario);

O también:

select dept_no as "Código", max (salario) as "Máximo salario" from emp group by dept_no order by "Máximo salario";

O también:

select dept_no as "Código", max (all salario) as "Máximo salario" from emp group by dept_no order by "Máximo salario";

Ejemplo de coexistencia de las cláusulas group by y order by

En el esquema empresa, se quiere contar cuántos empleados de cada oficio hay en cada departamento, y ver los resultados ordenados por departamento de manera ascendente y por número de empleados de manera descendente. La instrucción para alcanzar el objetivo es esta:

select dept_no as "Código", oficio as "Oficio", count (*) as "¿Cuántos empleados"

from emp group by dept_no, oficio order by dept_no, 3 desc;

Ejemplo de coexistencia de las cláusulas group by y where

En el esquema empresa, se quiere mostrar cuántos empleados de cada oficio hay en el departamento 20. La instrucción para alcanzar el objetivo es esta:

select oficio as "Oficio", count (*) as "¿Cuántos empleados" from emp
where dept_no = 20 group by oficio;

Ejemplo de utilización de la cláusula having

En el esquema empresa, se quiere mostrar el número de empleados de cada oficio que hay para los oficios que tienen más de un empleado. La instrucción para alcanzar el objetivo es esta:

select oficio as "Oficio", count (*) as "¿Cuántos empleados" from emp
group by oficio having count (*)> 1;

3.5 Unión, intersección y diferencia de sentencias SELECT

El lenguaje SQL permite efectuar operaciones sobre los resultados de las sentencias SELECT para obtener un resultado nuevo.

Tenemos tres operaciones posibles: unión, intersección y diferencia. Los conjuntos que hay que unir, interseccionar o restar deben ser compatibles: igual cantidad de columnas y columnas compatibles -tipo de datos equivalentes- dos a dos.

3.5.1 Unión de sentencias SELECT

El lenguaje SQL proporciona el operador unión para combinar todas las filas del resultado de una sentencia SELECT con todas las filas del resultado de otra sentencia SELECT, y elimina cualquier duplicación de filas que se pudiera producir en el conjunto resultante. La sintaxis es la siguiente:

sentencia_select_sin_order_by
union
sentencia_select_sin_order_by
[order by …]

El resultado final mostrará, como títulos, los correspondientes a las columnas de la primera sentencia SELECT. Así, pues, en caso de querer asignar alias a las columnas, basta definirlos en la primera sentencia SELECT.

Ejemplo de la operación unión entre sentencias SQL

En el esquema sanidad, se quiere presentar el personal que trabaja en cada hospital, incluyendo el personal de la plantilla y los doctores, y mostrando el oficio que ejercen. Una posible instrucción para alcanzar el objetivo es esta:

```
select nombre as "Hospital", 'Doctor' as "Oficio", doctor_no as "Código",
apellido "Empleado" from hospital, doctor
where hospital.hospital_cod = doctor.hospital_cod
** unión **
select nombre, funcion, empleado_no, apellido from hospital, plantilla
where hospital.hospital_cod = plantilla.hospital_cod
order by 1,2;
```

Fijémonos que hemos asignado a los doctores como oficio la constante 'Doctor'. El resultado obtenido es este:

Hospital	Oficio	Codigo	Empleado
General	Doctor	585	Miller G.
General	Doctor	982	Cajal R.
General	Interno	6357	Karplus W.
La Paz	Doctor	386	Cabeza D.
La Paz	Doctor	398	Best K.
La Paz	Doctor	453	Galo D.
La Paz	Enfermero	8422	Bocina G.
La Paz	Enfermera	1009	Higueras D.
La Paz	Enfermera	6065	Rivera G.
La Paz	Enfermera	7379	Carlos R.
La Paz	Interno	9901	Adams C.
Provincial	Doctor	435	López A.
Provincial	Enfermero	3106	Hernández J.
Provincial	Enfermera	3754	Díaz B.
San Carlos	Doctor	522	Adams C.
San Carlos	Doctor	607	Nico P.
San Carlos	Enfermera	8526	Frank H.
San Carlos	Interno	1280	Amigó R.

3.5.2 Intersección y diferencia de sentencias SELECT

Otros SGBDR (no MySQL, en este caso) proporcionan operaciones de intersección y diferencia de consultas.

La intersección consiste en obtener un resultado de filas común (idéntico) entre dos sentencias SELECT concretas. La sintaxis más habitual para la intersección es:

```
sentencia_select_sin_order_by
** intersect **
```

sentencia_select_sin_order_by
[order by …]

La diferencia entre sentencias SELECT consiste en obtener las filas que se encuentran en la primera sentencia SELECT que no se encuentren en la segunda. La sintaxis habitual es utilizando el operador minus:

sentencia_select_sin_order_by
** minus **
sentencia_select_sin_order_by
[order by ...]

3.6 Combinaciones entre tablas

La cláusula from efectúa el producto cartesiano de todas las tablas que aparecen en la cláusula. El producto cartesiano no nos interesará casi nunca, sino únicamente un subconjunto de este.
Los tipos de subconjuntos que nos pueden interesar coinciden con los resultado de las combinaciones join, equi-join, natural-join y outer-join.

Operaciones para combinar tablas

Dadas dos tablas R y S, se define el join de I según el atributo A, y de S según el atributo Z, y se escribe R [AZ] S como el subconjunto de filas del producto cartesiano R S que verifican A Z en que es cualquiera de los operadores relacionales (>,> =, <, <=, ==,! =).

El equi-join es un join en que el operador es la igualdad. Se escribe R [A = Z] S.

El natural-join es un equi-join en que el atributo para el que se ejecuta la combinación sólo aparece una vez en el resultado. Se escribe R [A * Z] S.

La notación R * S indica el natural-join para todos los atributos del mismo nombre en ambas relaciones.

A veces, hay que tener el resultado del equi-join ampliado con todas las filas de una de las relaciones que no tienen tupla correspondiente en la otra relación. Nos encontramos ante un outerjoin y tenemos dos posibilidades (left o right) según donde se encuentre (izquierda o derecha) la tabla para la que deben aparecer todas las filas aunque no tengan correspondencia en la otra tabla.

Dadas dos relaciones R y S, se define el left-outer-join de I según el atributo A, y de S según el atributo Z, y se escribe por R [A = Z] S, como el subconjunto de filas del producto cartesiano R S que verifican A = Z (resultado de R [A = Z] S) más las filas

de R que no tienen, para el atributo A, correspondencia con ninguna tupla de S según el atributo Z, las cuales presentan valores NULL en los atributos provenientes de S.
Dadas dos relaciones R y S, se define el right-outer-join de I según el atributo A, y de S según el atributo Z, y se escribe R [A = Z] S, como el subconjunto de filas del producto cartesiano R S que verifican A = Z (resultado de R [A = Z] S) más las filas de S que no tienen, para el atributo Z, correspondencia con ninguna tupla de R según el atributo A, las que presentan valores NULL en los atributos provenientes de R.

También podemos considerar el full-outer-join de I según el atributo A, y de S según el atributo Z, y se escribe por R [A = Z] S, como la unión de un right-outer-join y de un left-outer-join. Recordemos que la unión de conjuntos no tiene en cuenta las filas repetidas. Es decir, con un full-outer-join conseguiríamos tener todas las filas de ambas tablas: las filas que tienen correspondencia para los atributos de la combinación y las filas que no tienen correspondencia.

Actualmente, tenemos varias maneras de efectuar combinaciones entre tablas, producto de la evolución de los estándares SQL y los diversos SGBD comerciales existentes: las combinaciones según la norma SQL-87 y las combinaciones según la norma SQL-92.

3.7 Subconsultas

A veces, es necesario ejecutar una sentencia SELECT para conseguir un resultado que hay que utilizar como parte de la condición de filtrado de otra sentencia SELECT. El lenguaje SQL nos facilita efectuar este tipo de operaciones con la utilización de las subconsultas.

Una subconsulta es una sentencia SELECT que se incluye en la cláusula where de otra sentencia SELECT. La subconsulta se cierra entre paréntesis y no incluye el punto y coma finales.

Una subconsulta puede contener, a la vez, otros subconsultas.

Ejemplo de subconsulta que calcula un resultado a utilizar en una cláusula where

En el esquema empresa, se pide mostrar los empleados que tienen salario igual o superior al salario medio de la empresa. La instrucción para alcanzar el objetivo es esta:

```
select emp_no as "Código", apellido as "Empleado", salario as "Salario" from emp
where salario> = (select avg (salario) from emp)
order by 3 desc, 1;
```

En ciertas situaciones puede ser necesario acceder desde la subconsulta a los valores de las columnas seleccionadas en la consulta. El lenguaje SQL lo permite sin problemas y, en caso de que los nombres de las columnas coincidan, se pueden utilizar alias.

Los nombres de columnas que aparecen en las cláusulas de una subconsulta se intentan evaluar, en primer lugar, como columnas de las tablas definidas en la cláusula from de la subconsulta, a menos que vayan acompañadas de alias que las identifiquen como columnas de una tabla en la consulta contenedora.

Ejemplo de subconsulta que hace referencia a columnas de la consulta contenedora

En el esquema empresa, se pide mostrar los empleados de cada departamento que tienen un salario menor que el salario medio del mismo departamento. La instrucción para alcanzar el objetivo es esta:

select dept_no as "Dpto", emp_no as "Código", apellido as "Empleado",
salario as "Salario" from emp e1
where salario> = (select avg (salario) from emp e2 where dept_no = e1.dept_no)
order by 1, 4 desc, 2;

Los valores devueltos por las subconsultas se utilizan en las cláusulas where como parte derecha de operaciones de comparaciones en las que intervienen los operadores: =, !=, <, <=, >, > =, [Not] in, %% <op> %% any y %% <op> %% all

Las subconsultas también se pueden vincular a la consulta contenedora por la partícula [Not] exists:

...
where [not] exists (subconsulta)

En este caso, la subconsulta suele hacer referencia a valores de las tablas de la consulta contenedora. Se llaman subconsultas sincronizadas.

Las consultas que pueden dar como resultado un único valor o ninguno pueden actuar como subconsultas en expresiones en las que el valor resultado se compara con cualquier operador de comparación.

Las consultas que pueden dar como resultado más de un valor (aunque en ejecuciones concretas sólo se de uno) nunca pueden actuar como subconsultas en expresiones en que los valores resultantes se comparan con el operador =, ya que el SGBDR no sabría con cuál de los resultados efectuar la comparación de igualdad y se produciría un error.

Si hay que aprovechar los resultados de más de una columna de la subconsulta, ésta se coloca a la derecha de la operación de comparación y en la parte izquierda se colocan los valores que se deben comparar, en el mismo orden que los valores devueltos por la subconsulta, separados por comas y encerrados entre paréntesis:

...
where (valor1, valor2 ...) <op> (select col1, col2 …)

Ejemplo de utilización del operador = para comparar con el resultado de una subconsulta

En el esquema empresa, se quieren mostrar los empleados que tienen el mismo oficio que el oficio que tiene el empleado de apellido 'ALONSO'. La instrucción para alcanzar el objetivo parece que podría ser esta:

select apellido as "Empleado" from emp
where oficio = (select oficio from emp where upper (apellido) = 'ALONSO')
and upper (apellido) != 'ALONSO';

En esta sentencia hemos utilizado el operador = de manera errónea, ya que no podemos estar seguros de que no hay dos empleados con el apellido 'ALONSO'. Como sólo hay uno, la sentencia se ejecuta correctamente, pero en caso de que hubiera más de uno, lo que puede suceder en cualquier momento, la ejecución de la sentencia provocaría el error antes mencionado.

Por lo tanto, deberíamos buscar otro operador de comparación para evitar este problema:

select apellido as "Empleado" from emp
where oficio in (select oficio from emp where upper (apellido) = 'ALONSO')
and upper (apellido) != 'ALONSO';

O también:

select apellido as "Empleado" from emp e
*where exists (select * from emp where upper (apellido)='ALONSO' and oficio=e.oficio)*
and upper (apellido) != 'ALONSO';

Ejemplo de utilización de los operadores ANY y EXISTS
En el esquema empresa, se pide mostrar los nombres y oficios de los empleados del departamento 20 el trabajo de los cuales coincida con la de algún empleado del departamento de 'VENTAS'. La instrucción para alcanzar el objetivo puede ser esta:

select apellido as "Empleado", oficio as "Oficio" from emp

where dept_no = 20 and oficio = ANY (select oficio from emp where dept_no = ANY (select dept_no from dept where upper (dnombre) = 'VENTAS'));

Esta instrucción está pensada para que el resultado sea correcto en caso de que pueda haber diferentes departamentos con nombre 'VENTAS'. En caso de que la columna dnombre tabla DEPT tuviera definida la restricción de unicidad, también sería correcta la instrucción siguiente:

select apellido as "Empleado", oficio as "Oficio" from emp
where dept_no = 20 and oficio = ANY (select oficio from emp
where dept_no = (select dept_no from dept where upper (dnombre) = 'VENTAS'));

Otra manera de resolver el mismo problema es con la utilización del operador EXISTS:

select apellido as "Empleado", oficio as "Oficio" from emp e where dept_no = 20
*and EXISTS (select * from emp, dept where emp.dept_no = dept.dept_no and upper (dnombre) = 'VENTAS' and oficio = e.ofici);*

Ejemplo de utilización del operador IN

En el esquema empresa, se pide mostrar los empleados con el mismo oficio y salario que 'JIMÉNEZ'. La instrucción para alcanzar el objetivo puede ser esta:

select emp_no "Código", apellido "Empleado" from emp where (oficio, salario)
IN (select oficio, salario from emp where upper (apellido) = 'JIMÉNEZ')
and upper (apellido)! = 'JIMÉNEZ';

Ejemplo de condición compleja de filtrado con varias subconsultas y operaciones

Se pide, en el esquema empresa, mostrar los empleados que efectúen el mismo trabajo que 'JIMÉNEZ' o que tengan un salario igual o superior al de 'FERNÁNDEZ'.

select emp_no "Código", apellido "Empleado" from emp
where (oficio IN (select oficio from emp where upper (apellido) = 'JIMÉNEZ')
and upper (apellido)! = 'JIMÉNEZ')
or (salario> = (select salario from emp where upper (apellido) = 'FERNÁNDEZ')
and upper (apellido)! = 'FERNÁNDEZ');