2. 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.

2.1. Funciones incorporadas a MySQL

Funciones de agrupamiento

La sentencia SELECT tiene más cláusulas aparte de las conocidas select, from y where. Así, tiene una cláusula que permite agrupar las filas resultantes de la consulta y aplicar funciones de agrupamiento: max () para el cálculo del valor más grande de cada grupo, min () para el cálculo del valor más pequeño de cada grupo , etc.

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.

2.1.1. funciones matemáticas

La tabla 2 .1 nos presenta las funciones y los operadores matemáticos principales proporcionados por el SGBD MySQL.

Tabla 2.1. Funciones matemáticas principales proporcionadas por el SGBD MySQL

Función u operador matemático	**Descripción**	**ejemplos**
ABS (x)	Devuelve el valor absoluto de x	
ACOS (x)	Devuelve el arco coseno de x	SELECT ACOS (1);
ASIN (x)	Devuelve el arco seno de x	SELECT ASIN (0.2);
Atan (x), atan2 (x, y)	Devuelve el arco tangente de x el arco tangente de las coordenadas (x, y), respectivamente	SELECT atan (2); SELECT atan (-2,2);
Ceil (x)	Devuelve el valor entero inmediatamente superior ax, ox si ya es un valor entero	SELECT CEILING (1:23); Devuelve 2. SELECT CEILING (3); retorna 3
CEILING (x)	Sinónimo de Ceil (x)	
CONV (x, b1, b2)	Convierte el número x, considerado expresado en base b1, a base b2.	SELECT CONV (10,10,2); retorna 1010
COS (x)	Devuelve el coseno de x	
COT (x)	Devuelve la cotangente de x	
CRC32 (x)	Calcula el valor CRC32 (*cyclic redundancy check*)	
Degrees (x)	Convertir x, expresado en radianes, en grados	SELECT Degrees (PI ());
DIV	Calcula la división entera	SELECT 5 DIV 2; Retorna 2.
/	Operador de división	SELECT 3/5; Devuelve: 0.60
EXP (x)	Calcula y elevado ax	
FLOOR (x)	Devuelve el valor entero inmediatamente inferior a x. O x, si ya es un valor entero.	SELECT FLOOR (-1.23); Devuelve: -2.
LN (x)	Devuelve el logaritmo de base y de x	
Log10 (x)	Devuelve el logaritmo de base 10 de x	
Log2 (x)	Devuelve el logaritmo de base 2 de x	
LOG (b, x)	Devuelve el logaritmo de base b de x	
-	operador resto	SELECT 5-3;
MOD (x, y)	Devuelve el resto de la división de xe y. Es equivalente a N% M, y también a N MOD M	SELECT MOD (29,9); Devuelve: 2

OCT (x)	Devuelve la representación octal del número x, expresado en decimal.	
PI ()	Devuelve el valor pi	
+	Operador de suma	SELECT 5 + 3;
POW (x, y)	Devuelve x elevado ay	
POWER (x, y)	Sinónimo de POW	
Radian (x)	Devuelve x, expresado en grados, en radianes	
RAND ()	Devuelve un valor real aleatorio entre 0 y 1	
ROUND (x)	Redondea x al número entero más próximo.	SELECT ROUND (1.9); Devuelve 2. SELECT ROUND (1.5); Devuelve 2. SELECT ROUND (1.2); Devuelve 1.
SIGN (x)	Devuelve -1 si x es negativo. Devuelve 1 si x es positivo. Y devuelve 0 si x = 0	
SIN (x)	Devuelve el valor del seno de x	
SQRT (x)	Devuelve la raíz cuadrada de x	
TAN (x)	Devuelve la tangente de x	
*	Operador de multiplicación	SELECT 5 * 3;
TRUNCATE (x, d)	Trunca XAD decimales	SELECT TRUNCATE (-1.999,1); devuelve -1.9
-	Operador cambio de signo	SELECT - 2; Devuelve: -2

En negrita se han destacado las funciones más utilizadas a la hora de manipular expresiones numéricas.

2.1.2. Funciones de cadenas de caracteres

La tabla 2 .2 nos presenta las funciones principales para la gestión de cadenas de caracteres proporcionadas por el SGBD MySQL.

Tabla 2.2. Principales funciones de gestión de cadenas de caracteres proporcionados por el SGBD MySQL

función	**Descripción**	**ejemplo**
ASCII (s)	Devuelve el valor numérico del carácter de más a la izquierda.	SELECT ASCII ('2'); Devuelve: 50
BIN (n)	Devuelve un *string* con la representación en binario del número n.	SELECT BIN (12); Devuelve: '1100'
BIT_LENGTH (s)	Devuelve la longitud en bits de la cadena s.	SELECT BIT_LENGTH ('texto'); Devuelve: 32
CHAR_LENGTH (s)	Devuelve el número de caracteres de la cadena s.	SELECT BIT_LENGTH ('texto'); Devuelve: 4
CHAR (n, ... [USING nom_charset])	Devuelve la lista de caracteres por cada entero.	SELECT CHAR (77,121,83,81, '76 '); Devuelve: 'MySQL'. Explicitando el tipo de caracteres: SELECT CHAR (83 USING utf8);
CHARACTER_LENGTH (s)	Sinónimo de CHAR_LENGTH (s).	

CONCAT (s1, s2, ...)	Devuelve la concatenación de s1, s2, etc. Si uno de los parámetros es null, la función devuelve null.	SELECT CONCAT ('My', 'S', 'QL'); Devuelve: 'MySQL'. Que es equivalente a SELECT 'My' 'S' 'QL';
CONCAT_WS (s, s1, s2, ...)	Devuelve la concatenación de s1, s2, etc, separados por s.	SELECT CONCAT_WS (',', 'Nombre', 'Teléfono', 'Dirección'); Devuelve: 'Nombre, Teléfono, Dirección'
ELT (n, s1, s2, ...)	Si n = 1, devuelve s1. Si n = 2, devuelve s2, etc	SELECT ELT (4, 'ej', 'Héja', 'Hej', 'foo'); Devuelve: 'foo'
EXPORT_SET (bits, donde, off [, separador [, nombre_de_bits]])	Devuelve una cadena en la que, por cada bit del valor bits, puede obtener una cadena donde y por cada bit reasignado obtiene una cadena off. Los bits en bits examinan de derecha a izquierda (de bits más pequeños a más grandes). Las cadenas se añaden al resultado de izquierda a derecha, separados por la cadena separador (',' por defecto). El número de bits examinados obtiene por nombre_de_bits (por defecto 64).	SELECT EXPORT_SET (5, 'Y', 'N', ',', 4); Devuelve: 'Y, N, Y, N'
FIELD (s, s1, s2, s3, ...)	Devuelve el índice (posición) del primer argumento en los siguientes argumentos s1, s2, etc.	SELECT FIELD ('ej', 'Hej', 'ej', 'Héja', 'Hej', 'foo'); Devuelve: 2
FIND_IN_SET (s1, s2)	Devuelve la posición en que se encuentra s1 dentro de s2.	SELECT FIND_IN_SET ('b', 'a, b, c, d'); Devuelve: 2
FORMATO (x, d)	Devuelve el número x, como un *string* formateado con la plantilla siguiente '#, ###, ###. ##' y con de decimales.	SELECT FORMATO (12332.1,4); Devuelve: '12, 332.1000 '
HEX (x)	Devuelve la representación hexadecimal de un número decimal o de un *string* .	SELECT HEX ('abc'); Devuelve: 616263
INSERT (s1, p, l, s2)	Insertar s2 en la posición pi longitud l de s1	SELECT INSERT ('Hola', 5, 3, 'mundo'); Devuelve: 'HolaMundo'
INSTR (s1, s2)	Devuelve la primera posición en que se encuentra s2 dentro de s1.	SELECT INSTR ('pie de página', 'página'); Devuelve: 8
LCASE (s)	Sinónimo de LOWER (s).	
LEFT (s, l)	Devuelve los del caracteres de más a la izquierda de s.	SELECT LEFT ('Hola mundo', 4); Devuelve: 'Hola'
LENGTH (s)	Devuelve la longitud de s en bytes.	SELECT LENGTH ('Hola'); Devuelve: 4
LIKE	Operador de comparación entre expresiones, utilizado, sobre todo en cláusulas WHERE.	SELECT 'David' LIKE 'Dav%'; Devuelve cierto (1)
LOAD_FILE (s)	Abrir el archivo llamado si devuelve el contenido.	
Locate (s1, s2, [p])	Devuelve la posición de la primera ocurrencia de s1 dentro de s2, a partir de la posición po de la primera posición, si no se especifica p.	SELECT locate ('bar', 'foobarbar'); retorna 4
LOWER (s)	Devuelve s en minúsculas.	SELECT LOWER ('Hola Mundo'); Devuelve: 'hola mundo'

LPAD (s1, n, s2)	Devuelve la cadena s1, alineada a la izquierda con la cadena s2 a una longitud de n caracteres.	SELECT LPAD ('hay', 4, '??'); Devuelve: '?? hay'
LTrim (s)	Borra los caracteres blancos que hay a la izquierda del *string* .	SELECT LTrim ('hola'); Devuelve: 'hola'
MAKE_SET (bits, s1, s2, ...)	Devuelve el conjunto de *strings* que seleccionan los índices indicados en bits	SELECT MAKE_SET (14, 'hello', 'nice', 'world'); Devuelve: 'hello, world'
MATCH	Operador de búsqueda.	
MID (s, p, l)	Devuelve la subcadena de s, de longitud l, que comienza en la posición p. Es sinónimo de substring (s, p, l).	SELECT MID ('Hola Mundo', 1,4); Devuelve: 'Hola'
NOT LIKE	Negación del operador LIKE.	
NOT regexp	Negación del operador de expresión regular.	
OCTET_LENGTH (s)	Sinónimo de LENGTH (s).	
ORD (s)	Devuelve el código ACSI del primer carácter de s.	
POSITION (s1 IN s2)	Sinónimo de locate (s1, s2).	
QUOTE (s)	Devuelve la cadena s formateada con los caracteres especiales para poder ser utilizada correctamente como código SQL.	SELECT QUOTE ('Don \' t! '); Devuelve: 'Don \' t! '
regexp	Operador para usar expresiones regulares.	
REPEAT (s, n)	Devuelve la cadena s repetida n veces.	SELECT REPEAT ('MySQL', 3); Devuelve: 'MySQLMySQLMySQL'
REPLACE (s, s1, s2)	Cambia las ocurrencias de s1 por s2 dentro de s.	SELECT REPLACE (' www.mysql.com ', 'w', 'Ww'); Devuelve: 'WwWwWw.mysql.com'
REVERSE (s)	Devuelve la cadena s con el orden inverso de los caracteres.	SELECT REVERSE ('abc'); Devuelve: 'cba'
RIGHT (s, l)	Devuelve los del caracteres de más a la derecha de s.	SELECT RIGHT ('Hola mundo', 3); Devuelve: 'mundo'
RLIKE	Sinónimo de expresión regular	
RPAD (s1, n, s2)	Devuelve la cadena s, alineada a la derecha con la cadena s2 con longitud n.	SELECT RPAD ('hay', 5, '?'); Devuelve: 'hay ???'
RTrim (s)	Devuelve s sin espacios en blanco a la derecha.	SELECT RIGHT ('Hola'); Devuelve: 'Hola'
Soundex (s)	Devuelve una cadena soudex de s.	
SOUNDS LIKE	Operador equivalente a Soundex (exp1) = Soundex (exp2).	
SPACE (n)	Devuelve una cadena de n espacios en blanco.	SELECT SPACE (6); Devuelve: ''
Strcmp (s1, s2)	Compara dos cadenas.	
Sustrato (s, p [, l])	Devuelve la subcadena de s de longitud la partir de la posición p. También admite las sintaxis siguientes: Sustrato (str, pos), sustrato (str FROM pos), sustrato (str, pos, len), sustrato (str FROM pos FOR instalan). Sinónimo también de substring.	SELECT sustrato ('Hola Mundo', 1,4); Devuelve: 'Hola'
SUBSTRING_INDEX (s, d, n)	Devuelve la subcadena de la cadena s antes de n ocurrencias del delimitador d.	SELECT SUBSTRING_INDEX (' www.mysql.com ', '.', 2); Devuelve: 'www.mysql'

Substring (s, p [, l])	Devuelve la subcadena de s de longitud la partir de la posición p. También admite las sintaxis siguientes: substring (str, pos), substring (str FROM pos), substring (str, pos, len), substring (str FROM pos FOR instalan).	SELECT sustrato ('Hola Mundo', 1,4); Devuelve: 'Hola'
TRIM ([{BOTH o LEADING o trailing} [remstr] FROM] s)	Se utiliza para eliminar los espacios en blanco del inicio y el final de la cadena s. Devuelve la cadena s con todos los prefijos y / o sufijos remstr eliminados. Por defecto, remstr es el espacio en blanco.	SELECT TRIM ('hola'); Devuelve: 'hola'
UCASE (s)	Sinónimo de UPPER (s).	
UNHEX (s)	Convierte los códigos hexadecimales en caracteres. Es la función contraria a HEX (s).	SELECT UNHEX ('4D7953514C'); Devuelve: 'MySQL'
UPPER (s)	Convertir sano mayúsculas.	SELECT UPPER ('Hola Mundo'); Devuelve: 'HOLA MUNDO'

En negrita se han destacado las funciones más utilizadas a la hora de manipular expresiones numéricas.

2.1.3. Funciones de gestión de fechas

La tabla 2 .3 nos presenta algunas de las funciones para la gestión de datos DATE proporcionadas por el SGBD MySQL.

Tabla 2.3. Funciones para la gestión de datos DATE proporcionadas por el SGBD MySQL

función	**Descripción**	**ejemplo**
ADDDATE (d, n)	Suma n días a la fecha de	SELECT DATE_ADD ('1998-01-02', 31); Devuelve: '1998-02-02'
ADDTIME (t1, t2)	Suma t1 y t2	SELECT ADDTIME ('2007-12-31 23: 59: 59.999999', '1 1: 1: 1.000002'); Devuelve: '2008-01-02 01: 01: 01.000001'
CONVERT_TZ (t, z1, z2)	Convierte t de una zona a otra	SELECT CONVERT_TZ ('2004-01-01 12:00:00', 'GMT', 'MET'); Devuelve: '2004-01-01 13:00:00'
CURDATE ()	Devuelve la fecha actual en formato 'AAAA-MM-DD'	SELECT CURDATE ();
CURRENT_DATE (), CURRENT_DATE	Sinónimos de CURDATE ()	
CURRENT_TIME (), CURRENT_TIME	Sinónimos de CURTIME ()	
CURRENT_TIMESTAMP (), CURRENT_TIMESTAMP	Sinónimos de NOW ()	
CURTIME ()	Devuelve la hora actual en formato 'HH: MM: SS'	SELECT CURTIME ();
DATE_FORMAT (d, f)	Devuelve la fecha de en formato f, en la que f está codificado siguiendo la tabla 2 .4	
DATE_SUB (d, n)	De manera similar a DATE_ADD, en este caso, resto de la fecha el valor del segundo parámetro	

DATE (d)	Obtiene la fecha de un dato que contiene fecha-hora	SELECT DATE ('2003-12-31 01:02:03'); Devuelve: '2003-12-31'
DATEDIFF (d1, d2)	Resto dos fechas	
DAY (d)	Sinónimo de DAYOFMONTH ()	SELECT DAY ('2007-02-03'); Devuelve: 3
DAYNAME (d)	Devuelve el nombre del día de la semana	SELECT DAYNAME ('2007-02-03'); Devuelve: 'Saturday'
DAYOFMONTH (d)	Devuelve el nombre del día del mes (0-31)	SELECT DAYOFMONTH ('2007-02-03'); Devuelve: 3
DAYOFWEEK (d)	Devuelve el número de orden del día de la semana a la que corresponde de (1 = Sunday, 2 = Monday, ..., 7 = Saturday)	SELECT DAYOFWEEK ('2007-02-03'); Devuelve: 7
DAYOFYEAR (d)	Devuelve el día del año	SELECT DAYOFYEAR ('2007-02-03'); Devuelve: 34
EXTRACT (u FROM d)	Extrae parte de la fecha	SELECT EXTRACT (YEAR FROM '2009-07-02'); Devuelve: 2009
FROM_DAYS (n)	Dado un número n lo convierte a tipo fecha	SELECT FROM_DAYS (730.669); Devuelve: '2007-07-03'
FROM_UNIXTIME ()	Formatear las fechas-horas de UNIX como una fecha	
GET_FORMAT ({DATE o TIME o DATETIME}, { 'EUR' o 'USA' o 'JIS' o 'ISO' o 'INTERNAL'})	Devuelve el formato de fecha válido solicitado. Se suele combinar con DATE_FORMAT	SELECT DATE_FORMAT ('2003-10-03', GET_FORMAT (DATE, 'EUR')); Devuelve: '03 .10.2003 '
HOUR (d)	Extrae la hora	SELECT HOUR ('10: 05: 03 '); Devuelve: 10
LAST_DAY (d)	Devuelve el último día del mes de la fecha de	SELECT LAST_DAY ('2003-02-05'); Devuelve: '2003-02-28'
Localtime (), localtime	Sinónimo de NOW ()	
LOCALTIMESTAMP, LOCALTIMESTAMP ()	Sinónimo de NOW ()	
MAKEDATE (d, día)	Crea una fecha a partir del año y el día del año	SELECT MAKEDATE (2.011,32); Devuelve: '2011-02-01'
MAKETIME MAKETIME (h, m, s)	Crea una hora a partir de las horas, minutos y segundos dados	SELECT MAKETIME (12,15,30); Devuelve '12: 15: 30 '
MICROSECOND (d)	Devuelve los microsegundos	
MINUTE (d)	Devuelve los minutos de de	SELECT MINUTE ('2008-02-03 10:05:03'); Devuelve: 5
MONTH (d)	Devuelve el mes de de	SELECT MONTH ('2008-02-03'); Devuelve: 2
MONTHNAME (d)	Devuelve el nombre del mes	SELECT MONTH ('2008-02-03'); Devuelve: 'February'
NOW ()	Devuelve la fecha y hora actuales	SELECT NOW ();
PERIOD_ADD (p, n)	Agregar n meses el período p	SELECT PERIOD_ADD (200.801,2); Devuelve: 200803
PERIOD_DIFF (p1, p2)	Devuelve el número de meses entre p1 y p2	
CUARTEL (d)	Devuelve el trimestre de	SELECT CUARTEL ('2008-04-01'); Devuelve: 2
SEC_TO_TIME (n)	Convertir n según el formato de horas	SELECT SEC_TO_TIME (2378);

	'HH: MM: SS'	Devuelve: '00: 39: 38 '
SECOND (t)	Devuelve el número de segundos de la hora determinada t (0-59)	SELECT SEC_TO_TIME (2378); Devuelve: '00: 39: 38 '
STR_TO_DATE (s, f)	Convertir una cadena sa una fecha en formato f	SELECT STR_TO_DATE ('May 1, 2013', '% M% d,% Y'); Devuelve: '2013-05-01'
SUBDATE (d, n)	Sinónimo de DATE_SUB ()	
SUBTIME (d1, d2)	Resto d1 y d2	
SYSDATE ()	Devuelve el día y hora en el momento de ejecutar la función	SELECT SYSDATE ();
TIME_FORMAT (d, f)	Formatear de según el formato f, de manera similar a DATE_FORMAT	
TIME_TO_SEC (t)	Devuelve t convertido en segundos	
TIME (d)	Extrae la hora de de	SELECT TIME ('2003-12-31 01:02:03'); Devuelve: '01: 02: 03 '
TIMEDIFF (t1, t2)	Resto t1 - t1	
TIMESTAMP (d)	Devuelve la fecha-hora de una fecha de	SELECT TIMESTAMP ('2003-12-31'); Devuelve: '2003-12-31 12:00:00'
TIMESTAMPADD (n, y, d)	Suma n intervalos ya una fecha de	SELECT TIMESTAMPADD (WEEK, 1, '2003-01-02'); Devuelve: '2003-01-09'
TIMESTAMPDIFF (y, d1, d2)	Resto d1 y d2, y obtiene el resultado según la unidad y	SELECT TIMESTAMPDIFF (MONTH, '2003-02-01', '2003-05-01'); Devuelve: 3
TO_DAYS (d)	Devuelve de convertido en días	SELECT TO_DAYS ('2007-10-07'); Devuelve: 733321
TO_SECONDS (d)	Devuelve de convertido en segundos	SELECT TO_SECONDS ('2009-11-29'); Devuelve: 63426672000
UNIX_TIMESTAMP ()	Devuelve la hora en formato UNIX	
UTC_DATE ()	Devuelve la fecha UTC	
UTC_TIME ()	Devuelve la hora UTC	
UTC_TIMESTAMP ()	Devuelve la fecha y hora UTC	
WEEK (d)	Devuelve el número de semana	SELECT WEEK ('2008-02-20'); Devuelve: 7
Weekday (d)	Devuelve el índice de día de la semana (0 = Monday, 1 = Martes, ... 6 = Sunday)	SELECT Weekday ('2007-11-06'); Devuelve: 1
WEEKOFYEAR (d)	Devuelve la semana del calendario de d (0-53)	SELECT WEEKOFYEAR ('2008-02-20'); Devuelve: 8
YEAR (d)	Devuelve el año	SELECT YEAR ('1987-01-01'); Devuelve: 1987
YEARWEEK (d)	Devuelve el año y semana de de	SELECT YEARWEEK ('1987-01-01'); Devuelve: 198653

En negrita se han destacado las funciones más utilizadas a la hora de manipular expresiones numéricas.

Para especificar las fechas y horas en diversos formatos, hay que seguir las notaciones descritas en la tabla 2 .4 .

Tabla 2.4. Notaciones para formatear las fechas en MySQL

especificador	Descripción
% a	Día de la 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)
% y	Día del mes numérico (0..31)
% f	Microsegundos (000000..999999)
% H	Hora (00..23)
% h	Hora (01..12)
% I	Hora (01..12)
% y	Minutos, numérico (00..59)
% j	Día del año (001..366)
% k	Hora (0..23)
% del	Hora (1..12)
% M	Nombre del 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	Según (00..59)
% s	Según (00..59)
% T	Hora, 24 horas (hh: mm: ss)
% U	Semana (00..53), en el que domingo es el primer día de la semana
% u	Semana (00..53), en el que el lunes es el primer día de la semana
% V	Semana (01..53), en el que domingo es el primer día de la semana; utilizado con% X
% v	Semana (01..53), en el que el lunes es el primer día de la semana; utilizado con% x
% W	Nombre del día de la semana (Sunday..Saturday)
% w	Día de la semana (0 = Sunday..6 = Saturday)
% X	Año para la semana en que domingo es el primer día de la semana, numérico, cuatro dígitos; usado con% V
% x	Año para la semana, en la que el lunes es el primer día de la semana, numérico, cuatro dígitos; usado con% v
% Y	Año, numérico, cuatro dígitos
% y	Año, numérico (dos dígitos)

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'
```

2.1.4. 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.

operador CASE

El operador CASEpermite obtener varios valores en función de unas condiciones o comparaciones previas. Hay dos maneras de utilizar el operador CASE:

- CASEvalor WHEN[valor_comparació] THENresultado [WHEN[valor_comparació] THENresultado ...] [ELSEresultado] **END**
- CASE WHEN[condición] THENresultado [WHEN[condición] THENresultado ...] [ELSEresultado]END

Estas dos formas se pueden utilizar como en los ejemplos:

```
SELECT CASE 1 WHEN 1 THEN 'one' WHEN 2 THEN 'two' ELSE 'more' END ;
Devuelve: 'one'

SELECT CASE WHEN 1 > 0 THEN 'true' ELSE 'false' END ;
Devuelve: 'true'
```

construcción IF

La construcción IFpermite, de manera similar a CASE, obtener valores diferenciados en función de una condición.

La sintaxis de la construcción IFes la siguiente:

```
IF ( condición , expresión 1 , expresión 2 )
```

Y funciona de la siguiente manera: si la condición es cierta, se devuelve el Expresión1, de otro modo, el Expresión2. Veamos el siguiente ejemplo:

```
SELECT IF ( 1 < 2 , 'si' , 'no' ) ;
Devuelve: 'si'
```

función IFNULL

La función IFNULL(expresión) es una función que devuelve el valor de la expresión si este no es nulo, y cero en caso de que se evaluara como nulo.

Esta función se puede utilizar para evitar que expresiones complejas en que hay valores nulos evalúen globalmente como nulo, si es posible asignarles el valor de cero en caso de nulo. Fíjese en ella con el siguiente ejemplo:

En el esquema *empresa* , se quieren mostrar los empleados (código, apellido, salario y comisión) acompañados de una columna que contenga la suma del salario y la comisión. En un principio, consideraríamos la sentencia siguiente:

```
SELECT emp_no AS  "Código" , apellido AS  "Empleado" , salario AS  "Salario" ,
      comisión AS  "Comisión" , salario + comision AS  "Suma"
FROM emp;
```

Y se obtendrá el siguiente resultado:

```
Código Empleado Salario Comisión Suma
------- ---------- --------- ------------ ------------ -----
7369 SÁNCHEZ 104000
7499 ARROYO 208000 39000 247000
7521 SALA 162500 65000 227500
7566 JIMÉNEZ 386750
7654 MARTÍN 162.500 182000 344500
7698 NEGRO 370500
7782 CEREZO 318500
7788 GIL 390000
7839 REY 650000
7844 TOVAR 195000 0 195000
7876 ALONSO 143000
7900 JIMENO 123500
7902 FERNÁNDEZ 390000
7934 MUÑOZ 169000

14 rows selected
```

El resultado no es el deseado, ya que todos los empleados tienen salario pero no todos tienen comisión asignada (para los que no tienen comisión hay un valor NULLen la columna correspondiente), y este hecho motiva que la suma sólo se calcule para quienes tienen comisión. Posiblemente esperaríamos que el SGBD considerara que un NULLen la comisión de los empleados es equivalente a cero. Se lo debemos especificar, tal como se ve en la siguiente sentencia:

```
SELECT emp_no AS  "Código" , apellido AS  "Empleado" , salario AS  "Salario" ,
      comisión AS  "Comisión" , salario + ifnull ( comisión , 0 )  AS  "Suma"
FROM emp;
```

Ahora sí que el resultado es el querido:

```
Código Empleado Salario Comisión Suma
------ ---------- --------- ------------ ------------ ---------
7369 SÁNCHEZ 104000 104000
7499 ARROYO 208000 39000 247000
7521 SALA 162500 65000 227500
7566 JIMÉNEZ 386750 386750
7654 MARTÍN 162.500 182000 344500
7698 NEGRO 370500 370500
7782 CEREZO 318500 318500
7788 GIL 390000 390000
7839 REY 650000 650000
7844 TOVAR 195000 0 195000
```

```
7876 ALONSO 143000 143000
7900 JIMENO 123500 123500
7902 FERNÁNDEZ 390000 390000
7934 MUÑOZ 169000 169000

14 rows selected
```

función NULLIFIES

La sintaxis que utiliza la función **NULLIFIES** es la siguiente:

```
NULLIFIES ( expresión 1 , expresión 2 )
```

Si el Expresión1 y el Expresión2 evalúan como iguales la función devuelve NULL; de otro modo, devuelve el Expresión1.

Veamos los siguientes ejemplos:

```
SELECT NULLIFIES ( 1 , 1 );
Devuelve: NULL

SELECT NULLIFIES ( 1 , 2 );
Devuelve:  1
```

Otras funciones de MySQL

MySQL proporciona otras funciones propias que enriquecen el lenguaje. Por ejemplo, en la versión del software 5.5 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, ORo XORo desplazamientos de bits, por ejemplo).

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

2.2. Clasificación de filas. Cláusula ORDER BY

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

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

Esta cláusula se añade detrás de la cláusula wheresi las hay, de modo que ampliamos la sintaxis de la sentencia SELECT:

```
SELECT < expresión / columna >, < expresión / columna >, ...
FROM < tabla >, < tabla >, ...
[ WHERE < condició_de_recerca > ]
[ ORDER BY < expresión / columna > [ ASC | DESC ] , < expresión / columna > [ ASC | DESC ] , ... ] ;
```

Como se puede ver, la cláusula order bypermite 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 byque 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 lo 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",
  comisión 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 para las filas que tienen NULL en la columna comisión.

2.3. Exclusión de filas repetidas. opción DISTINCT

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

La opción distinctacompañando la cláusula selectpermite 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 <condició_de_recerca>]
[ORDER BY <expresión / columna> [ASC | DESC], <expresión / columna> [ASC | DESC], ...];
```

La utilización de la opción distinctimplica que el SGBD ejecute obligatoriamente una order bysobre 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

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

2.4. 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 bypermite agrupar las filas resultado de las cláusulas select, fromy wheresegún una o más de las columnas seleccionadas.

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

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

```
SELECT [ DISTINCT ] < expresión / columna >, < expresión / columna >, ...
FROM < tabla >, < tabla >, ...
[ WHERE < condició_de_recerca > ]
[ GROUP BY < alias / columna >, < alias / columna >, ... ]
[ HAVING < condició_sobre_grups > ]
[ ORDER BY < expresión / columna > [ ASC | DESC ] , < expresión / columna > [ ASC | DESC ] , ... ] ;
```

Recuerde que los elementos que se ponen entre corchetes ([]) indican opcionalidad.

La tabla 2 .5 nos muestra las funciones de agrupamiento más importantes que se pueden utilizar en las sentencias SELECTde selección de conjuntos.

Tabla 2.5. Funciones de agrupamiento más importantes que se pueden utilizar en las sentencias SELECT de agrupamiento de filas

función	Descripción	ejemplos
AVG (n)	Devuelve el valor medio de la columna n ignorando los valores nulos.	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.	COUNT (dept_no) (sobre la mesa de empleados) cuenta cuántos empleados están asignados a algún departamento.
MAX (expr)	Devuelve el valor máximo de expr.	MAX (salario) devuelve el salario más alto.
MIN (expr)	Devuelve el valor mínimo de expr.	MIN (salario) devuelve el salario más bajo.
STDDEV (expr)	Devuelve la desviación típica de expr sin tener en cuenta los valores nulos.	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.	SUM (salario) devuelve la suma de todos los salarios.
Variance (expr)	Devuelve la varianza de expr sin tener en cuenta los valores nulos.	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 distinctpara indicar que se evalúe sobre los valores distintos de la expresión, o de la opción allpara 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 byo la cláusula havingo una función de agrupamiento; es decir, una sentencia SELECTpuede 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 selectde una sentencia SELECTde selección de conjuntos deben aparecer obligatoriamente en la cláusula group byde la sentencia. Ahora bien, no todas las columnas y expresiones de la cláusula group bydeben 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 el 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;
```

2.5. Unión, intersección y diferencia de sentencias SELECT

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

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

2.5.1. Unión de sentencias SELECT

El lenguaje SQL proporciona el operador unionpara combinar todas las filas del resultado de una sentencia SELECTcon 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:

```
sentència_select_sense_order_by
UNION
sentència_select_sense_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
** UNION **
SELECT nombre, función, empleat_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 el siguiente:

```
Hospital Oficio Código 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.

18 rows selected
```

2.5.2. 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:

```
sentència_select_sense_order_by
** INTERSECT **
sentència_select_sense_order_by
[ ORDER  BY  ... ]
```

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

```
sentència_select_sense_order_by
** minus **
sentència_select_sense_order_by
[ ORDER  BY  ... ]
```

2.6. Combinaciones entre tablas

La cláusula fromefectúa 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 RS que verifican AZ en qué 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 *outer-join* y tenemos dos posibilidades (left o right) según donde se encuentre (izquierda o derecha) la mesa por 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 RS 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 RS 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.

2.6.1. Combinaciones entre tablas según la norma SQL-87 (SQL-ISO)

El lenguaje SQL-86, ratificado por la ISO en 1987, establecía que los diferentes tipos de combinaciones se podían alcanzar añadiendo, en la cláusula where, los filtros correspondientes a las combinaciones entre las columnas de las tablas que se han de combinar.

Ejemplo de combinación entre dos tablas según el SQL-87

En el esquema *empresa* , se quieren mostrar los empleados (código y apellido) junto con el código y nombre del departamento al que pertenecen.

La instrucción para alcanzar el objetivo es esta:

```
SELECT emp . emp_no AS  "Código empleado" , emp . apellido AS  "Empleado" ,
  emp . dept_no AS  "Código departamento" , dnom AS  "Descripción"
FROM emp , dept
WHERE emp . dept_no = dept . dept_no;
```

Fijémonos en que el acceso a la tabla DEPT es necesario para conseguir el nombre del departamento. Tengamos en cuenta, también, que, como que el campo deptno de la tabla EMP no permite valores NULL, todos los empleados tendrán valor en este campo y, debido a la integridad referencial, no pueden tener un valor que no sea en la tabla DPTO. Así pues, una vez ejecutado el filtro de la cláusula where *, todas las filas de la tabla EMP estarán combinadas con una fila de la tabla DEPT.

El resultado obtenido es el siguiente:

```
Código empleado Empleado Código departamento Descripción
--------------- -------- ------------------- -----------
7369 SANCHEZ 20 INVESTIGACIÓN
7499 ARROYO 30 VENTAS
7521 SALA 30 VENTAS
7566 JIMENEZ 20 INVESTIGACIÓN
7654 MARTIN 30 VENTAS
7698 NEGRO 30 VENTAS
7782 CEREZO 10 CONTABILIDAD
7788 GIL 20 INVESTIGACIÓN
7839 REY 10 CONTABILIDAD
7844 TOVAR 30 VENTAS
7876 ALONSO 20 INVESTIGACIÓN
7900 JIMENO 30 VENTAS
7902 FERNANDEZ 20 INVESTIGACIÓN
7934 MUÑOZ 10 CONTABILIDAD

14 rows selected
```

Aplicando los filtros adecuados en la cláusula **where** , conseguimos implementar el *join* , el *equi-join* y el *natural-join* , pero no llegamos a poder implementar los *outer-join* , ya que el producto cartesiano no puede "inventar" filas con valores nulos .

En 1987, el estándar SQL-ISO propuso (quizás presionado por Oracle, que ya tenía implementada la solución) la utilización de una marca en la condición de combinación entre las columnas de las tablas R y S que fuera necesario combinar que indicara la necesidad de hacer aparecer todas las filas de una tabla (por ejemplo R), a pesar de que para la columna de combinación no hubiera correspondencia en la otra tabla (S), haciendo aparecer valores nulos en las columnas de la tabla S indicadas en la cláusula select. La marca adoptada fue el símbolo (+) pegado a la derecha de la mesa (S) para la que hay que generar valores nulos.

Es decir, un *left-outer-join* de la tabla I con la tabla S según las columnas A (tabla I) y Z (tabla S) respectivas, que se escribiría como **R [A = Z] S** , se convertiría en SQL en una condición como la siguiente:

```
WHERE R . A = S . Z ( + )
```

Del mismo modo, un *right-outer-join* de la tabla I con la tabla S según las columnas A (tabla I) y Z (tabla S) respectivas, que en álgebra relacional se escribiría como **R [A = Z] S** , se convertiría en SQL en una condición como la siguiente:

```
WHERE R.A(+) = S.Z
```

El estándar SQL-ISO no proporciona ninguna instrucción específica para alcanzar un *full-outer-join* entre dos tablas y **no** está permitido escribir lo siguiente:

```
WHERE R.A(+) = S.Z(+)
```

La solución en SQL-ISO para conseguir un *full-outer-join* pasa por una operación unión entre una sentencia SELECTcon el *left-outer-join* y una sentencia SELECT con el *right-outer-join* .

MySQL no soporta este estándar de implementación de las *left-outer-join* , *right-outer-join* y *full-outer-join* .

2.6.2. Combinaciones entre tablas según la norma SQL-92

No todos los SGBD siguieron la modalidad de la marca (+) para implementar las dos variantes de *outer-join* . Y es comprensible, ya que hay una manera más comprensible de explicar el porqué de las combinaciones entre diferentes tablas.

A menudo, en la combinación entre dos tablas hay una tabla que se puede considerar la tabla principal (donde tenemos que ir a buscar la información) y otra tabla que se puede considerar secundaria (donde tenemos que ir a buscar información que complemente la información buscada en la tabla principal).

Para conseguir nuestro propósito, desde la revisión del 92 (SQL-92), el lenguaje SQL facilita las operaciones *join* en la cláusula fromindicando la condición de combinación, la que ya no se deberá indicar (excepto en un caso) dentro de la cláusula where.

Es decir, pasamos de una sentencia SELECTque incluye una parte similar a:

```
...
FROM tabla1 , Tabla2
WHERE  < condición combinación entre tabla1 y Tabla2 >
...
```

a una sentencia SELECTen que la condición de combinación entre tablas ya no se indica en la cláusula where, sino que acompaña la cláusula from:

```
...
FROM tabla1 [ INNER | LEFT | RIGHT ] JOIN Tabla2
ON  < condición combinación entre tabla1 y Tabla2 >
WHERE  ...
```

Como se ve en la sintaxis anterior, hay diferentes opciones de *join* : inner, lefty right.

Los SGBD relacionales actuales (MySQL, Oracle, SQL Server, PostgreSQL, MS-Access ...) incorporan las combinaciones inner, lefty rightcon las que se pueden conseguir todos los tipos de combinaciones entre tablas. Hay otras opciones que algunos SGBD también soportan, pero no siempre siguiendo una sintaxis idéntica. La sintaxis aquí presentada es la proporcionada por el SGBD MySQL a partir de la versión 5.

La combinación inneres la más común y, de hecho, es la que se ejecuta si no se indica ninguna de las opciones. Se denomina *combinación interna* y combina filas de dos tablas siempre que haya valores coincidentes en el campo o campos de combinación.

Ejemplo de combinación inner entre dos tablas

En el esquema *empresa* , se quieren mostrar los empleados (código y apellido) junto con el código y nombre del departamento al que pertenecen.

```
SELECT emp . emp_no AS "Código empleado" , emp . apellido AS "Empleado" ,
  emp . dept_no AS "Código departamento" , dnom AS "Descripción"
FROM emp INNER JOIN dept ON emp . dept_no = dept . dept_no;
```

Fijémonos en que la columna correspondiente al código de departamento sólo aparece una vez, ya que sólo la hemos indicado una vez en la cláusula select. Recordemos también que no es obligatorio utilizar la palabra inner.

Las combinaciones **left join** , **right join** y **hoja join** (llamadas *combinaciones externas*) son las opciones que proporciona el SQL-92 para alcanzar las diversas opciones de *outer-join* .

La combinación **left join** permite combinar todas las filas de la tabla de la izquierda de *join* con las filas con valores coincidentes de la tabla de la derecha, y proporciona valores nulos para las columnas de la tabla de la derecha cuando no hay filas con valores coincidentes.

La combinación **right join** permite combinar todas las filas de la tabla de la derecha del *join* con las filas con valores coincidentes de la tabla de la izquierda, y proporciona valores nulos para las columnas de la tabla de la izquierda cuando no hay filas con valores coincidentes.

La combinación **hoja join** es la unión del **left join** y **right join** eliminando la duplicidad de filas debido a las filas de las dos tablas que tienen valores coincidentes.

Ejemplo 1 de right-outer-join y left-outer-join entre tablas según el SQL-92

En el esquema *empresa* , se quieren mostrar todos los departamentos (código y descripción) acompañados del salario más alto de sus empleados.

```
SELECT de . dept_no AS "Código" , dnom AS "Departamento" ,
   MAX ( salario ) AS "Mayor salario"
FROM dept de LEFT JOIN emp y ON de . dept_no = e . dept_no
GROUP BY de . dept_no , dnom
ORDER BY 1 ;
```

El resultado obtenido es el siguiente:

```
Código Departamento Salario más alto
------ ------------ --------------------
10     CONTABILIDAD  650000
20     INVESTIGACIÓN 390000
30     VENTAS        370500
```

```
40        PRODUCCIÓN

4 rows selected
```

En la sentencia anterior hemos utilizado un left join, el cual se puede convertir en un right join si cambiamos el orden de las tablas en la cláusula from:

```
SELECT de . dept_no AS "Código" , dnom AS "Departamento" ,
  MAX ( salario ) AS "Mayor salario"
FROM emp y RIGHT JOIN dept de ON y . dept_no = d . dept_no
GROUP BY de . dept_no , dnom
ORDER BY 1 ;
```

El resultado obtenido en este caso es el mismo que en la sentencia anterior.

Ejemplo 2 de right-outer-join y left-outer-join entre tablas según el SQL-92

Se quieren mostrar, en el esquema *empresa* , todos los empleados acompañados de los clientes de quienes son representantes.

La instrucción para alcanzar el objetivo es esta:

```
SELECT emp_no AS "Código" , apellido AS "Empleado" , client_cod AS "Cliente" , nombre AS "Razón s
FROM emp LEFT JOIN cliente DONDE emp_no = repr_cod
ORDER BY 1 , 2 ;
```

O también:

```
SELECT emp_no AS "Código" , apellido AS "Empleado" , client_cod AS "Cliente" , nombre AS "Razón s
FROM cliente RIGHT JOIN emp DONDE repr_cod = emp_no
ORDER BY 1 , 2 ;
```

El resultado que se obtiene, en ambos casos, es este:

```
Código Empleado Cliente Razón social
--------- --------- ---------- -------------------- --------------------
7369 SANCHEZ
7499 ARROYO 107 WOMEN SPORTS
7499 ARROYO 104 EVERY MOUNTAIN
7521 SALA 101 TKB SPORT SHOP
7521 SALA 103 JUSTO TENIS

7521 SALA 106 SHAPE UP
7566 JIMENEZ
7654 MARTIN 102 VOLLYRITE
7698 NEGRO
7782 CEREZO
7788 GIL
7839 REY
7844 TOVAR 108 NORTH WOODS HEALTH AND FITNESS SUPPLY CENTER
7844 TOVAR 105 K + T SPORTS
7844 TOVAR 100 JOCKSPORTS
7876 ALONSO
7900 JIMENO
7902 FERNANDEZ
7934 MUÑOZ

19 rows selected
```

Fijémonos que, para asegurar la aparición de todos los empleados, hay que utilizar un outer-join, ya que de lo contrario los empleados que no tienen asignado ningún cliente no aparecerían.

Ejemplo 3 de right-outer-join y left-outer-join entre tablas según el SQL-92

En el esquema *empresa* , se quieren mostrar todos los clientes acompañados del empleado que tienen como representante.

La instrucción para alcanzar el objetivo es esta:

```
SELECT client_cod AS "Cliente" , nombre AS "Razón social" , emp_no AS "Código" , apellido AS "Emp
FROM cliente LEFT JOIN emp DONDE repr_cod = emp_no;
```

O también:

```
SELECT client_cod AS "Cliente" , nombre AS "Razón social" , emp_no AS "Código" , apellido AS "Emp
FROM emp RIGHT JOIN cliente DONDE emp_no = repr_cod;
```

El resultado que se obtiene es este:

```
Cliente Razón social Código Empleado
--------- ---------- ----- ---------- ---------
107 WOMEN SPORTS 7499 ARROYO
104 EVERY MOUNTAIN 7499 ARROYO
106 SHAPE UP 7521 SALA
103 JUSTO TENIS 7521 SALA
101 TKB SPORT SHOP 7521 SALA
102 VOLLYRITE 7654 MARTIN
108 NORTH WOODS HEALTH AND FITNESS SUPPLY CENTER 7844 TOVAR
105 K + T SPORTS 7844 TOVAR
100 JOCKSPORTS 7844 TOVAR
109 SPRINGFIELD NUCLEAR POWER PLANT

10 rows selected
```

Fijémonos que, para asegurar la aparición de todos los clientes, es necesario utilizar un outer-join, pues de lo contrario los clientes que no tienen asignado representante no aparecerían.

Ejemplo 4 de ojo-outer-join entre tablas según el SQL-92

En el esquema *empresa* , se quieren mostrar todos los clientes y todos los empleados relacionando cada cliente con su representante (y, de paso, cada empleado con sus clientes).

La instrucción para alcanzar el objetivo es esta:

```
SELECT client_cod AS "Cliente" , nombre AS "Razón social" , emp_no AS "Código" , apellido AS "Emp
FROM cliente LEFT JOIN emp DONDE emp_no = repr_cod
UNION
SELECT client_cod AS "Cliente" , nombre AS "Razón social" , emp_no AS "Código" , apellido AS "Emp
FROM cliente RIGHT JOIN emp DONDE emp_no = repr_cod;
```

El resultado obtenido es el siguiente:

```
Cliente Razón social Código Empleado

--------- ------------------------------------ ----- ---------- ----------
100 JOCKSPORTS 7844 TOVAR
101 TKB SPORT SHOP 7521 SALA
102 VOLLYRITE 7654 MARTIN
103 JUSTO TENIS 7521 SALA
104 EVERY MOUNTAIN 7499 ARROYO
105 K + T SPORTS 7844 TOVAR
106 SHAPE UP 7521 SALA
107 WOMEN SPORTS 7499 ARROYO
108 NORTH WOODS HEALTH AND FITNESS SUPPLY CENTER 7844 TOVAR
109 SPRINGFIELD NUCLEAR POWER PLANT
                                        7369 SANCHEZ
                                        7566 JIMENEZ
                                        7698 NEGRO
                                        7782 CEREZO
                                        7788 GIL
                                        7839 REY
                                        7876 ALONSO
                                        7900 JIMENO
                                        7902 FERNANDEZ
                                        7934 MUÑOZ
20 rows selected
```

El lenguaje SQL proporciona dos simplificaciones en la escritura de las combinaciones *join* que necesitan la opción on, para los casos en que las columnas que hay que combinar tengan coincidencia de nombres:

- Si la combinación se quiere efectuar para todas las columnas que tengan nombres coincidentes en las tablas que se han de combinar, disponemos de las combinaciones **natural join** . En este caso, la palabra **natural** ante el tipo de combinación *join* provoca que se efectúe la combinación entre las tablas para todas las columnas que tienen coincidencia de nombre, sin tener que indicar la condición de combinación. El resultado de las *natural join* proporciona una única columna para las columnas de las tablas combinadas que tienen el mismo nombre y, por tanto, en caso de tener que hacer referencia a esta columna en alguna cláusula de la sentenciaSELECT, No se debe indicar el nombre de la tabla a la que pertenece, ya que pertenece simultáneamente a diferentes tablas.
- Si la combinación se quiere efectuar para algunas de las columnas que tengan nombres coincidentes en las tablas que hay que combinar, disponemos de las combinaciones *join* con la opción using (col1, col2 ...). En este caso, la opción using (col1, col2 ...)provoca que se efectúe la combinación entre las tablas para las columnas indicadas, sin tener que indicar la condición de combinación. Como en los *natural join* , también da como resultado una única columna para las columnas coincidentes.

Ejemplo de simplificación de una combinación inner join

La sentencia siguiente, correspondiente al esquema *empresa* , muestra los empleados (código y apellido) junto con el código y nombre del departamento al que pertenecen.

```
SELECT emp . emp_no AS "Código empleado" , emp . apellido AS "Empleado" ,
  emp . dept_no AS "Código departamento" , dnom AS "Descripción"
FROM emp INNER JOIN dept ON emp . dept_no = dept . dept_no;
```

o bien, lo que es lo mismo:

```
SELECT emp . emp_no AS "Código empleado" , emp . apellido AS "Empleado" ,
  emp . dept_no AS "Código departamento" , dnom AS "Descripción"
FROM emp JOIN dept ON emp . dept_no = dept . dept_no;
```

Como en las tablas EMPy DEPThay coincidencia de nombre para la columna de combinación deptno y no hay otras columnas con nombres coincidentes, podemos utilizar un natural join /:

```
SELECT emp . emp_no AS "Código empleado" , emp . apellido AS "Empleado" ,
  dept_no AS "Código departamento" , dnom AS "Descripción"
FROM emp NATURAL JOIN dept;
```

Fijémonos en que, al visualizar la columna deptno, en la cláusula select se ha tenido que suprimir la referencia a la tabla a la que pertenece, ya que el natural join * sólo proporciona una de las dos columnas con coincidencia de nombres. Pero también se habría podido utilizar la opción using:

```
SELECT emp . emp_no AS "Código empleado" , emp . apellido AS "Empleado" ,
  dept_no AS "Código departamento" , dnom AS "Descripción"
FROM emp INNER JOIN dept USING ( dept_no ) ;
```

Asimismo, al tratarse de una combinación inner join, nos podemos ahorrar la palabra inner, y tendríamos lo siguiente:

```
SELECT emp . emp_no AS "Código empleado" , emp . apellido AS "Empleado" ,
  dept_no AS "Código departamento" , dnom AS "Descripción"
FROM emp JOIN dept USING ( dept_no ) ;
```

Ejemplo de simplificación de combinaciones left join y right join

Las sentencias siguientes, correspondientes al esquema *empresa* , muestran todos los departamentos (código y descripción) acompañados del salario más alto de sus empleados.

```
SELECT de . dept_no AS "Código" , dnom AS "Departamento" ,
  MAX ( salario ) AS "Salario más alto"
FROM dept de LEFT JOIN emp y ON de . dept_no = e . dept_no
GROUP BY de . dept_no , dnom
ORDER BY 1 ;

SELECT de . dept_no AS "Código" , dnom AS "Departamento" ,
  MAX ( salario ) AS "Salario más alto"
FROM emp y RIGHT JOIN dept de ON y . dept_no = d . dept_no
GROUP BY de . dept_no , dnom
ORDER BY 1 ;
```

Dado que la columna de combinación tiene el mismo nombre y no hay otras columnas con coincidencia de nombres, habríamos podido emplear un natural join:

```
SELECT dept_no AS "Código", dnom AS "Departamento",
  MAX(salario) AS "Salario más alto"
FROM dept NATURAL LEFT JOIN emp y
GROUP BY dept_no, dnom
ORDER BY 1;

SELECT dept_no AS "Código", dnom AS "Departamento",
  MAX(salario) AS "Salario más alto"
FROM emp e NATURAL RIGHT JOIN dept de
GROUP BY dept_no, dnom
ORDER BY 1;
```

Y también, utilizando la opción using:

```
SELECT dept_no AS "Código", dnom AS "Departamento",
  MAX(salario) AS "Salario más alto"
FROM dept LEFT JOIN emp y USING (dept_no)
GROUP BY dept_no, dnom
ORDER BY 1;

SELECT dept_no AS "Código", dnom AS "Departamento",
  MAX(salario) AS "Salario más alto"
FROM emp y RIGHT JOIN dept de USING (dept_no)
GROUP BY dept_no, dnom
ORDER BY 1;
```

2.7. 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 wherede 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"
```

```
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 fromde la subconsulta, a menos que vayan acompañadas de alias que las identifiquen como columnas de una mesa 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 wherecomo parte derecha de operaciones de comparaciones en las que intervienen los operadores:

```
=, != , <, <=,>,>=, [NOT] IN, %%<op>%% año 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 en den 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 AÑO y EXISTS

En el esquema *empresa* , se pide mostrar los nombres y oficios de los empleados del departamento 20 cuyo trabajo 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 = AÑO ( SELECT oficio
      FROM emp
      WHERE dept_no = AÑO ( SELECT dept_no
           FROM dept
           WHERE UPPER ( dnom ) = '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 dnom 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 = AÑO ( SELECT oficio
      FROM emp
      WHERE dept_no =  ( SELECT dept_no
          FROM dept
          WHERE UPPER ( dnom ) = '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 ( dnom ) = 'VENTAS'
     AND oficio = e . oficio
    ) ;
```

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'
 ) ;
```