

3

Realización de consultas

OBJETIVOS DEL CAPÍTULO

- ✓ Aprender a realizar consultas de diversa complejidad sobre las tablas de una base de datos.
- ✓ Conocer las funciones principales que incorpora MySQL.
- ✓ Conocer el funcionamiento de las expresiones regulares en MySQL.

Una de las principales ventajas de las bases de datos es la potencialidad que ofrecen a la hora de hacer consultas de la información que contienen. Esta potencia es tanto mayor cuanto mejor y más óptimo es el diseño.

En este capítulo aprenderemos las técnicas para diseñar consultas utilizando el lenguaje MDL de SQL. Empezaremos con consultas básicas sobre una sola tabla para después pasar a consultas más complejas que incluyen varias tablas, funciones de agrupación, funciones propias de MySQL, expresiones regulares, etc.



Para la realización de todos los ejemplos y ejercicios de este capítulo se debe instalar el cliente y servidor MySQL, así como las bases de datos incluidas en la Web como material complementario. El proceso, siguiendo el asistente, es muy sencillo y está perfectamente documentado en su web. Concretamente en español se puede encontrar en la dirección (es posible que en el momento de publicar este libro haya podido cambiar):

<http://dev.mysql.com/doc/refman/5.0/es/mysql-install-wizard-introduction.html>

Para idioma inglés (en una versión más fiable y actualizada) en:

<http://dev.mysql.com/doc/refman/5.6/en/windows-installation.html>

Para poder usar la interfaz gráfica MySQL Workbench (que nos facilitará enormemente el trabajo tanto en este capítulo como en el de programación) podemos encontrar la documentación, solamente en inglés, en:

<http://dev.mysql.com/doc/workbench/en/wb-installing-windows.html>

3.1 INTRODUCCIÓN SENTENCIA SELECT EN MYSQL

El MDL (*Manipulation Data Language*) es la parte de SQL dedicada a la manipulación de los datos, es decir, inserción, borrado, modificación y consulta de los mismos.

En este capítulo nos centraremos en la parte de consultas. Existen muchos tipos de consultas de diversa complejidad. Empezaremos con las más simples. Usaremos como base de datos para los ejemplos una parte de la base de datos *liga* comentada en el capítulo anterior. Para facilitar la comprensión de las consultas incluimos también el contenido de las tablas.

Cada comando SQL se compone de cláusulas. Para consultas la cláusula principal es *SELECT*. Su sintaxis es compleja así que la veremos poco a poco.

De momento estudiaremos consultas simples según la sintaxis siguiente:

Formato básico de la sentencia *SELECT*

```
SELECT [ALL | DISTINCT | DISTINCTROW]
    expresion_select, ...
    FROM referencias_de_tablas
    WHERE condiciones
    [GROUP BY
        [ASC | DESC], ... [WITH ROLLUP] ]
    [HAVING condiciones]
    [ORDER BY
        [ASC | DESC] ,...]
    [LIMIT ]
```

Antes de comentar el significado de cada cláusula recordamos la nomenclatura para describir los comandos:

- [] Indica opciones. Todo lo que va entre corchetes es opcional.
- {} Indica opciones obligatorias. Debe elegirse una de entre varias opciones.
- | Sirve para distinguir entre opciones.

Así, en el caso de la sintaxis de *SELECT* tendríamos los siguientes significados para cada cláusula:

- *expression_select*: indica una expresión, es decir, una operación sobre valores o campos de las tablas subyacentes.
- *ALL*: indica todos los valores de la expresión.
- *DISTINCT*: para mostrar solo valores distintos.
- *DISTINCTROW*: para mostrar filas distintas.
- *FROM*: indica la o las tablas afectadas en la consulta.
- *WHERE*: permite incluir condiciones sobre las filas de las tablas.
- *GROUP BY*: sirve para agrupar registros según uno o varios campos.
- *WITH ROLLUP*: para hacer resúmenes de datos.
- *HAVING*: permite añadir condiciones sobre agrupaciones de registros.
- *ORDER BY*: muestra los datos ordenados según uno o varios campos.
- *ASC|DESC*: indica el orden ascendente o descendente respectivamente.
- *LIMIT*: indica el número de registros a mostrar.

Veremos ejemplos de todos ellos a continuación.

Tras la cláusula *SELECT* se escriben expresiones –habitualmente nombres de columnas, o expresiones en las que figuren nombres de columnas– referentes a la tabla cuyo nombre aparece en la cláusula *FROM*. Si en lugar de nombres de columnas ponemos un * equivale a escribir los nombres de todas las columnas de la tabla.

El resultado de la ejecución de una sentencia *SELECT* es siempre otra tabla. Las columnas de la tabla resultante serán las que figuren enumeradas tras la cláusula *SELECT*, y en el mismo orden.

3.2 BASE DE DATOS DE EJEMPLO

Antes de estudiar la sintaxis en detalle y con ejemplos recordamos la definición de las tablas y su contenido:

Definición de las tablas de la base liga

```
equipo(id_equipo, nombre_equipo, ciudad, web_oficial, puntos)
```

Donde:

- *id_equipo*: es un valor numérico que representa el identificador (clave principal) de cada equipo.
- *nombre_equipo*: es un campo de tipo cadena que representa el nombre de cada equipo.
- *ciudad*: es un dato de tipo cadena para indicar la ciudad de origen de cada equipo.
- *web_oficial*: es de tipo cadena para indicar la web oficial, en caso de haberla, del equipo.

```
jugador(id_jugador, nombre, apellido, id_capitan, posicion, fecha_alta, salario_bruto,  
equipo)
```

Donde:

- *id_jugador*: es el campo de tipo numérico para identificar a cada jugador de la liga.
- *Nombre y apellido*: son campos de tipo cadena que indica el nombre del jugador.
- *id_capitan*: es el identificador del capitán del jugador (que a su vez también es jugador).
- *posición*: campo numérico para indicar el puesto del jugador (alero, pívot, base o escolta).
- *fecha_alta*: campo de tipo fecha-hora para el día que el jugador se dio de alta en el equipo.
- *salario_bruto*: campo numérico para el salario bruto anual de cada jugador.
- *equipo*: campo numérico para indicar el equipo del jugador.

```
partido(elocal, evisitante, resultado, fecha, arbitro)
```

Donde:

- *evisitante y elocal*: son campos numéricos que representa al identificador del equipo local.
- *resultado*: es un campo de tipo carácter para representar el resultado de un encuentro.
- *fecha*: es de tipo fecha-hora para indicar el momento del partido.
- *arbitro*: es un campo de tipo numérico para indicar el identificador del árbitro.

Contenido de las tablas de la base liga

Tabla 3.1 Datos tabla jugadores

id_jugador	nombre	apellido	puesto	id_capitan	fecha_alta	salario	num_equipo	altura
1	Juan Carlos	Navarro	Escolta	1	10/01/2010	130.000	1	
2	Felipe	Reyes	Pivot	2	20/02/2009	120.000	2	2,04
3	Victor	Claver	Alero	3	08/03/2009	90.000	3	2,08
4	Rafa	Martinez	Escolta	4	11/11/2010	51.000	3	1,91
5	Fernando	San Emeterio	Alero	6	22/09/2008	60.000	4	1,99
6	Mirza	Teletovic	Pivot	6	13/05/2010	70.000	4	2,06
7	Sergio	Llull	Escolta	2	29/10/2011	100.000	2	1,90
8	Victor	Sada	Base	1	01/01/2012	80.000	1	1,92
9	Carlos	Suarez	Alero	2	19/02/2011	60.000	2	2,03
10	Xavi	Rey	Pivot	14	12/10/2008	95.000	5	2,09
11	Carlos	Cabezas	Base	13	21/01/2012	105.000	6	1,86
12	Pablo	Aguilar	Alero	13	14/06/2011	47.000	6	2,03
13	Rafa	Hettsheimeir	Pivot	13	15/04/2008	53.000	6	2,08
14	Sitapha	Savané	Pivot	14	27/07/2011	60.000	5	2,01

Tabla 3.2 Datos tabla equipos

1	Regal Barcelona	Barcelona	http://www.fcbarcelona.com/web/index_idiomes.html	10
2	Real Madrid	Madrid	http://www.realmadrid.com/cs/Satellite/es/1193040472450/SubhomeEquipo/Baloncesto.htm	9
3	P.E. Valencia	Valencia	http://www.valenciabasket.com/	11
4	Caja Laboral	Vitoria	http://www.baskonia.com/prehomes/prehomes.asp?id_prehomes=69	22
5	Gran Canaria	Las Palmas	http://www.acb.com/club.php?id=CLA	14
6	CAI Zaragoza	Zaragoza	http://basketzaragoza.net/	23

Tabla 3.3 Datos tabla partidos

elocal	evisitante	resultado	fecha	arbitro
1	2	100-100	10/10/2011	4
2	3	90-91	17/11/2011	5
3	4	88-77	23/11/2011	6
1	6	66-78	30/11/2011	6
2	4	90-90	12/01/2012	7
4	5	79-83	19/01/2012	3
3	6	91-88	22/02/2012	3
5	4	90-66	27/04/2012	2
6	5	110-70	30/05/2012	1

3.3 CONSULTAS BÁSICAS

La consulta más simple que podemos hacer es la que nos devuelve todos los datos de una tabla.



EJEMPLO 3.1

Obtener todos los datos de todos los equipos:

```
SELECT * FROM equipo;
```

El `*` sirve como carácter comodín. Es equivalente a incluir todas las columnas de la tabla indicada en la cláusula `FROM`.

Los nombres de las columnas de la tabla resultante serán los mismos que los de las columnas de la tabla de la que proceden, salvo en el caso de que se usen alias o expresiones. Puede asignarse un nombre, o cambiar el nombre de la columna escribiendo el nuevo nombre a continuación del nombre de la columna o de la expresión que genera la columna en la sentencia `SELECT`. Si el nombre tiene espacios en blanco, debe escribirse entre comillas.

También puede usarse la palabra reservada `AS` para indicar un alias para el nombre de la columna.



EJEMPLO 3.2

Obtener el nombre de todos los jugadores:

```
SELECT nombre "NOMBRE JUGADOR" FROM jugador;
```

Que es equivalente a:

```
SELECT nombre AS "NOMBRE JUGADOR" FROM jugador;
```

- También pueden usarse alias para las tablas, ya que cuando las consultas se hacen más largas y complejas se facilita su escritura. Para ello simplemente ponemos el alias al lado del nombre de la tabla en la cláusula *FROM*. De este modo, para hacer referencia a un campo conviene poner primero el alias de la tabla (sobre todo en consultas *multitable* como veremos más adelante). El ejemplo anterior quedaría:

```
SELECT j.nombre AS "NOMBRE JUGADOR" FROM jugador j;
```

3.3.1 CLÁUSULA ORDER BY

La columna especificada en la cláusula *ORDER BY* será, habitualmente, una de las que aparecen en la cláusula *SELECT*.

Si no se especifica *ORDER BY*, las filas se devuelven en cualquier orden.

También podemos clasificar el resultado por más de una columna, para ello se escriben sus nombres o sus números separados por comas. Si se especifica más de una columna, se clasifica por la primera y después, para cada valor clasificado de la primera, por la segunda y así sucesivamente.



EJEMPLO 3.3

Seleccionar los nombres, apellido y posición de todos los jugadores ordenados por posición:

```
SELECT nombre, apellido, posicion FROM jugador ORDER BY posicion;
```



EJEMPLO 3.4

Seleccionar el nombre, equipo y posición de los jugadores ordenados por equipo y posición:

```
SELECT nombre, equipo, posicion  
FROM jugador  
ORDER BY equipo, posicion;
```

Para ordenar podemos especificar un número en lugar del nombre de una columna. Para ello, escribimos el nombre de la columna cuya posición en la cláusula *SELECT* corresponde a ese número. Por ejemplo, si se especifica *ORDER BY 2* equivale a escribir el nombre de la segunda columna especificada en la cláusula *SELECT*. Así, la consulta del ejemplo anterior quedaría:



EJEMPLO 3.5

Seleccionar los datos de los jugadores ordenados por equipo y posición:

```
SELECT nombre, equipo, posicion  
FROM jugador  
ORDER BY 2, 3;
```

Por omisión, la clasificación se realiza en orden creciente (ASC). Si se desea en orden decreciente se debe escribir *DESC* detrás del nombre o el número de la columna para la que se desea la clasificación decreciente.



EJEMPLO 3.6

Mostrar el nombre, equipo y posición de los jugadores ordenados ascendente por equipo y descendente por su posición:

```
SELECT nombre, equipo, posicion  
FROM jugador  
ORDER BY 2, 3 DESC;
```

3.3.2 CLÁUSULA DISTINCT

El resultado de la ejecución de una sentencia *SELECT* devuelve todas las filas que cumplen la condición impuesta en la cláusula *WHERE*, incluidas las repetidas.

Para eliminar las filas repetidas, puede incluirse la palabra reservada *DISTINCT* antes del nombre de las columnas. En este caso, dos valores nulos se consideran iguales. En el siguiente ejemplo observamos la diferencia entre usar o no *DISTINCT*.



EJEMPLO 3.7

La siguiente consulta devuelve valores repetidos puesto que obviamente hay equipos repetidos en la tabla *jugador*:

```
SELECT equipo FROM jugador;
```



EJEMPLO 3.8

Seleccionar los distintos equipos que existen en la tabla *jugador*:

```
SELECT DISTINCT equipo FROM jugador;
```

3.3.3 CLÁUSULA LIMIT

Esta cláusula permite limitar el número de filas en el resultado de una consulta. Podemos especificar tanto un rango como un número máximo de filas a mostrar. Debe ir siempre al final de la consulta.



EJEMPLO 3.9

Obtener los 5 primeros registros de la tabla *jugador*:

```
SELECT * FROM jugador LIMIT 5;
```

Obtener los datos de los tres últimos equipos clasificados:

```
SELECT * FROM equipo ORDER BY puesto DESC LIMIT 3,6;
```

Ésta última consulta muestra los registros a partir del tercero en orden decreciente hasta el sexto ya que son 6 los equipos registrados.

3.3.4 EXPRESIONES

En una sentencia para formular una consulta se pueden realizar operaciones con los datos. Por ejemplo, se puede solicitar el resultado del producto de los valores de dos columnas, o el valor de una columna dividido por un valor. Para ello se utilizan expresiones en las que se combinan valores literales o de campos con operadores de distinto tipo.

También pueden utilizarse expresiones en las condiciones de búsqueda impuestas en la cláusula *WHERE*.

Una expresión es una combinación de operadores, operandos y paréntesis. El resultado de la ejecución de una expresión es un único valor.

En el formato de la sentencia *SELECT* descrito anteriormente, las expresiones pueden utilizarse en la cláusula *SELECT* en lugar de nombres de columnas y en la cláusula *WHERE* en la formulación de la condición.

Los operandos pueden ser nombres de columnas, constantes u otras expresiones. Más adelante veremos otros tipos de operandos, como las funciones de columna.

Los operadores actúan sobre datos homogéneos, es decir, bien numéricos o bien alfanuméricos.

Los tipos de operadores son aritméticos, de comparación, lógicos, de asignación, de bits, de cadenas y de control de flujo. A continuación, presentamos una tabla resumen de los mismos:

Tabla 3.4 Operadores en MySQL

Aritméticos	+	Suma
	-	Resta
	*	Multiplicación
	/	División
	** ^	Exponenciación
Relacionales o de comparación	<	Menor
	<=	Menor o igual
	>	Mayor
	>=	Mayor o igual
	<> !=	Distinto de
	!<	No menor que
	!>	No mayor que
Lógicos	AND	
	NOT	Los operadores lógicos permiten comparar expresiones lógicas devolviendo siempre un valor verdadero o falso. Los operadores lógicos se evalúan de izquierda a derecha.
	OR	
	XOR	

Como ejemplo de uso de operador aritmético multiplicación (representado por *) tenemos la siguiente consulta.



EJEMPLO 3.10

Calcular el salario neto anual a percibir por cada jugador suponiendo que el IRPF es un 18%:

1. `SELECT nombre, apellido, salario * 0,82 AS "salario neto anual"`
2. `FROM jugador;`

Las expresiones también admiten el uso de funciones. Normalmente los gestores incorporan distintos tipos como son las de cadenas, matemáticas o de fecha y hora.



EJEMPLO 3.11

Obtener la fecha actual del sistema:

```
SELECT CURRENT_DATE();
```

Obtener el nombre y apellido de cada jugador concatenados en un solo campo:

```
SELECT CONCAT(nombre, ' ', apellido) FROM jugador;
```

MySQL incorpora funciones predefinidas y agrupadas por tipo (de cadenas, de fecha hora, matemáticas etc.). Además, permite definir las nuestras propias como veremos en el Capítulo 5.

3.3.5 FUNCIONES PROPIAS DE MYSQL

Como hemos comentado el servidor MySQL incorpora funciones predefinidas que podemos utilizar en expresiones de consultas o en otros objetos del sistema gestor como disparadores o procedimientos. Igualmente podemos definir nuestras propias funciones adaptadas a nuestras necesidades usando el lenguaje propio del sistema gestor.

Todas ellas están perfectamente documentadas en el manual oficial, así que en esta sección nos limitaremos a mostrar una clasificación de las más utilizadas.

Funciones de control de flujo

Como su nombre indica controlan el flujo de la ejecución de la consulta devolviendo un valor u otro según ciertas condiciones.

Se dividen en: *CASE*, *IF*, *IFNULL*, *NULLIF*.

Funciones de cadena

Son funciones que operan sobre una o más cadenas de caracteres como, por ejemplo, la función *UPPER(c1)*, que devuelve la cadena c1 en mayúsculas.

Se dividen a su vez en:

- **Operadores de cadena:** funciones cuyos parámetros de entrada o salida son cadenas de caracteres.
- **Funciones de comparación:** funciones que comparan cadenas. Únicamente hay tres que son *LIKE*, *NOT LIKE* y *STRCMP*.
- **Expresiones regulares:** sirven para hacer comparaciones más complejas en cadenas. Son básicamente dos, *NOT REGEXP* y *REGEXP*, sinónima de *RLIKE*.

Veremos ejemplos de algunas de ellas en las siguientes secciones del capítulo.

Funciones matemáticas

Trabajan con valores numéricos como, por ejemplo, *ABS(n)*, que devuelve el valor absoluto de un número *n* o *EXP(n)*, que calcula la exponencial del número *n*.

Funciones de fecha y hora

Permiten operar con valores de tipo fecha y hora, por ejemplo, la función *ADDDATE(date, INTERVAL expr unit)* suma un intervalo de tiempo dado por *expr* en ciertas unidades dadas por *unit* a una fecha dada por *date*, o *CURTIME()* que devuelve la hora actual.

Funciones de búsqueda tipo full-text

Sirven para hacer búsquedas sobre campos de gran tamaño como *TEXT* o *BLOB* y que han sido indexados con índices de tipo *FULLTEXT*.

Las búsquedas se realizan mediante la cláusula *MATCH(col1, col2, ...) AGAINST(expr [search modifier])* en la que se especifica un conjunto de columnas en las que se buscará una coincidencia con una expresión. *Search modifier* especifica el tipo búsqueda a realizar.

Funciones XML

Trabajan con datos de tipo XML y, de momento, hay dos que son *ExtractValue(cadena xml, expresión xpath)* que extrae un valor a partir de una cadena XML y mediante el uso de la notación *XPath* y *UpdateXML(cadena xml objetivo, expresión xpath, nuevo xml)*, que sustituye en una cadena XML por otra según una expresión *XPath*.

Funciones de compresión y codificación

Sirven para comprimir y/o (de)codificar cadenas de caracteres. También permiten hacer sumas de verificación (*checksum*) de cadenas de texto.

Por ejemplo, *DES_DECRYPT* decodifica la cadena *c1* codificada con la función *DES_ENCRYPT*.

3.3.6 CLÁUSULA WHERE

En las consultas vistas hasta ahora hemos visto consultas sin condiciones. Pueden especificarse condiciones de búsqueda complejas que proporcionan una gran potencia de selección de filas. Estas condiciones se denominan predicados.

Un predicado expresa una condición que se cumple o no sobre un conjunto de dos valores o expresiones y el resultado de su evaluación puede ser “Verdadero”, “Falso” o “Desconocido”.

Los predicados se expresan normalmente mediante cláusulas *WHERE*.

Solo se considera satisfecha la condición de búsqueda expresada en un predicado cuando toma el valor “Verdadero”. Esto quiere decir que el resultado de la evaluación de un predicado expresado en la cláusula *WHERE* da lugar a la recuperación de las filas para las que toma el valor “Verdadero” y se rechazarán las filas para las que tome el valor “Falso” o “Desconocido”.

3.3.7 PREDICADOS EN SQL

Predicados de comparación

Expresan condiciones de comparación entre dos valores usando operadores lógicos. De manera que el predicado es “Verdadero” si y solo si:

- ✓ $x = y$ x es igual a y
- ✓ $x <> y$ ó $x \neq y$ x no es igual a y
- ✓ $x < y$ x es menor que y
- ✓ $x > y$ x es mayor que y
- ✓ $x <= y$ x es menor o igual que y
- ✓ $x >= y$ x es mayor o igual que y

Si alguno o ambos de los operadores x o y es nulo, el resultado de la evaluación del predicado toma el valor “Desconocido”. Para el resto de los casos toma el valor “Falso”.

Los operandos x o y pueden ser expresiones.



EJEMPLO 3.12

Seleccionar el nombre y apellido de aquellos jugadores que sean pivot:

```
SELECT nombre, apellido  
FROM jugador  
WHERE posicion = "pivot";
```



EJEMPLO 3.13

Seleccionar datos de jugadores que no pertenezcan al equipo 3:

```
SELECT * FROM jugador WHERE equipo <> 3;
```

Comprobación de valor nulo. Predicado **NULL**

✓ Formato

```
nom_columna IS [NOT] NULL
```

Se utiliza para consultar si el valor de la columna, *nom_columna*, de una fila determinada es o no nulo. Si es nulo el resultado será “Verdadero”, si no lo es “Falso”. No puede tomar el valor “Desconocido”.



EJEMPLO 3.14

Seleccionar aquellos datos de equipos cuya web es nula:

```
SELECT * FROM equipo WHERE web IS NULL;
```

Predicado **IN**. Pertenencia a un conjunto

✓ Formato

```
Expresión [NOT] IN (valor1, valor2,...)
```

Se utiliza para averiguar si el resultado de la evaluación de una expresión está incluido en la lista de valores especificada tras la palabra *IN*.

Si el resultado de la expresión es no nulo, y es igual a alguno de los valores de la lista, el predicado es “Verdadero”, si no es “Falso”.

Si la expresión devuelve un valor nulo, el predicado toma el valor “Desconocido”.

En lugar de una lista de valores, puede especificarse una sentencia *SELECT* subordinada (una consulta anidada dentro de otra), que deberá devolver una tabla con una sola columna y no podrá contener la cláusula *ORDER BY*.

En este caso, el formato sería:

Expresión IN (subselect)

Siendo *subselect* una sentencia *SELECT* subordinada o subconsulta (es una consulta dentro de otra. Las veremos más tarde en este capítulo).



EJEMPLO 3.15

Obtener los datos de los equipos menos los de Valencia y Madrid:

```
SELECT * FROM equipo WHERE ciudad NOT IN ('Valencia', 'Madrid');
```

Predicado [NOT] BETWEEN *expr1*- AND *expr2*

✓ Formato

```
expresion1 [NOT] BETWEEN expr2 AND expr3
```

Se utiliza para comprobar si un valor está comprendido entre otros dos (ambos inclusive), o no. Si se omite *NOT*, el predicado es verdadero si el valor considerado en *expresión* está comprendido entre el valor de *expr2* y el de *expr3*, ambos inclusive.

Si se especifica *NOT*, el predicado es verdadero si el valor no está en ese rango.

Si alguno de los valores de *expresión1*, *expr2* o *expr3* toma un valor nulo, el predicado toma un valor desconocido.



EJEMPLO 3.16

Obtener los datos de partidos de marzo de 2010:

```
SELECT * FROM partido WHERE fecha BETWEEN '01-03-2010' AND '31-03-2010';
```

Predicado LIKE

✓ Formato

```
nom_columna [NOT] LIKE cte_alfanumérica
```

Se utiliza para buscar combinaciones de caracteres que coincidan con un patrón especificado según la expresión *X LIKE Y*.

Donde *X* es el nombre de una columna de tipo alfanumérico e *Y* una constante del mismo tipo utilizada como patrón de búsqueda.

La constante alfanumérica puede contener cualquier carácter válido, pero dos de ellos, el carácter de subrayado “_” y el símbolo de porcentaje “%” son comodines. El carácter “_” es equivalente a cualquier otro carácter y el “%” equivale a cualquier conjunto de caracteres.

Es decir, a partir del valor de la cadena *Y* se generan otras cadenas de caracteres sustituyendo cada carácter “_” por un único carácter cualquiera y siempre uno, y cada “%” por una cadena cualquiera de cualquier longitud (incluida la cadena vacía).

Los resultados pueden ser:

- Si *X* no es nulo, el predicado es verdadero si su valor está incluido entre los que se pueden generar a partir del patrón *Y*. Si no toma el valor “Falso”.
- Si *X* e *Y* son ambas cadenas vacías (cadenas de longitud cero), se conviene que el predicado es verdadero.
- Si *X* es nulo, el resultado es “Desconocido”.
- Si el valor de *Y* no contiene caracteres “_” o “%”, el predicado *X LIKE Y* equivale a *X=Y*.

Predicado *REGEXP* y expresiones regulares

✓ Formato

```
nom_columna [NOT] REGEXP cte_alfanumérica
```

El predicado *REGEXP* (equivalente a *RLIKE*) tiene la misma funcionalidad que *LIKE* aunque es mucho más potente ya que se basa en el uso de expresiones regulares.

Una expresión regular es una forma muy potente de especificar un patrón para una búsqueda compleja.

Una expresión regular es una cadena formada por caracteres literales y otros con una función especial en el mismo sentido que “_” o “%”. Así esta cadena describe un conjunto de cadenas posibles.

La expresión regular más sencilla es aquella que no contiene caracteres especiales. Por ejemplo, la expresión regular “hola” coincide con “hola” y con nada más.

Las expresiones regulares no triviales usan ciertas construcciones especiales de modo que pueden coincidir con más de una cadena. Por ejemplo, la expresión regular “Hola | mundo” coincide tanto con la cadena “Hola” como con la cadena “mundo”.

Como ejemplo algo más complejo, la expresión regular “B[an]*s” coincide con cualquiera de las cadenas siguientes “Bananas”, “Baaaaas”, “Bs”, y cualquier otra cadena que empiece con “B”, termine con “s”, y contenga cualquier número de caracteres “a” o “n” entre la “B” y la “s”.

Una expresión regular para el operador *REGEXP* puede incluir cualquiera de una serie de caracteres especiales. En el siguiente ejemplo mostramos los más básicos:



EJEMPLO 3.17

1. Carácter ^: coincidencia del principio de una cadena.

Datos de jugadores cuyo nombre comience por A:

```
SELECT * FROM jugador WHERE nombre REGEXP '^A';
```

2. Carácter \$: coincidencia del final de una cadena.

Datos de jugadores cuyo nombre termine por a:

```
SELECT * FROM jugador WHERE nombre REGEXP 'a$';
```

3. Carácter .: coincidencia de cualquier carácter.

Datos de jugadores cuyo nombre tenga 4 caracteres:

```
SELECT * FROM jugador WHERE nombre REGEXP '....';
```

4. Carácter ?: coincidencia de cero o más caracteres cualesquier. Es equivalente al ‘_’ en el predicado *LIKE*.

Datos de jugadores cuyo nombre comience por A o por B:

```
SELECT * FROM jugador WHERE nombre REGEXP '^A?B'
```

5. Carácter *: coincidencia de cero o más caracteres iguales que el precedente

Datos de equipos cuya web tenga cero o más w:

SELECT * FROM equipo WHERE web REGEXP 'w*';

6. Carácter +: coincidencia de cualquier secuencia de uno o más caracteres iguales que el precedente.

Datos de equipos cuya web tenga una o más w:

SELECT * FROM equipo WHERE web REGEXP 'w+';

7. Carácter |: coincidencia de una entre varias secuencias posibles

Datos de jugadores cuyo nombre emiece por San o por Ju:

SELECT * FROM jugador WHERE nombre REGEXP '(San|Ju)';

8. Carácter (secuencia): para indicar coincidencia de una secuencia de varios caracteres.

Datos de equipos cuya web tenga una ocurrencia o más de tres w:

SELECT * FROM equipo WHERE web REGEXP '(www)+';

9. Carácter {n}, {n,m}: permite indicar un número o rango de ocurrencias para un determinado patrón o carácter.

Datos de equipos cuya web tenga entre una y tres w:

SELECT * FROM equipo WHERE web REGEXP 'w{1,3}';

10. Carácter [a-z], [^a-z]: coincide con cualquier carácter que esté o no esté (de ahí el ^) en el rango indicado entre corchetes.

Nombres de jugadores sin vocales:

SELECT nombre FROM jugador WHERE REGEXP '[^aeiou]';

Datos de equipos cuya web sea del tipo *http://www.dominio.extensión* siendo la extensión de tres caracteres:

SELECT * FROM equipo WHERE web REGEXP '^http://www\\.\\.\\.\\{3}';

En este último ejemplo más complejo vemos la mayoría de los aspectos de expresiones regulares comentados. La expresión indica la necesidad de empezar con la cadena *http://www* para después incluir un punto. En este caso debemos "escaparlo" o poner dos barras delante para que no se interprete como un comodín. Después obligamos a que haya cualquier secuencia de caracteres exceptuando el punto, de nuevo "escapado", y otra vez un punto. Finalmente, la cadena debe terminar en tres, y solo tres, caracteres cualesquiera. Para ser más estrictos debiéramos haber excluido caracteres como punto y coma o dos puntos pero para el propósito del ejemplo es suficiente.

Para realizar los ejemplos debemos haber instalado el servidor MySQL y conectarnos con el programa cliente, bien usando *mysql.exe* desde consola o con el MySQL Workbench de manera gráfica, tal como se indicó al comenzar el capítulo.



La sintaxis de expresiones regulares es mucho más extensa que lo que hemos mostrado. Para estudiar y profundizar en ello se recomienda consultar tanto el manual de MySQL como las direcciones de referencia al final del libro.

*^ [a-z] + [a-z] \\ . - _ 0 - 9 \n] + @
[a-z \d - 9] + (\\ . [a-z] {2,5}) + 2 \$*

*Select 'a.a@ya.com' like '^ [a-z] + \\ . ? [a-z] + @ [a-z] {2,5}
\\ . [a-z] {2,3} \$';*

Predicados compuestos

Los predicados compuestos son combinaciones de predicados simples hechas con los operadores lógicos *AND*, *OR*, *XOR* y *NOT*.

AND, *XOR* y *OR* se aplican a dos operandos, mientras que *NOT* se aplica a uno solo. En todos los casos, los operandos son otros predicados.

Los predicados compuestos, al igual que los simples, pueden tomar los valores “Verdadero”, “Falso” o “Desconocido”.

Cuando se utiliza *AND*, el resultado es “Verdadero” cuando los dos predicados lo son.

Cuando se utiliza *OR*, el resultado es “Verdadero” cuando lo es cualquiera de sus operandos.

XOR devuelve “Verdadero” cuando los dos predicados son uno verdadero y otro falso y “Falso” en caso de que sean iguales.

Si hay más de dos predicados el valor devuelto será “Verdadero” si hay un número impar de verdaderos y “Falso” en caso contrario.

Cuando se utiliza *NOT*, el resultado es “Verdadero” cuando el predicado sobre el que se aplica es “Falso”.



EJEMPLO 3.18

Seleccionar el nombre de los jugadores pívot que ganen más de 100.000 euros:

```
SELECT nombre FROM jugador  
WHERE posicion = "pivot" AND salario > 100000;
```



EJEMPLO 3.19

Seleccionar el nombre de jugadores de los equipos 1 y 2 que jueguen como pívot:

```
SELECT nombre FROM jugador  
WHERE posicion = "base" AND (equipo=1 OR equipo=2);
```

Debemos tener cuidado cuando hay más de dos predicados ya que la expresión lógica puede cambiar si no incluimos paréntesis. En el caso anterior si omitimos los paréntesis obtendríamos los jugadores base del equipo 1 y todos los del equipo 2.

ACTIVIDADES 3.1



► Realice consultas para obtener la siguiente información sobre las bases de datos *liga* y *motorblog*:

- a. Datos de jugadores del equipo 3 ordenados por apellido.
- b. Datos de los jugadores que sean pivot ordenados por su identificador.
- c. Datos de jugadores de más de dos metros y ganen menos de 40.000 euros.
- d. Datos de los partidos jugados en febrero.
- e. Nombre y apellido de los capitanes de equipos en la base de datos *liga*.
- f. Datos de jugadores de los equipos 1 y 2 que ganen más de 80.000 euros al mes.
- g. Título de noticias que contengan una dirección web.
- h. Enlaces que no tengan "www".
- i. Enlaces que terminen en *rss* ó *xml*.
- j. Enlaces con el formato: <http://loquesea.loquesea.loquesea/loquesea>

3.3.8 FUNCIONES DE AGREGADO

Estas funciones (también llamadas de columnas, o colectivas) permiten obtener un solo valor como resultado de aplicar una determinada operación a los valores contenidos en una columna.

Son aquellas cuyo argumento es una colección de valores tomados de los pertenecientes a una o más columnas o campos de una tabla. Se llaman también por ello funciones de columna.

Se aplican a la colección de valores del argumento y producen un único resultado a partir de ellos. Son las siguientes:

- *Avg*: devuelve la media de los valores de la colección.
- *Max*: devuelve el valor máximo de la colección.
- *Min*: devuelve el valor mínimo.
- *Sum*: devuelve la suma.
- *Count*: devuelve el número de elementos que tiene la colección.

Antes de aplicar las funciones se construyen uno o más grupos de filas. La forma de construir grupos se especifica mediante la cláusula *GROUP BY* (que veremos más adelante). Si no se emplea *GROUP BY* se considera un solo grupo formado por todas las filas de la tabla que cumplen el predicado de la cláusula *WHERE*. A todas ellas se les aplican las funciones colectivas y el resultado será una tabla con una sola fila y con tantas columnas como expresiones haya en la cláusula *SELECT*.

Reglas y formatos de las funciones de agregado

Antes de aplicar una función de agregado a la colección de valores de su argumento se eliminan los valores nulos, si existen.

Si la colección es vacía, la función *COUNT* devuelve un valor cero y las demás un valor nulo.

Para *AVG*, *MAX*, *MIN* y *SUM* el resultado tiene el mismo tipo de dato que el argumento. Este debe ser numérico para *AVG* y *SUM* y puede ser de cualquier tipo para *MAX* y *MIN*.

Si el argumento es de tipo *DECIMAL(p,s)* el resultado de la función *SUM* es también de tipo *DECIMAL* salvo para la función *AVG* que devolvería el resultado con más decimales.

Si el argumento es de tipo *INTEGER* o *SMALLINTEGER* la función *AVG* puede perder cifras decimales al calcular la media, al ser el resultado también de tipo entero.

Hay tres formatos:

✓ **Formato 1**

- nom_función ([DISTINCT] nom_columna)
- nom_función: cualquiera de las vistas.
- nom_columna: nombre de una columna. No puede ser una expresión.

La palabra *DISTINCT* no se considera un argumento de la función. Se emplea para, antes de aplicar la función de columna a los valores de la colección, eliminar de ella los valores repetidos. Evidentemente, el uso de *DISTINCT* con *MAX* y *MIN* es, aunque lícito, absurdo.

En una cláusula *SELECT* no puede especificarse *DISTINCT* más de una vez, ya sea dentro de una función o detrás de la cláusula *SELECT*.

✓ **Formato 2**

- nom_función (expresión)
- nom_función: cualquiera, excepto *COUNT*.
- expresión: en la que debe haber al menos un nombre de columna y no puede haber otra función colectiva.

✓ **Formato 3**

COUNT (*)

El asterisco indica cualquier campo. Solo es válido para la función *COUNT*.

Devuelve el número de filas que hay en el grupo sobre el que se aplica.



EJEMPLO 3.20

Calcular el número de jugadores que miden más de dos metros:

```
SELECT COUNT(comision) FROM jugador WHERE altura > 2,00;
```

Calcular el salario medio de todos los jugadores:

```
SELECT AVG(salario) AS "Salario Medio" FROM jugador;
```

Encontrar el salario más alto, el más bajo y la diferencia entre ambos:

```
SELECT MAX(salario), MIN(salario), MAX(salario) - MIN(salario) "Diferencia  
salarios" FROM jugador;
```

Hallar el número de ciudades en las que hay equipos registrados:

```
SELECT COUNT (DISTINCT ciudad) FROM equipo;
```

Obtener el salario mensual neto de cada jugador suponiendo un IRPF del 18%:

```
SELECT SUM(salario *0,82/12) "Salario mensual" FROM jugador;
```

3.3.9 CLÁUSULA GROUP BY. CONSULTAS CON AGRUPAMIENTO DE FILAS

Existe la posibilidad de formar grupos de filas de acuerdo con un determinado criterio para aplicarles después una función colectiva.

Cláusula opcional de la sentencia *SELECT* que sirve para agrupar filas.

Si se especifica, debe aparecer después de la cláusula *WHERE*, si ésta existe.

Formato

GROUP BY col1 [, col2]...

Donde *col1*, *col2*, son nombres de columnas a las que denominaremos **columnas de agrupamiento**.

La cláusula *GROUP BY* indica que se han de agrupar las filas de la tabla de modo que todas las que tengan iguales valores para las columnas de agrupamiento formen un grupo.

Pueden existir grupos de una sola fila.

Los valores nulos se consideran iguales. Se incluyen en el mismo grupo.

Una vez formados los grupos, para cada uno de ellos se evalúan las expresiones de la cláusula *SELECT*. Por lo tanto, cada uno de ellos produce una única fila en la tabla resultante.

Las columnas que participen en estas expresiones y no sean de agrupamiento solo pueden especificarse en los argumentos de funciones colectivas. Dicho de otro modo, si aparece una columna en un *SELECT* y no está incluida en una función colectiva debe ser una columna de agrupamiento, si no la consulta será errónea.



EJEMPLO 3.21

Seleccionar el número de jugadores por equipo:

```
SELECT COUNT(*) GROUP BY equipo;
```

Seleccionar el salario, mínimo y máximo de los jugadores, agrupados por equipo:

```
SELECT equipo, MIN(salario), MAX(salario)  
FROM jugador  
GROUP BY equipo;
```

Como veremos más adelante puede hacerse agrupaciones sobre más de un campo.

3.3.10 CLÁUSULA HAVING

Cláusula opcional de la sentencia *SELECT* utilizada para filtrar los resultados de una función de agregado cuando hay agrupamiento de filas.

Formato

HAVING condición

Indica que, después de haber formado los grupos de filas, se descarten aquellos grupos que no cumplan la condición expresada con un predicado.

Como ya hemos visto, la configuración de las filas en grupos se realiza mediante la cláusula *GROUP BY* o, si ésta no existe, formando un solo grupo con todas las filas.

Si se especifica la cláusula *GROUP BY*, ésta debe preceder a la cláusula *HAVING*.

La condición es un predicado simple o compuesto en el que las columnas que participen y no sean de agrupamiento deberán figurar como argumentos de funciones colectivas.



EJEMPLO 3.22

Seleccionar el salario medio de cada equipo, pero solo para aquellos cuya media sea superior a 50.000:

```
SELECT AVG(salario) FROM jugador GROUP BY equipo HAVING AVG(salario) > 50000;
```

La cláusula *HAVING* puede utilizarse también sin un *GROUP BY* previo. En este caso se aplica la condición al único grupo de filas formado por todas las filas de la tabla resultante.

Funcionamiento de consultas con agrupamiento de filas

Conviene, en este punto, describir en detalle cómo funciona el agrupamiento en MySQL.

El agrupamiento ocurre cuando, o bien se utiliza la cláusula *GROUP BY*, o bien se utilizan funciones colectivas en las expresiones de la cláusula *SELECT*, o bien se dan ambos casos.

Si se especifica *GROUP BY* se agrupan las filas que tengan iguales valores en las columnas de agrupamiento. Si no, se asume que todas las filas forman un único grupo.

Una vez formados los grupos:

- Para cada grupo se evalúan las expresiones de la cláusula *SELECT*, dando lugar a una fila en la tabla resultante.
- Las columnas que no sean de agrupamiento solo pueden usarse como participantes en los argumentos de funciones colectivas, ya sea en las expresiones de la cláusula *SELECT* o en la condición de la cláusula *HAVING*.

La palabra *DISTINCT* solo puede especificarse una vez, bien en la cláusula *SELECT* o bien dentro de funciones colectivas en la condición de la cláusula *HAVING*. Este límite no rige para las sentencias subordinadas que pueda haber en los predicados de las cláusulas *WHERE* o *HAVING*.

ACTIVIDADES 3.2



➤ Obtenga mediante consultas la siguiente información sobre la base de datos *liga*:

- a. Número de partidos jugados en febrero.
- b. *Id* de equipo y suma de las alturas de sus jugadores.
- c. *Id* de equipo y salario total de cada equipo para equipos con más de 4 jugadores registrados.
- d. Número de ciudades distintas.
- e. Datos del jugador más alto.

3.4 SUBCONSULTAS

En muchas ocasiones no conocemos de antemano el valor de una condición en un predicado de la cláusula *WHERE* o *HAVING* y a que depende del valor de otra consulta. Por ejemplo, si queremos saber que jugadores cobran más que Gasol. En este caso no conocemos el salario de Gasol así que se requiere una consulta adicional. Para evitar hacer dos consultas se usan las consultas subordinadas o subconsultas. En nuestro ejemplo quedaría así:



EJEMPLO 3.23

```
SELECT * FROM jugadores WHERE salario > (SELECT salario FROM jugadores WHERE apellido='Gasol');
```

El segundo *SELECT* debe ir entre paréntesis y devolver como resultado un único valor. Es decir, la tabla resultante debe tener una sola columna y una fila o ninguna. Además no se puede especificar en ella la cláusula *ORDER BY*. Si el resultado de esta sentencia *SELECT* es una tabla vacía, su valor se toma como "desconocido".

Una sentencia subordinada de otra puede tener a su vez otras sentencias subordinadas a ella. Llamamos sentencia externa a la primera de todas, la que no es subordinada de ninguna. Una sentencia es antecedente de otra cuando esta es su subordinada directa o subordinada de sus subordinadas a cualquier nivel.

A las sentencias subordinadas suele llamarlas anidadas. Puede haber varios niveles de anidamiento según el SGBD.

También sirve para funciones de agregado.



EJEMPLO 3.24

Calcular el número de jugadores por equipo que cobra más que el salario medio de todos los jugadores:

```
SELECT equipo, COUNT(*) FROM jugador WHERE salario > (SELECT AVG(salario) FROM jugador) GROUP BY equipo
```

Las subconsultas pueden ser parte de los siguientes predicados:

- Predicados básicos de comparación.
- Predicados cuantificados (*ANY*, *SOME*, *ALL*).
- Predicado *EXISTS*.
- Predicado *IN*.

Predicados cuantificadores (*ALL*, *SOME*, *ANY*)

Como hemos visto, cuando se utiliza una sentencia *SELECT* subordinada en un predicado de comparación, el resultado debe ser un valor único (una tabla con una sola fila y una sola columna).

Pero se permite que el resultado de la sentencia *SELECT* subordinada tenga más de un valor si ésta viene precedida de una de las palabras reservadas *ALL*, *SOME*, *ANY* (palabras cuantificadoras). Cuando se utilizan estas palabras, los predicados en los que participan se denominan predicados cuantificados.

En ellos, el resultado de la ejecución de la sentencia *SELECT* subordinada debe ser una tabla con una sola columna y cero o más filas.

■ Cuantificador *ALL*

El predicado cuantificado es verdadero si la comparación es verdadera para todos y cada uno de los valores devueltos por la *SELECT* subordinada.

Si la *SELECT* subordinada devuelve una tabla vacía, el predicado cuantificado toma el valor “Verdadero”.

Si devuelve uno o más valores y alguno de ellos es nulo, el predicado cuantificado puede ser:

- “Falso”, si para alguno de los valores no nulos la comparación toma el valor “Falso”.
- “Desconocido”, si la comparación es verdadera para todos los valores no nulos.

Si devuelve uno o más valores y ninguno de ellos es nulo, el predicado cuantificado es:

- “Verdadero”, si la comparación lo es para todos los valores de la tabla devuelta. En otro caso es “Falso”.



EJEMPLO 3.25

Obtener el nombre de los jugadores que ganen más que todos los del equipo 2:

```
SELECT nombre FROM jugador WHERE salario > ALL (SELECT salario FROM jugador WHERE equipo= 2);
```

■ Cuantificador *ANY* o *SOME*

El predicado cuantificado es verdadero si la comparación es verdadera para uno cualquiera de los valores devueltos por la ejecución de la sentencia *SELECT* subordinada.

Si la sentencia subordinada devuelve una tabla vacía, el predicado cuantificado toma el valor “Falso”.

Si devuelve una o más filas y alguna de ellas es nula, el predicado cuantificado puede ser:

- “Verdadero”, si para alguno de los valores no nulos el resultado de la comparación es “Verdadero”.
- “Desconocido”, si para todos los valores no nulos de la tabla el resultado de la comparación es “Falso”.

Si devuelve una o más filas y ninguna es nula, el predicado cuantificado es verdadero si la comparación es verdadera para alguno de los valores. En cualquier otro caso es “Falso”.



EJEMPLO 3.26

Seleccionar los jugadores que ganen más que alguno de los del equipo 5:

```
SELECT nombre FROM jugador WHERE salario > ANY (SELECT salario FROM jugador WHERE equipo = 5);
```

Predicado IN

Podemos considerar una subconsulta como un conjunto de valores con los que usar la cláusula *IN*.



EJEMPLO 3.27

Datos de los jugadores que jueguen en Zaragoza:

```
SELECT * FROM jugador WHERE equipo IN (SELECT id_equipo FROM equipo WHERE ciudad='Zaragoza');
```

En este caso la subconsulta devuelve los equipos de Zaragoza y los compara con los equipos de la tabla *jugador*. Cada coincidencia será una fila de la tabla resultante.

Predicado EXISTS

Devuelve “Verdadero” si la subconsulta subsiguiente es no vacía y “Falso” en caso contrario.



EJEMPLO 3.28

Obtener los datos de los jugadores pero solo si hay más de 10 equipos:

```
SELECT * FROM jugador WHERE EXISTS (SELECT COUNT(*) FROM equipo HAVING COUNT(*)>10);
```

Si la subconsulta no devuelve nada (o sea no hay más de 10 equipos) no se mostrará ningún jugador.

Este tipo de consultas tiene su verdadera fuerza en las consultas correlacionadas que veremos a continuación.

3.4.1 CONSULTAS CORRELACIONADAS

En las sentencias anidadas vistas hasta ahora, éstas no hacen referencia a columnas de tablas que no estén en su propia cláusula *FROM*. Esto significa que el resultado de la sentencia subordinada puede evaluarse independientemente de sus sentencias antecedentes de cualquier nivel. El SGBD las evalúa una sola vez y reemplaza los valores resultantes en el predicado donde se encuentre.

En las sentencias correlacionadas no ocurre así. Se llaman correlacionadas las sentencias subordinadas en las que se hace referencia a alguna columna de una tabla mencionada en la cláusula *FROM* de alguna de sus sentencias antecedentes.

De esta forma, una sentencia correlacionada no puede evaluarse independientemente de sus antecedentes, pues su resultado puede cambiar, según qué filas se consideren en la evaluación de éstas en cada momento. El SGBD, por tanto, las evaluará múltiples veces, tantas como filas haya en la tabla de la consulta principal. El proceso en detalle es el siguiente:

- ✓ 1. La consulta externa pasa los valores de cada fila de la consulta a la consulta interna o subconsulta.
- ✓ 2. La consulta interna usa los valores que le pasa la consulta externa para evaluarlos.
- ✓ 3. La consulta interna devuelve los valores que cumplan las condiciones a la consulta externa.
- ✓ 4. Se repite el proceso para todas las filas de la consulta externa.

Dado que en estas consultas se hace referencia a más de una tabla y que los campos pueden tener el mismo nombre es normal usar prefijos para hacer referencia a los mismos y así evitar ambigüedades. Así para referirnos al nombre de un jugador escribiremos *jugador.nombre*. Del mismo modo para hacer más cómoda su escritura se usan alias para las tablas en la cláusula *FROM* tal y como se hace en el siguiente ejemplo.



EJEMPLO 3.29

Obtener los datos de jugadores que miden más que la media de su equipo:

```
SELECT * FROM jugador j1 WHERE altura > (SELECT AVG(altura) FROM jugador j2 WHERE j1.equipo=j2.equipo);
```

En este caso se selecciona la primera fila de la tabla *jugador* y se envían sus valores a la subconsulta.

Se observa el uso del alias *j1* para la tabla *jugador* y *j2* para la tabla *jugador* de la subconsulta.

En ella se calcula la media teniendo en cuenta que se filtran las filas del mismo equipo que el jugador correspondiente a la fila que está siendo procesada.

Con el valor obtenido y el valor de la altura de la primera fila se verifica la condición y en caso afirmativo se guarda para ser mostrada al final de la consulta. El proceso se itera con el resto de filas de *jugador*.

Obtener los datos de equipos con más de 5 jugadores:

```
SELECT * FROM equipo e WHERE 5<(SELECT COUNT(*) FROM jugador j WHERE j.equipo=e.id_equipo);
```

En este segundo caso se evalúa cada fila de equipo para calcular el número de jugadores en la subconsulta. Obviamente, hace falta considerar el equipo para calcular ese número por lo que la consulta es correlacionada. Depende del valor de equipo que estemos considerando. Así, si la cuenta resulta mayor de 5, la fila correspondiente se mostrará en la tabla resultante.

Predicado EXISTS en consultas correlacionadas

Este predicado es frecuentemente usado en subconsultas correlacionadas para verificar cuando un valor recuperado por la consulta externa existe en el conjunto de resultados obtenidos por la consulta interna. Si la subconsulta obtiene al menos una fila, el operador obtiene el valor “Verdadero” y termina. Si el valor no existe, se obtiene el valor “Falso”. Consecuentemente, *NOT EXISTS* verifica cuándo un valor recuperado por la consulta externa no es parte del conjunto de resultados obtenidos por la consulta interna.



EJEMPLO 3.30

Obtener los datos de los capitanes de los equipos:

```
SELECT * FROM jugador j1
WHERE j1.id_jugador EXISTS (SELECT * FROM jugador j2
WHERE j1.id_jugador = j2.capitan);
```

Esta consulta puede resolverse también con *IN*:

```
SELECT * FROM jugador j1
WHERE j1.id_jugador IN (SELECT j2.capitan FROM jugador j2
WHERE j2.capitan IS NOT NULL);
```

Uso de una subconsulta como una expresión

Podemos incluir una consulta dentro de una cláusula *SELECT* a modo de expresión.



EJEMPLO 3.31

Por ejemplo, si queremos saber los datos de los jugadores con el salario medio de su equipo y la diferencia de éste con el de cada jugador:

```
SELECT num_emp, sal, (SELECT AVG(sal) 'media' FROM emp) AS t, sal-(SELECT
avg(sal) 'media' FROM emp) AS diferencia;
```

Consultas con tablas derivadas

Otro tipo de subconsultas se dan cuando necesitamos usar una consulta en la cláusula *FROM*, es decir, como una tabla derivada.



EJEMPLO 3.32

Obtener el maximo salario total de todos los equipos:

```
SELECT max(tderivada.maxsal) FROM (SELECT sum(salario) 'maxsal' FROM jugador GROUP
BY equipo) AS tderivada;
```

Ahora la cláusula *FROM* actúa sobre una tabla derivada (con el alias *tderivada*) tal como lo hace sobre una tabla base considerando el resultado de la subconsulta como una tabla.

① Select nombre, equipo, salario, (select avg(salario) from jugador j2
where j1.equipo = j2.equipo) as media
from Jugador j1 order by equipo;

ACTIVIDADES 3.3

► Obtenga mediante consultas la siguiente información sobre la base de datos *liga*:

- Datos del jugador más alto.
- Suma de alturas de los jugadores del CAI y Madrid.
- Datos de jugadores de equipos que hayan jugado algún partido contra el Valencia en casa.
- Resultado más repetido del Tenerife.
- Nombre de jugadores que midan más que todos los del Caja Laboral
- Datos de jugadores cuyo salario sea mayor que el de sus capitanes.
- Datos del equipo con más jugadores registrados.
- Datos del equipo que ha jugado más partidos.
- Nombre de los jugadores mejor y peor pagados.
- Datos de equipos que se hayan enfrentado a todos los demás.

3.5 CONSULTAS SOBRE VARIAS TABLAS

En las consultas vistas hasta ahora y con la excepción de las subconsultas solo hemos necesitado una sola tabla ya que era todo lo necesario, tanto los campos a mostrar como las condiciones impuestas estaban en la tabla. Sin embargo, en la mayoría de casos necesitaremos campos de otras tablas bien porque necesitamos obtener más información (como el nombre del equipo de cada jugador) o bien porque las condiciones o filtros afecten a campos de otras tablas (como cuando necesitamos los jugadores del equipo CAI Zaragoza). Para estos casos necesitamos incluir en la cláusula *FROM* las tablas requeridas por la consulta en un proceso conocido como producto cartesiano o combinación de tablas.

Este proceso, cuando ocurre entre dos tablas, tiene como producto una nueva tabla resultado de la combinación de cada fila de la primera tabla con cada fila de la segunda, de modo que la nueva tabla tiene un número de filas igual al producto de las filas de las dos tablas iniciales.

Por ejemplo, la siguiente consulta:

```
SELECT * FROM equipo, jugador
```

Produce una nueva tabla con 14*5 filas.

Evidentemente no todas ellas son válidas puesto que estamos combinando cada jugador con todos los equipos.

A estas filas se les llama **filas espurias** y debemos deshacernos de ellas antes de seguir con el diseño de la consulta.

Para ello debe incluirse un filtro con la cláusula *WHERE* que deje solamente las filas válidas. En nuestro ejemplo son aquellas en que el campo común (o clave ajena) *id_equipo* es igual. De este modo quedaría:

```
SELECT * FROM equipo, jugador WHERE e.id_equipo=j.equipo
```

(h) *Select elowl From (select elowl, count(*) as MAX From (select elowl From partidas Union All Select visitante From partidas) as t1 group by elowl) as t2 Where t2.MAX = (select max(MAX) From (select elowl, count(*) as MAX From partidas Union All Select visitante From partidas) as t1 group by elowl)* →

Así veríamos los datos de cada jugador incluidos los de su equipo.

Para eliminar la posible ambigüedad derivada del hecho de que puede haber campos con el mismo nombre es conveniente usar siempre prefijos para las tablas.

El proceso de diseño de una consulta de varias tablas se resume en:

- ✓ 1. Analizar la consulta para ver las tablas necesarias para resolverla.
- ✓ 2. Incluir dichas tablas en el *FROM*.
- ✓ 3. Filtrar filas espurias usando campos comunes o claves ajenas.
- ✓ 4. Añadir los filtros o cláusulas necesarias como si trabajásemos con una única tabla.

3.5.1 OPERACIONES DE REUNIÓN (JOIN)

Para indicar la combinación usamos el carácter “,” que hace referencia a la llamada reunión interna. Sin embargo podemos realizar distintos tipos de combinación o *JOIN*.

INNER JOIN: composición interna

Es la combinación de las tablas indicadas en la cláusula *FROM* tal y como hemos visto en la sección anterior. En este caso se requiere el filtro adicional para eliminar filas espurias.



EJEMPLO 3.33

Obtener número de jugadores de equipos de Madrid:

```
SELECT COUNT(*) FROM jugador j, equipo e  
WHERE j.equipo = e.id_equipo AND ciudad= "Madrid";
```

El uso de la coma es equivalente a usar *JOIN* que también puede ir precedido por las palabras *INNER* y *CROSS* (tal y como aparecen en la sintaxis de MySQL) indistintamente.

También puede usarse la cláusula *ON* en lugar de *WHERE* para especificar la condición de filtro de filas espurias. Así el ejemplo anterior quedaría:

```
SELECT COUNT(*) FROM jugador j, equipo e  
ON j.equipo = e.id_equipo AND ciudad= "Madrid"
```

OUTER JOIN: composición externa

En este tipo de combinaciones todas las filas se combinan incluso aunque tengan valores nulos en los campos comunes. Hay dos tipos, izquierda (*LEFT JOIN*) y derecha (*RIGHT JOIN*).

LEFT JOIN: se combinan todas las filas de la primera tabla en la cláusula *FROM* con cada fila de la segunda tabla que cumpla la condición expresada con *ON* (en este tipo de consultas no se puede usar *WHERE* para comparar campos comunes).

(select elocal from partido union all select visitante from partido) as t3 group by elocal) as t4;



EJEMPLO 3.34

Mostrar los datos de todos los jugadores, incluyendo datos de sus equipos en caso de tener:

```
SELECT COUNT(*) FROM jugador j LEFT JOIN equipo e  
ON j.equipo = e.id_equipo;
```

El caso de *RIGHT JOIN* es igual salvo que se considera primero la tabla de la derecha.

***STRAIGHT_JOIN*: composición directa**

Es igual que el *JOIN* o reunión interna salvo que hace que se lea primero la primera tabla o más a la izquierda ya que en ocasiones el gestor lo puede hacer al revés y hacer la consulta más lenta o ineficiente. Esto es especialmente útil para la optimización de consultas.

En definitiva trabajar con dos o más tablas no difiere de trabajar con una, siempre que hayamos tenido cuidado de evitar registros falsos o espurios. El resto de operaciones son exactamente las mismas que vimos para el caso de una tabla.

ACTIVIDADES 3.4



➤ Obtenga mediante consultas la siguiente información sobre la base de datos *liga*:

- a. Nombre de jugador, nombre de equipo y puesto del mismo.
- b. Datos de equipo y número de partidos que han jugado como locales.
- c. Datos de equipos con más de tres jugadores registrados.
- 20/11* d. Repita la consulta anterior usando la cláusula *EXISTS*.
- e. Nombre de todos los equipos y datos de sus partidos como locales en caso de haberlos.
- f. Datos de equipos y salario máximo entre sus jugadores.
- g. Datos del partido con mayor puntuación.
- h. Nombre y número de victorias de cada equipo.
- i. Nombre de equipo, nombre de su capitán para cada equipo.
- j. Explique desde el punto de vista de la optimización las diferencias entre usar *EXISTS* en una consulta o usar combinación de tablas.

3.5.2 OPERACIONES DE UNIÓN/INTERSECCIÓN/DIFERENCIA

SQL soporta las tres operaciones del álgebra relacional sobre dos conjuntos de resultados o relaciones. Para ello, ambas relaciones deben tener el mismo número de campos y dominios compatibles.

Sin embargo, aunque gestores como Oracle y SQL Server soportan todos ellos en MySQL solamente se implementa la unión. No obstante se pueden lograr la intersección y diferencia indirectamente como veremos.

Unión

En ocasiones podemos requerir que nuestros datos estén divididos en varias tablas. Por ejemplo, una empresa que vende productos deportivos podría tener en su base de datos una tabla para cada categoría de sus productos (atletismo, baloncesto, etc.). Aunque cada categoría tiene sus propios atributos, es normal que comparten algunos de ellos como el identificador, nombre, precio etc. De este modo si queremos mostrar los datos comunes de todos ellos una unión sería el tipo de consulta más apropiado (aunque hay otras alternativas como la combinación).

Para efectuar una unión todas las consultas involucradas deben tener el mismo número y tipo de campos.

Los nombres de columna usados por el primer *SELECT* se usan como nombres de columna para los resultados retornados.

Sintaxis de *UNION*

```
SELECT ...
UNION [ALL | DISTINCT]
SELECT ...
[UNION [ALL | DISTINCT]
SELECT ...]
```

Salvo que usemos la cláusula *ALL*, todos los registros devueltos son únicos, como si hubiéramos hecho un *DISTINCT* para el conjunto de resultados total. Si especificamos *ALL*, obtenemos todos los registros coincidentes de todos los comandos *SELECT* usados.

Aunque podemos unir resultados ordenados por cada *SELECT* si preferimos mostrar todos los valores ordenados podemos usar *ORDER BY* y *LIMIT* como en el siguiente ejemplo.



EJEMPLO 3.35

Mostrar, ordenados por nombre, los nombres de jugadores de los equipos 1 y 2:

```
(SELECT nombre FROM jugador WHERE equipo=1)
UNION
(SELECT nombre FROM jugador WHERE equipo=2)
ORDER BY 1 LIMIT 10;
```

Intersección

Es una operación entre dos conjuntos cuyo resultado es el conjunto de elementos comunes en ambos.

Como ya hemos señalado no existe una cláusula en MySQL para obtener la intersección de dos conjuntos de datos pero se puede hacer usando combinaciones como en el siguiente ejemplo.



EJEMPLO 3.36

Obtener el listado de equipos que han jugado como locales y visitantes:

Si lo hiciésemos con otro gestor como PostgreSQL:

```
SELECT local FROM equipo  
INTERSECT  
SELECT visitante FROM equipo;
```

En MySQL podemos usar una combinación:

```
SELECT local FROM equipo a, equipo b WHERE a.local=b.visitante;
```

En este ejemplo realizamos una combinación de una tabla consigo misma (también llamada combinación reflexiva) filtrando aquellas filas que cumplen la condición de que el equipo local de la tabla a coincide con el visitante de la tabla b.

Diferencia

Es la operación entre dos conjuntos cuyo resultado es el conjunto de elementos que son distintos.

En otros gestores la cláusula SQL es *EXCEPT* (PostgreSQL) o *MINUS* (SQL Server).



EJEMPLO 3.37

Obtener todos los nombres de jugadores del equipo 1 que no coincidan con ningún nombre del equipo 2.

En PostgreSQL:

```
SELECT nombre FROM equipo WHERE id_equipo=1  
EXCEPT  
SELECT nombre FROM equipo WHERE id_equipo=2;
```

En MySQL tenemos dos posibilidades, una subconsulta y una combinación:

```
SELECT nombre FROM equipo WHERE id_equipo=1 AND nombre NOT IN (SELECT nombre FROM  
equipo WHERE id_equipo=2);  
SELECT DISTINCT a.nombre  
FROM jugador a LEFT JOIN jugador b USING a.nombre  
WHERE (a.equipo=1 OR a.equipo=2)  
AND (b.equipo=1 OR b.equipo=2)  
AND b.nombre IS NULL;
```

En este último caso obtenemos una combinación izquierda de todos los jugadores del equipo 1 combinados con todos los del equipo 2. Aquellos registros no coincidentes aparecerán con el campo *nombre* de la segunda tabla como nulos.



RESUMEN DEL CAPÍTULO

En este capítulo hemos estudiado cómo acceder a los datos de nuestras bases de datos mediante consultas usando la cláusula *SQL SELECT* con toda su complejidad, pasando desde consultas simples sobre una tabla a consultas complejas que implican el uso de más tablas (mediante reuniones, uniones o subconsultas), funciones y expresiones regulares para comparación de patrones.

En este caso, hemos utilizado el SGBD MySQL por su facilidad de uso y para proporcionar una interfaz gráfica de usuario (MySQL Workbench) muy amigable y eficiente.



EJERCICIOS PROPUESTOS

- 1. Obtenga la siguiente información sobre la base de datos *liga*:
 - Obtener datos de todos los jugadores menos los de los equipos uno, dos y tres.
 - Obtener el número de ciudades en las que hay equipos.
 - Listado de partidos ordenado por equipo, local y fecha.
 - Número de partidos ganados por equipos locales.
 - Nombres de jugadores que empiecen por “A” y tengan al menos 2 vocales.
 - Datos del último partido, incluyendo el nombre de los equipos y jugadores.
 - Datos del equipo y del capitán para equipos que hayan ganado más de 2 partidos como visitantes.
 - Realizar una consulta para mostrar los equipos que no han jugado ningún partido como locales.

- 2. Obtenga la siguiente información sobre la base de datos *motorblog*:
 - Datos de noticias que contengan una dirección IP.
 - Mostrar los títulos de las noticias invertidos y en mayúsculas.
 - Mostrar los enlaces sin el prefijo *http://www*.
 - Datos de noticias que contengan la palabra “Alonso”.
 - Datos de la noticia con más comentarios.
 - Mostrar el titular de cada noticia con su antigüedad en segundos. Añadir también antigüedad en meses, días, horas, minutos y segundos.
 - Mostrar datos de noticias que no contengan direcciones web.



TEST DE CONOCIMIENTOS



1 Indique la opción cierta:

- a) Una consulta debe incluir como mínimo las cláusulas *SELECT* y *FROM*.
- b) Una consulta debe incluir como mínimo las cláusulas *SELECT*, *FROM* y *WHERE*.
- c) Una consulta debe incluir como mínimo las cláusulas *SELECT*, *ORDER BY* y *FROM*.
- d) Una consulta debe incluir como mínimo la cláusula *SELECT*.

2 ¿Qué es cierto respecto a la cláusula *HAVING*?

- a) Significa tener.
- b) Sirve para poner un filtro sobre filas de las tablas.
- c) Permite poner un filtro sobre grupos de valores.
- d) Se usa solo con funciones de agregado.

3 Una consulta correlacionada:

- a) Relaciona campos de dos o más tablas.
- b) Se da cuando el resultado de la consulta secundaria o subconsulta depende de los valores de la tabla principal.
- c) Es igual que una reunión externa.
- d) Es igual que una reunión interna.

4 ¿Cuál de las siguientes afirmaciones es cierta?

- a) Toda consulta correlacionada se puede resolver con una combinación de tablas.
- b) Solo algunas consultas correlacionadas pueden resolverse con una combinación.
- c) Las consultas correlacionadas no pueden, en ningún caso, resolverse con una combinación.

5 ¿Qué es el producto cartesiano?

- a) La combinación de filas coincidentes de dos tablas.
- b) La combinación de cada elemento de un conjunto con todos los elementos de otro conjunto.
- c) Lo que hacemos cuando hacemos un *JOIN* en una consulta.
- d) Todo lo anterior.

6 Si hacemos el producto cartesiano de tres tablas de 4, 5 y 10 filas, la tabla resultante, ¿cuántas filas tendrá?

- a) 1.000.
- b) 2.000.
- c) 300.
- d) 200.

7 El resultado de una consulta:

- a) Solo existe en memoria cuando se realiza.
- b) Se almacena en disco.
- c) Se encuentra siempre en la caché.
- d) Es una tabla con datos filtrados.

8 Una vista:

- a) Es lo mismo que una consulta pero con los datos en disco.
- b) Es lo que tenemos cuando guardamos una consulta.
- c) Es una tabla virtual cuya definición es una consulta.
- d) Es la definición de las tablas de la base de datos.