Base de Datos

5. Consultas de Selección Avanzadas

Curso 2022-2023

Tabla de contenido

Una vez conocemos los tipos de datos que facilita el SGBD y sabemos utilizar y a 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 sus 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...), como también 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 en otras consultas.

1. EXCLUSIÓN DE FILAS REPETIDAS. OPCIÓN DISTINCT

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

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

La sintaxis es la siguiente:

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

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

Ejemplo de la necesidad de utilizar la opción DISTINCT

Como ejemplo en que hay que utilizar la opción veamos cómo 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 "Codi" FROM empleado;
```

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 debemos utilizar la opción distinct, ya que de lo contrario un mismo departamento aparecería tantas veces como empleados tuviera asignados.

2. COMBINACIONES ENTRE TABLAS

La cláusula FROM efectúa el producto cartesiano de todas las tablas que aparecen en la cláusula. El producto cartesiano no nos interesará casi nunca, sino únicamente un subconjunto del mismo.

Los tipos de subconjuntos que nos pueden interesar coinciden con los resultado de las combinaciones ***join***, ***inner-join***, ***natural-join*** y ***outer-join***.

Operaciones para combinar tablas

Dadas dos tablas R y S, se define el ***join*** de R 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 **inner-join** es un **join** en el que el operador es la igualdad. Se escribe R[A=Z]S.

El ***natural-join*** es un **inner-join** en el 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.

En ocasiones, es necesario tener el resultado del ***inner-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 mesa.

Dadas dos relaciones R y S, se define el ***left-outer-join*** de R 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 ningún 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 R 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 ningún tupla de R según el atributo A, que presentan valores NULL en los atributos provenientes de R.

También podemos considerar el ***full-outer-join*** de R 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*** . Recordamos que la unión de conjuntos no contempla 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 de los diversos SGBD comerciales existentes: las combinaciones según la norma SQL-87 y las combinaciones según la norma SQL-92.

2.1. 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 lograr añadiendo, en la cláusula, los filtros correspondientes a las combinaciones entre las columnas de las tablas a 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 empleado.emp_no  AS "Codi Empleado", empleado.nombre AS "Empleado",
  empleado.dept_no  AS "Codi Departamento", dnom AS "Descripción"
FROM empleado, departamento
WHERE empleado.dept_no  = departamento.dept_no ;
```

Fijémonos que el acceso a la tabla DEPARTAMENTO es necesario para conseguir el nombre del departamento. Tengamos en cuenta, también, que, como el campo depto de la tabla EMPLEADO 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 DEPARTAMENTO. Así, pues, una vez ejecutado el filtro de la cláusula where*, todas las filas de la tabla EMPLEADO estarán combinadas con una fila de la tabla DEPARTAMENTO.

El resultado obtenido es éste:

```
Codi Empleado       Empleado    Codi 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 inner-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 el año 1987, el estándar SQL-ISO propuso (quizá 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 hubiera que combinar que indicara la necesidad de hacer aparecer todas las filas de una tabla (por ejemplo R), aunque 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 tabla (S) para la que hay que generar valores nulos.

Es decir, un **left-outer-join** de la tabla R con la tabla S según las columnas A (tabla R) 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 R con la tabla S según las columnas A (tabla R) 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 una 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 una ***full-outer-join*** pasa por una operación entre una sentencia con 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***.

2.2. 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 lograr nuestro propósito, desde la revisión del 92 (SQL-92), el lenguaje SQL facilita las operaciones *join* en la cláusula indicando la condición de combinación, la cual ya no deberá indicarse (excepto en un caso) dentro de la cláusula.

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

```
...
FROM tabla1, tabla2
WHERE <condició combinació entre tabla1 y tabla2>
```

```
...
```

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

```
...
FROM tabla1 [ INNER | LEFT | RIGHT ] JOIN tabla2
ON <condició combinació entre tabla1 y tabla2>
WHERE ...
```

Como se ve en la sintaxis anterior, hay diferentes opciones de *join*.

Los SGBD relacionales actuales (MySQL, Oracle, SQLServer, PostgreSQL, MS-Access...) incorporan las combinaciones, y con 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 es la más común y, de hecho, es la que se ejecuta si no se indica ninguna de las opciones. Se llama *combinación interna* y combina filas de dos tablas siempre que haya valores coincidentes en el campo o campos de combinación.

Ejemplo de combinación iner 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 empleado.emp_no  AS "Codi Empleado", empleado.nombre AS
"Empleado",
   empleado.dept_no  AS "Codi Departamento", dnom AS "Descripción"
FROM empleado INNER JOIN departamento ON empleado.dept_no  =
departamento.dept_no ;
```

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

Las combinaciones **left join**, **right join** y **full 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** del **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 **full 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 d.dept_no  AS "Codi", dnom AS "Departamento",
    MAX(salario) AS "Salario más alto "
FROM departamento d LEFT JOIN empleado e ON d.dept_no  = e.dept_no
GROUP BY d.dept_no, dnom
ORDER BY 1;
```

El resultado obtenido es éste:

```
Codi       Departamento    Salario mas 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 puede convertirse en un **right join** si cambiamos el orden de las tablas en la cláusula :

```
SELECT d.dept_no  AS "Codi", dnom AS "Departamento",
   MAX(salario) AS "Mejor salario"
FROM empleado e RIGHT JOIN departamento d ON e.dept_no  = d.dept_no
GROUP BY d.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

Quieren mostrarse, 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 "Codi", nombre AS "Empleado", Cliente_cod AS
"Cliente", nom AS "Razón social"
FROM empleado LEFT JOIN Cliente ON emp_no  = repr_cod
ORDER BY 1,2;
```

O también:

```
SELECT emp_no  AS "Codi", nombre AS "Empleado", Cliente_cod AS
"Cliente", nom AS "Razón social"
FROM Cliente RIGHT JOIN empleado ON repr_cod = emp_no
ORDER BY 1,2;
```

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

```
Codi       Empleado    Cliente     Razón social
---------- ---------- ---------- ---------------------------------------
7369       ºSANCHEZ
7499       ARROYO     107        WOMEN SPORTS
7499       ARROYO     104        EVERY MOUNTAIN
7521       SALA       101        TKB SPORT SHOP
7521       SALA       103        JUST TENNIS
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 uno, ya que de lo contrario los empleados que no tienen asignado ningún cliente 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 Cliente_cod AS "Cliente", nom AS "Razón social", emp_no  AS
"Codi", nombre AS "Empleado"
FROM Cliente LEFT JOIN empleado ON repr_cod = emp_no ;
```

O también:

```
SELECT Cliente_cod AS "Cliente", nom AS "Razón social", emp_no  AS
"Codi", nombre AS "Empleado"
FROM empleado RIGHT JOIN Cliente ON emp_no  = repr_cod;
```

El resultado que se obtiene es este:

```
Cliente     Razón social                                    Codi       Empleado
---------- ---------------------------------------------- ---------- ---------
107        WOMEN SPORTS                                   7499       ARROYO
104        EVERY MOUNTAIN                                 7499       ARROYO
106        SHAPE UP                                       7521       SALA
103        JUST TENNIS                                    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, hay que utilizar un **outer-join**, ya que de lo contrario los clientes que no tienen asignado representante no aparecerían.

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

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

La instrucción para alcanzar el objetivo es esta:

```
SELECT Cliente_cod AS "Cliente", nom AS "Razón social", emp_no  AS
"Codi", nombre AS "Empleado"
FROM Cliente LEFT JOIN empleado ON emp_no  = repr_cod
UNION
SELECT Cliente_cod AS "Cliente", nom AS "Razón social", emp_no  AS
"Codi", nombre AS "Empleado"
FROM Cliente RIGHT JOIN empleado ON emp_no  = repr_cod;
```

El resultado obtenido es éste:

```
Cliente     Razón social                                    Codi       Empleado
---------- ---------------------------------------------- ---------- ---------
-
100        JOCKSPORTS                                     7844       TOVAR
101        TKB SPORT SHOP                                 7521       SALA
102        VOLLYRITE                                      7654       MARTIN
103        JUST TENNIS                                    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, para los casos en que las columnas a combinar tengan coincidencia de nombres:

- Si la combinación se quiere efectuar para todas las columnas que tengan nombres coincidentes en las tablas a combinar, disponemos de las combinaciones **naturales**. 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 **naturales** 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 sentencia, no debe indicarse 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 a combinar, disponemos de las combinaciones **join** con la opción. En este caso, la opción provoca que se efectúe la combinación entre las tablas para las columnas indicadas, sin tener que indicar la condición de combinación. Como en los **naturales**, 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 emprendida, muestra los Empleados (código y número) junto con el código y nombre del Departamento al que pertenecen.

```
SELECT empleado.emp_no  AS "Codi Empleado", empleado.nombre AS
"Empleado",
   empleado.dept_no  AS "Codi Departamento", dnom AS "Descripción"
FROM empleado INNER JOIN departamento ON empleado.dept_no  =
departamento.dept_no ;
```

O bién:

```
SELECT empleado.emp_no  AS "Codi Empleado", empleado.nombre AS
"Empleado",
   empleado.dept_no  AS "Codi Departamento", dnom AS "Descripción"
FROM empleado JOIN departamento ON empleado.dept_no  =
departamento.dept_no ;
```

Puesto que en las tablas y hay coincidencia de nombre para la columna de combinación **deptno** y no hay otras columnas con nombres coincidentes, podemos utilizar un **natural join**:

```
SELECT empleado.emp_no  AS "Codi Empleado", empleado.nombre AS
"Empleado",
   dept_no  AS "Codi Departamento", dnom AS "Descripción"
FROM empleado NATURAL JOIN departamento;
```

Fijémonos que, al visualizar la columna **dept_no**, 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 empleado.emp_no  AS "Codi Empleado", empleado.nombre AS
"Empleado",
   dept_no  AS "Codi Departamento", dnom AS "Descripción"
FROM empleado INNER JOIN departamento USING (dept_no);
```

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

```
SELECT empleado.emp_no  AS "Codi Empleado", empleado.nombre AS
"Empleado",
   dept_no  AS "Codi Departamento", dnom AS "Descripción"
FROM empleado JOIN departamento USING (dept_no);
```

Ejemplo de simplificació de combinacions left join y right join

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

```
SELECT d.dept_no  AS "Codi", dnom AS "Departamento",
   MAX(salario) AS "Salario más alto'"
FROM departamento d LEFT JOIN empleado e ON d.dept_no  = e.dept_no
GROUP BY d.dept_no, dnom
ORDER BY 1;
```

```
SELECT d.dept_no  AS "Codi", dnom AS "Departamento",
   MAX(salario) AS "Salario más alto'"
FROM empleado e RIGHT JOIN departamento d ON e.dept_no  = d.dept_no
GROUP BY d.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, habíamos podido utilizar un **natural join**:

```
SELECT dept_no  AS "Codi", dnom AS "Departamento",
   MAX(salario) AS "Salario más alto'"
FROM departamento NATURAL LEFT JOIN empleado e
GROUP BY dept_no, dnom
ORDER BY 1;

SELECT dept_no  AS "Codi", dnom AS "Departamento",
   MAX(salario) AS "Salario más alto'"
FROM empleado e NATURAL RIGHT JOIN departamento d
GROUP BY dept_no, dnom
ORDER BY 1;
```

Y también, utilizando la opción :

```
SELECT dept_no  AS "Codi", dnom AS "Departamento",
   MAX(salario) AS "Salario más alto'"
FROM departamento LEFT JOIN empleado e USING (dept_no)
GROUP BY dept_no, dnom
ORDER BY 1;

SELECT dept_no  AS "Codi", dnom AS "Departamento",
   MAX(salario) AS "Salario más alto'"
FROM empleado e RIGHT JOIN departamento d USING (dept_no)
GROUP BY dept_no, dnom
ORDER BY 1;
```

3. 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 de otra sentencia SELECT. La subconsulta se cierra entre paréntesis y no incluye el punto y coma finales.

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

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

En el esquema empresa, se pide mostrar a 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 "Codi", nombre AS "Empleado", salario AS "Salario"
FROM empleado
WHERE salario >= (SELECT avg(salario) FROM empleado)
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 de la subconsulta, salvo 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 a los empleados de cada Departamentoo que tienen un salario menor que el salario medio del mismo Departamentoo.

La instrucción para alcanzar el objetivo es esta:

```
SELECT dept_no  AS "Departamento.", emp_no  AS "Codi", nombre AS
"Empleado",
   salario AS "Salario"
FROM empleado e1
WHERE salario >= (SELECT avg(salario)
       FROM empleado e2
       WHERE dept_no =e1.dept_no
             )
ORDER BY 1, 4 DESC,2;
```

Los valores devueltos por las subconsultas se utilizan en las cláusulas como parte derecha de operaciones de comparaciones en las que intervienen los operadores:

```
=,!=,<,< =,>,> =,[NOT] IN,%%<op>%% any y %%<op>%% ALL
```

Las subconsultas también pueden vincularse a la consulta contenedora por la partícula [**not**] **exist**:

```
...
WHERE [NOT] EXISTS (subconsulta)
```

En este caso, la subconsulta acostumbra a 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 pueden actuar como subconsultas en expresiones en las que el valor resultado se compara con cualquier operador de comparación.

IMPORTANTE

Las consultas que pueden dar como resultado más de un valor (aunque en ejecuciones concretas sólo den uno) nunca pueden actuar como subconsultas en expresiones en las 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 deben compararse, en el mismo orden que los valores devueltos por la subconsulta, separados por comas y cerrados 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 cargo que el cargo que tiene el empleado de apellido 'ALONSO'.

La instrucción para alcanzar el objetivo parece que podría ser esta:

```
SELECT nombre AS "Empleado"
FROM empleado
WHERE cargo = (SELECT cargo
      FROM empleado
      WHERE UPPER(nombre)='ALONSO')
  AND UPPER(nombre)!='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, deberemos buscar otro operador de comparación para evitar este problema:

```
SELECT nombre AS "Empleado"
```

```
FROM empleado
WHERE cargo IN (SELECT cargo
        FROM empleado
        WHERE UPPER(nombre)='ALONSO')
  AND UPPER(nombre)!='ALONSO';
```

O también:

```
SELECT nombre AS "Empleado"
FROM empleado e
WHERE EXISTS (SELECT *
       FROM empleado
       WHERE UPPER(nombre)='ALONSO'
        AND cargo=e.cargo
            )
  AND UPPER(nombre)!='ALONSO';
```

Ejemplo de utilización de los operadores ANY y EXISTS

En el esquema empresa, se pide mostrar los nombres y oficios de los empleados del Departamentoo 20 cuyo trabajo coincida con el de algún empleado del Departamentoo de 'VENTAS'.

La instrucción para alcanzar el objetivo puede ser esta:

```
SELECT nombre AS "Empleado", cargo AS "Cargo"
FROM empleado
WHERE dept_no =20
 AND cargo =ANY (SELECT cargo
        FROM empleado
        WHERE dept_no  =ANY (SELECT dept_no
                FROM departamento
                WHERE UPPER(dnom)='VENTAS'
               )
       );
```

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

```
SELECT nombre AS "Empleado", cargo AS "Cargo"
FROM empleado
WHERE dept_no =20
  AND cargo =ANY (SELECT cargo
        FROM empleado
        WHERE dept_no  = (SELECT dept_no
              FROM departamento
              WHERE UPPER(dnom)='VENTAS'
              )
       );
```

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

```
SELECT nombre AS "Empleado", cargo AS "Cargo"
```

```
FROM empleado e
WHERE dept_no =20
  AND EXISTS (SELECT *
      FROM empleado, departamento
      WHERE empleado.dept_no =departamento.dept_no
       AND UPPER(dnom)='VENTAS'
       AND cargo=e.cargo
     );
```

Ejemplo de utilización del operador IN

En el esquema empresa, es demana mostrar els Empleados amb el mateix cargo y salario que 'JIMÉNEZ'.

La instrucción para alcanzar el objetivo puede ser esta:

```
SELECT emp_no  "Codi", nombre "Empleado"
FROM empleado
WHERE (cargo,salario) IN (SELECT cargo, salario
          FROM empleado
          WHERE UPPER(nombre)='JIMÉNEZ'
         )
  AND UPPER(nombre)!='JIMÉNEZ';
```

Ejemplo de condición compleja de filtrado con diversas subconsultas y operaciones

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

```
SELECT emp_no  "Codi", nombre "Empleado"
FROM empleado
WHERE (cargo IN (SELECT cargo
       FROM empleado
       WHERE UPPER(nombre)='JIMÉNEZ'
      )
   AND UPPER(nombre)!='JIMÉNEZ'
  )
  OR (salario>= (SELECT salario
        FROM empleado
        WHERE UPPER(nombre)='FERNÁNDEZ'
      )
   AND UPPER(nombre)!='FERNÁNDEZ'
  );
```

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 cómo quedarnos con las columnas interesantes (cláusula **SELECT**), hay que ver cómo agrupar las filas seleccionadas y cómo filtrar por condiciones sobre los grupos .

La cláusula **GROUP BY** permite agrupar las filas resultado de las cláusulas **SELECT**, **FROM** y **WHERE** según una o más de las columnas seleccionadas. La cláusula **HAVING** permite especificar condiciones de filtrado sobre los grupos alcanzados por la cláusula **GROUP BY**.

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

SELECT [**DISTINCT**] <expresión/columna>, <expresión/columna>,...

FROM <tabla>, <tabla>,...

[**WHERE** <condición_de_búsqueda>]

[**GROUP BY** <alias/columna>, <alias/columna>,...]

[**HAVING** <condición_sobre_grupos>]

[**ORDER BY** <expresión/columna> [ASC|**DESC**], <expresión/columna> [**ASC**|**DESC**],...];

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

Funciones de agrupamiento más importantes que se pueden utilizar en las sentencias SELECT de agrupamiento de filas:

- **COUNT ([* o expr])** : devuelve el número total de filas seleccionadas por la consulta.
- **MIN (expr)**: devuelve el valor mínimo del campo que especifiquemos (expr)..
- **MAX (expr):** devuelve el valor máximo del campo que especifiquemos (expr).
- **SUM (expr):** suma los valores del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.
- **AVG(n)**: devuelve el valor promedio del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.

La expresión sobre la que se calculan las funciones de agrupamiento puede ir precedida de la opción **DISTINCT** para indicar que se evalúe sobre los valores diferentes de la expresión, o de la opción ajo para indicar que se evalúe sobre todos los valores de la expresión. El valor predeterminado es ajo.

Una sentencia **SELECT** es una sentencia de selección de conjuntos cuando aparece la cláusula **GROUP BY** o la cláusula **having** o una función de agrupamiento; es decir, una sentencia **SELECT** puede ser sentencia de selección de conjuntos aunque no exista

cláusula **GROUP BY**, en cuyo caso se considera que hay un único conjunto formado por todas las filas seleccionadas.

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

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

En el esquema empresa, se quiere contar cuántos empleados existen.

La instrucción para alcanzar el objetivo es ésta:

SELECT COUNT(*) AS " Cantidad empleados " FROM empleado;

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 poder contarlas.

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

En el esquema ***empresa,*** se quiere contar cuántos oficios distintos existen.

La instrucción para alcanzar el objetivo es ésta:

SELECT COUNT(DISTINCT cargo) AS "Cuántos oficios" FROM empleado;

En este caso es necesario indicar la opción **distinct**, ya que de lo contrario contaría todas las filas que tienen algún valor en la columna cargo, 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 existen de cada cargo.

La instrucción para alcanzar el objetivo es ésta:

SELECT cargo AS “Cargo", COUNT(*) AS "Cantidad empleados"

FROM empleado GROUP BY oficina;

El resultado obtenido es éste:

```
Oficina      Cantidad 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 empleado
        GROUP BY dept_no
        ORDER BY MAX(salario);
```

O también:

```
SELECT dept_no  AS "Codi", MAX(salario) AS "Salario Máximo"
        FROM empleado
        GROUP BY dept_no
        ORDER BY "Salario Máximo";
```

O también:

```
SELECT dept_no  AS "Codi", MAX(ALL salario) AS "Salario Máximo"
        FROM empleado
        GROUP BY dept_no
    ORDER BY "Salario Máximo";
```

Ejemplo de coexistencia de las cláusulas GROUP BY y ORDER BY

En el esquema empresa, se quiere contar cuántos empleados de cada cargo 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 "Codi", Oficina AS " Oficina ",
   COUNT(*) AS " Cantidad Empleados"
        FROM empleado
        GROUP BY dept_no, Oficina
    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 cargo hay en el departamento 20.

La instrucción para alcanzar el objetivo es esta:

```
SELECT Oficina AS " Oficina ", COUNT(*) AS " Cantidad Empleados"
        FROM empleado
        WHERE dept_no =20
        GROUP BY Oficina;
```

Ejemplo de utilización de la cláusula HAVING

En el esquema empresa, se quiere mostrar el número de empleados de cada cargo que hay para los oficios que tienen más de un empleado.

La instrucción para alcanzar el objetivo es esta:

```
SELECT Oficina AS " Oficina ", COUNT(*) AS " Cantidad Empleados"
        FROM empleado
        GROUP BY Oficina
        HAVING COUNT(*)>1;
```

5. UNIÓN, INTERSECCIÓN Y DIFERENCIA DE SENTENCIAS SELECT

El lenguaje SQL permite efectuar operaciones sobre los resultados de las sentencias con el fin de 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 intervinientes– dos a dos.

5.1. UNIÓN DE SENTENCIAS SELECT

El lenguaje SQL proporciona el operador para combinar todas las filas del resultado de una sentencia con todas las filas del resultado de otra sentencia, y elimina cualquier duplicación de filas que se pudiera producir en el conjunto resultante.

La sintaxis es esta:

```
Sentencia_Select_Sin_Order_By
      UNION
Sentencia_Select_Sin_Order_By
      [ORDER BY ...]
```

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

Ejemplo de la operación union entre sentencias SQL

En el esquema sanidad, se quiere presentar el personal que trabaja en cada hospital, incluyendo al personal de la plantilla y a los doctores, y mostrando el cargo que ejercen en él.

Una posible instrucción para alcanzar el objetivo es esta:

```
SELECT nom AS "Hospital", 'Doctor' AS "Cargo", doctor_no AS "Codi",
  nombre "Empleado"
        FROM hospital, doctor
        WHERE hospital.hospital_cod=doctor.hospital_cod
UNION
SELECT nom, funcio, Emp_no, nombre
        FROM hospital, plantilla
        WHERE hospital.hospital_cod=plantilla.hospital_cod
        ORDER BY 1,2;
```

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

```
Hospital   Cargo      Codi       Empleado

---------- ---------- ---------- ---------------

General    Doctor     585        Miller G.

General    Doctor     982        Cajal R.

General    Intern     6357       Karplus W.

La Paz     Doctor     386        Cabeza D.
```

```
La Paz     Doctor     398        Best K.

La Paz     Doctor     453        Galo D.

La Paz     Infermer   8422       Bocina G.

La Paz     Infermera  1009       Higueras D.

La Paz     Infermera  6065       Rivera G.

La Paz     Infermera  7379       Carlos R.

La Paz     Intern     9901       Adams C.

Provincial Doctor     435        López A.

Provincial Infermer   3106       Hernández J.

Provincial Infermera  3754       Díaz B.

San Carlos Doctor     522        Adams C.

San Carlos Doctor     607        Nico P.

San Carlos Infermera  8526       Frank H.

San Carlos Intern     1280       Amigó R.

18 rows selected
```

5.2. INTERSECCIÓN Y DIFERENCIA DE SENTENCIAS SELECT

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

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

```
Sentencia_select_sin_order_by
    INTERSECT
Sentencia_select_sin_order_by
    [ORDER BY ...]
```

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

```
Sentencia_Select_Sin_Order_By
    MINUS
Sentencia_Select_Sin_Order_By
  [ORDER BY ...]
```

6. FUNCIONES INCORPORADAS EN MYSQL

Funciones de grupo

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 mayor valor 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 facilitadas 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.

6.1. FUNCIONES MATEMÁTICAS

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

Función o operador matemático	Descripción	Ejemplos
ABS(x)	Retorna el valor absoluto de x	
ACOS(x)	Retorna el arco-coseno de x	SELECT ACOS(1);
ASIN(x)	Retorna arco-seno de x	SELECT ASIN(0.2);
ATAN(x), ATAN2(x,y)	Retorna arco-tangente de x y arco-tangente de les coordenadas (x,y), respectivamente	SELECT ATAN(2); SELECT ATAN(-2,2);
CEIL(x)	Retorna el valor entero inmediatamente superior a x, o x si es un valor entero	SELECT CEILING(1.23); Retorna 2. SELECT CEILING(3); Retorna 3
CEILING(x)	Sinónimo de CEIL(x)	
CONV(x,b1,b2)	Convertir el nombre x, considera expresado en base b1, a base b2.	SELECT CONV(10,10,2); Retorna 1010
COS(x)	Retorna el coseno de x	
COT(x)	Retorna la cotangente de x	
CRC32(x)	Calcula el valor CRC32 (cíclica redundancia check)	

DEGREES(x)	Convertir x, expresado en radian, a grados	SELECT DEGREES(PI());
DIV	Calcula la división entera	SELECT 5 DIV 2; Retorna 2.
/	Operador de división	SELECT 3/5; Retorna: 0.60
EXP(x)	Calcula e elevado a x	
FLOOR(x)	Retorna el valor entero inmediatamente inferior a x. O x, si es un valor entero.	SELECT FLOOR(-1.23); Retorna: -2.
LN(x)	Retorna el logaritmo de base e de x	
LOG10(x)	Retorna el logaritmo de base 10 de x	
LOG2(x)	Retorna el logaritmo de base 2 de x	
LOG(b,x)	Retorna el logaritmo de base b de x	
-	Operador resta	SELECT 5-3;
MOD(x,y)	Retorna la resta de la división de x y y. Es innervalente a N % M, y també a N MOD M	SELECT MOD(29,9); Retorna: 2
OCT(x)	Retorna la representación octal del nombre x, expresado en decimal.	
PI()	Retorna el valor pi	
+	Operador de suma	SELECT 5+3;
POW(x,y)	Retorna x elevado a y	
POWER(x,y)	Sinónimo de POW	
RADIANS(x)	Retorna x, expresado en grados, a radianes	
RAND()	Retorna un valor real aleatorio entre 0 y 1	
ROUND(x)	Redondea x al número entero más cercano.	SELECT ROUND (1.9); Retorna 2. SELECT ROUND(1.5); Retorna 2. SELECT ROUND (1.2); Retorna 1.
SIGN(x)	Retorna -1 si x es negativo. Retorna 1 si x es positivo. y retorna 0 si x = 0	
SIN(x)	Retorna el valor del seno de x	
SQRT(x)	Retorna la raiz cuadrada de x	
TAN(x)	Retorna la tangente de x	
*	Operador de multiplicación	SELECT 5*3;

TRUNCATE(x,d)	Trunca x a d decimales	SELECT TRUNCATE(-1.999,1); Retorna -1.9
-	Operador cambio de signe	SELECT - 2; Retorna: -2

Tabla 1 Funciones matemáticas principales proporcionadas por el SGBD MySQL

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

6.2. FUNCIONES DE CADENAS DE CARACTERES

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

CONVERSIÓN A MAYÚSCULAS/MINÚSCULAS

- **LOWER** o **LCASE** convierte una cadena a minúsculas: **SELECT LOWER**('Hola'); ⇒ hola
- **UPPER** o **UCASE** convierte una cadena a mayúsculas: **SELECT UPPER**('Hola'); ⇒ HOLA

FUNCIONES DE EXTRACCIÓN DE PARTE DE LA CADENA

- **LEFT**(cadena, longitud) extrae varios caracteres del comienzo (la parte izquierda) de la cadena: **SELECT LEFT**('Hola',2); ⇒ Ho
- **RIGHT**(cadena, longitud) extrae varios caracteres del final (la parte derecha) de la cadena: **SELECT RIGHT**('Hola',2); ⇒ la
- **MID**(cadena, posición, longitud), **SUBSTR**(cadena, posición, longitud) o **SUBSTRING**(cadena, posición, longitud) extrae varios caracteres de cualquier posición de una cadena, tantos como se indique en "longitud": **SELECT SUBSTRING**('Hola',2,3); ⇒ ola (Nota: a partir MySQL 5 se permite un valor negativo en la posición, y entonces se comienza a contar desde la derecha -el final de la cadena-)
- **CONCAT** une (concatena) varias cadenas para formar una nueva: **SELECT CONCAT**('Ho', 'la'); ⇒ Hola
- **CONCAT_WS** une (concatena) varias cadenas para formar una nueva, usando un separador que se indique (With Separator): **SELECT CONCAT_WS**('-','Ho','la','Que','tal'); ⇒ Ho-la-Que-tal
- **LTRIM** devuelve la cadena sin los espacios en blanco que pudiera contener al principio (en su parte izquierda): **SELECT LTRIM**(' Hola'); ⇒ Hola
- **RTRIM** devuelve la cadena sin los espacios en blanco que pudiera contener al final (en su parte derecha): **SELECT RTRIM**('Hola '); ⇒ Hola
- **TRIM** devuelve la cadena sin los espacios en blanco que pudiera contener al principio ni al final: **SELECT TRIM**(' Hola '); ⇒ Hola (Nota: realmente, TRIM puede eliminar cualquier prefijo, no sólo espacios; mira el manual de MySQL para más detalles)

FUNCIONES DE CONVERSIÓN DE BASE NUMÉRICA

- **BIN** convierte un número decimal a binario: **SELECT BIN**(10); ⇒ 1010

- **HEX** convierte un número decimal a hexadecimal: **SELECT HEX**(10); ⇒ 'A' (Nota: HEX también tiene un uso alternativo menos habitual: puede recibir una cadena, y entonces mostrará el código ASCII en hexadecimal de sus caracteres: **SELECT HEX**('Hola'); ⇒ '486F6C61')
- **OCT** convierte un número decimal a octal: **SELECT OCT**(10); ⇒ 12
- **CONV**(número,baseInicial,baseFinal) convierte de cualquier base a cualquier base: **SELECT CONV**('F3',16,2); ⇒ 11110011
- **UNHEX** convierte una serie de números hexadecimales a una cadena ASCII, al contrario de lo que hace HEX: **SELECT UNHEX**('486F6C61'); ⇒ 'Hola')

OTRAS FUNCIONES DE MODIFICACIÓN DE LA CADENA

- **INSERT**(cadena,posición,longitud,nuevaCadena) inserta en la cadena otra cadena: **SELECT INSERT**('Hola', 2, 2, 'ADIOS'); ⇒ HADIOSa
- **REPLACE**(cadena,de,a) devuelve la cadena pero cambiando ciertas secuencias de caracteres por otras: **SELECT REPLACE**('Hola', 'l', 'LLL'); ⇒ HoLLLa
- **REPEAT**(cadena,numero) devuelve la cadena repetida varias veces: **SELECT REPEAT**(' Hola',3); ⇒ HolaHolaHola
- **REVERSE**(cadena) devuelve la cadena "del revés": **SELECT REVERSE**('Hola'); ⇒ aloH
- **SPACE**(longitud) devuelve una cadena formada por varios espacios en blanco: **SELECT SPACE**(3); ⇒ " "

FUNCIONES DE INFORMACIÓN SOBRE LA CADENA

- **CHAR_LENGTH** o **CHARACTER_LENGTH** devuelve la longitud de la cadena en caracteres
- **LENGTH** devuelve la longitud de la cadena en bytes
- **BIT_LENGTH** devuelve la longitud de la cadena en bits
- **INSTR**(cadena,subcadena) o **LOCATE**(subcadena,cadena,posInicial) devuelve la posición de una subcadena dentro de la cadena: **SELECT INSTR**('Hola','ol'); ⇒ 2

6.3. FUNCIONES DE GESTIÓN DE FECHAS

- **ADDDATE(fecha, interval expresion):** retorna la fecha agregándole el intervalo especificado. Ejemplos:
    - **adddate**('2006-10-10',**interval** 25 day) retorna "2006-11-04".
    - **adddate**('2006-10-10',**interval** 5 month) retorna "2007-03-10".
    - **adddate**(fecha, dias): retorna la fecha agregándole a fecha "dias". Ejemplo:
    - **adddate**('2006-10-10',25), retorna "2006-11-04".
- **ADDTIME(expresion1,expresion2):** agrega expresion2 a expresion1 y retorna el resultado.
- **CURRENT_DATE:** retorna la fecha de hoy con formato "YYYY-MM-DD" o "YYYYMMDD".
- **CURRENT_TIME:** retorna la hora actual con formato "HH:MM:SS" o "HHMMSS".
- **DATE_ADD(fecha,interval expresion tipo) y DATE_SUB(fecha,interval expresion tipo):** el argumento "fecha" es un valor "date" o "datetime", "expresion" especifica el valor de intervalo. Ejemplos;

  - **date_sub**('2006-08-10 18:55:44', interval 2 minute) retorna "2006-08-10 18:53:44";
  - **date_sub**('2006-08-10 18:55:44', interval '2:3' minute_second) retorna "2006-08-10 18:52:41".

Los valores para "tipo" pueden ser: **second**, **minute**, **hour**, **day**, **month**, **year**, **minute_second** (minutos y segundos), **hour_minute** (horas y minutos), **day_hour** (días y horas), **year_month** (año y mes), hour_second (hora, minuto y segundo), **day_minute** (dias, horas y minutos), **day_second**(dias a segundos).

- **datediff(fecha1,fecha2):** retorna la cantidad de días entre fecha1 y fecha2.
- **dayname(fecha)**: retorna el nombre del día de la semana de la fecha. Ejemplo:
  - **dayname**('2006-08-10') retorna "thursday".
- **dayofmonth(fecha):** retorna el día del mes para la fecha dada, dentro del rango 1 a 31. Ejemplo:
  - **dayofmonth**('2006-08-10') retorna 10.
- **dayofweek(fecha):** retorna el índice del día de semana para la fecha pasada como argumento. Los valores de los índices son: 1=domingo, 2=lunes,... 7=sábado). Ejemplo:
  - **dayofweek**('2006-08-10') retorna 5, o sea jueves.
- **dayofyear(fecha):** retorna el día del año para la fecha dada, dentro del rango 1 a 366. Ejemplo:
  - **dayofmonth**('2006-08-10') retorna 222.
- **extract(tipo from fecha): extrae partes de una fecha.**

Ejemplos:

  - **extract**(year from '2006-10-10'), retorna "2006".
  - **extract**(year_month from '2006-10-10 10:15:25') retorna "200610".
  - **extract**(day_minute from '2006-10-10 10:15:25') retorna "101015";

Los valores para tipo pueden ser: **second**, **minute**, **hour**, **day**, **month**, **year**, **minute_second**, **hour_minute**, **day_hour**, **year_month**, **hour_second** (horas, minutos y segundos), **day_minute** (días, horas y minutos), **day_second** (días a segundos).

- **hour(hora):** retorna la hora para el dato dado, en el rango de 0 a 23.Ejemplo:
  - **hour**('18:25:09') retorna "18";
- **minute(hora):** retorna los minutos de la hora dada, en el rango de 0 a 59.
- **monthname(fecha):** retorna el nombre del mes de la fecha dada. Ejemplo:
  - **monthname**('2006-08-10') retorna "August".
- **month(fecha):** retorna el mes de la fecha dada, en el rango de 1 a 12.
- **now() y sysdate():** retornan la fecha y hora actuales.
- **period_add(p,n):** agrega "n" meses al periodo "p", en el formato "YYMM" o "YYYYMM"; retorna un valor en el formato "YYYYMM". El argumento "p" no es una fecha, sino un año y un mes. Ejemplo:
  - **period_add**('200608',2) retorna "200610".

- **period_diff(p1,p2):** retorna el número de meses entre los períodos "p1" y "p2", en el formato "YYMM" o "YYYYMM". Los argumentos de período no son fechas sino un año y un mes. Ejemplo:
    - **period_diff**('200608','200602') retorna 6.
- **second(hora):** retorna los segundos para la hora dada, en el rango de 0 a 59.
- **sec_to_time(segundos):** retorna el argumento "segundos" convertido a horas, minutos y segundos. Ejemplo:
    - **sec_to_time**(90) retorna "1:30".
- **timediff(hora1,hora2):** retorna la cantidad de horas, minutos y segundos entre hora1 y hora2.
- **time_to_sec(hora):** retorna el argumento "hora" convertido en segundos.
- **to_days(fecha):** retorna el número de día (el número de día desde el año 0).
- **weekday(fecha):** retorna el índice del día de la semana para la fecha pasada como argumento. Los índices son: 0=lunes, 1=martes,... 6=domingo). Ejemplo:
    - **weekday**('2006-08-10') retorna 3, o sea jueves.
- **year(fecha):** retorna el año de la fecha dada, en el rango de 1000 a 9999. Ejemplo:
    - **year**('06-08-10') retorna "2006".

Notaciones para formatear las fechas en MySQL

Especificador	Descripción
%a	Día de la semana abreviado (Sun..Sat)
%b	Más abreviado (Jan..Dec)
%c	Mes, numérico (0..12)
%D	Día del mes con sufijo inglés (0th, 1st, 2nd, 3rd…)
%d	Día del mes numérico (00..31)
%e	Día del mes numérico (0..31)
%f	Microsegundos (000000..999999)
%H	Hora (00..23)
%h	Hora (01..12)
%I	Hora (01..12)
%i	Minutos, numérico (00..59)
%j	Día del año (001..366)

%k	Hora (0..23)
%l	Hora (1..12)
%M	Nombre 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	Segundos (00..59)
%s	Segundos (00..59)
%T	Hora, 24 horas (hh:mm:ss)
%U	Semana (00..53), en la que el domingo es el primer día de la semana
%u	Semana (00..53), en la que el lunes es el primer día de la semana
%V	Semana (01..53), en la que el domingo es el primer día de la semana; utilizado con %X
%v	Semana (01..53), en la 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 la que el domingo es el primer día de la semana, numérico, cuatro dígitos; usado con %V
%x	Año para la semana, en el 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 hora utilizando expresiones como las siguientes, obteniendo los resultados indicados:

```
SELECT DATE_FORMAT ( '1997-10-04 22:23:00',  '%W %M %Y') ;
Retorna: 'Saturday October 1997'
 SELECT DATE_FORMAT ( '1997-10-04 22:23:00',  '%H:%i:%s') ;
Retorna: '22:23:00'
SELECT DATE_FORMAT ( '1997-10-04 22:23:00',  '%D %y %a %d %m %b %j') ;
```

```
Retorna: '4th 97 Sat 04 10 Oct 277'
SELECT DATE_FORMAT ( '1997-10-04 22:23:00',  '%H %k %I %r %T %S %w') ;
Retorna:   '22 22 10 10:23:00 PM 22:23:00 00 6'
SELECT DATE_FORMAT ( '1999-01-01',  '%X %V') ;
Retorna: '1998 52'
```

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 **CASE** permite obtener varios valores en función de unas condiciones o comparaciones previas. Hay dos modos de utilizar el operador `CASE`:

```
CASE valor WHEN [valor_comparación] THEN resultado [WHEN [valor_comparación] THEN resultado ...] [ ELSE resultado] END
```

```
CASE WHEN [condición] THEN resultado [ WHEN [condición] THEN resultado ...] [ ELSE resultado] END
```

Estas dos formas pueden utilizarse como en los ejemplos:

```
SELECT CASE 1 WHEN 1 THEN 'one' WHEN 2 THEN 'two' ELSE 'more' END ;
Retorna: 'one'
SELECT CASE WHEN 1 > 0 THEN 'true' ELSE 'false' END ;
Retorna: 'true'
```

Operador IF

La operador **IF** permite, de forma similar a **CASE**, obtener valores diferenciados en función de una condición.

La sintaxis de la construcción **IF** es la siguiente:

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

Y funciona de la siguiente manera: si la condición es cierta, se devuelve la expresión1, de lo contrario, la expresión2. Veamos el ejemplo siguiente:

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

Función IFNULL

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

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

En el esquema *empresa*, se quieren mostrar a 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 siguiente sentencia:

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

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     162500  82000   344500

7698 NEGRO      370500

7782 CEREZO     318500

7788 GIL        390000

7839 REY        650000

7844 ZOBAR      195000  0       195000

7876 ALONSO     143000

7900 JIMENO     123500

7902 FERNÁNDEZ 390000

7934 MUÑOZ      169000

14 rows selected
```

El resultado no es el querido, ya que todos los empleados tienen salario pero no todos tienen comisión asignada (para quienes no tienen comisión hay un valor `NULL`en la columna correspondiente), lo que motiva que la suma sólo se calcule para quienes tienen comisión. Posiblemente esperaríamos que el SGBD considerase que uno `NULL`en la comisión de los empleados es innervalente a cero. Debemos especificar, tal y como se ve en la siguiente sentencia:

```
SELECT emp_no  AS  "Código", apellido AS  "Empleado", salario AS  "Salario",
     comision AS  "Comisión", salario + IFNULL ( comision, 0)  AS  "Suma"
```

```
FROM empleado;
```

Ahora sí que el resultado es el querido:

```
Código Empleado Salario Comisión Suma

------ ---------- --------- ------------ ------------- ---------

7369 SÁNCHEZ    104000  04000

7499 ARROYO     208000  39000   247000

7521 SALA       162500  65000   227500

7566 JIMÉNEZ    386750  386750

7654 MARTÍN     162500  182000  344500

7698 NEGRO      370500  370500

7782 CEREZO     318500  318500

7788 GIL        390000  390000

7839 REY        650000  650000

7844 ZOBAR      195000   195000

7876 ALONSO     143000  143000

7900 JIMENO     123500  123500

7902 FERNÁNDEZ  390000  390000

7934 MUÑOZ      169000  169000

14 rows selected
```

Función NULLIF

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

```
NULLIF ( expresión 1, expresión 2)
```

Si la expresión1 y la expresión2 se evalúan como iguales la función devuelve NULL; de lo contrario, devuelve la expresión1.

Veamos los siguientes ejemplos:

```
SELECT  NULLIF ( 1, 1) ;
```

Retorna: **NULL**

```
 SELECT  NULLIF ( 1, 2) ;
```

Retorna: **1**