LENGUAJE DE MANIPULACION DE DATOS

CONTENIDO

1. Ficheros 3
2. NOCIONES PREVIAS 3

Grafo relacional 3

Notación 5

3. MYSQL 5

HERRAMIENTAS GRÁFICAS 6

INTERPRETES 7

Ejecución de consultas en MySQL 8

4. El lenguaje DML 9
5. Sentencia SELECT 10
6. Filtros 12

Expresiones para filtros 13

Creación de filtros 15

Pertenencia a conjuntos 16

Filtros con operador de rango 17

Filtros con test de valor nulo 18

Filtros con test de patrón 18

Filtros con límite de registros 19

7. Ordenación 20
8. Consultas de resumen 22

Filtros por grupos 24

9. Subconsultas 25

Test de comparación 26

Test de pertenencia a conjunto 26

Test de existencia 27

Test cuantificados ALL y ANY 28

Subconsultas anidadas 29

10. Consultas multitabla 29
11. Consultas reflexivas 36
12. Consultas con tablas derivadas 36
13. Las vistas 37
14. Sentencia INSERT 39
15. Sentencia INSERT extendida 40
16. INSERT y SELECT 40
17. Sentencia UPDATE 41
18. Sentencia DELETE 42
19. Borrado y modificación con relaciones 42
20. UPDATE y DELETE con subconsultas 44
21. Script en MySQL 45

FICHEROS

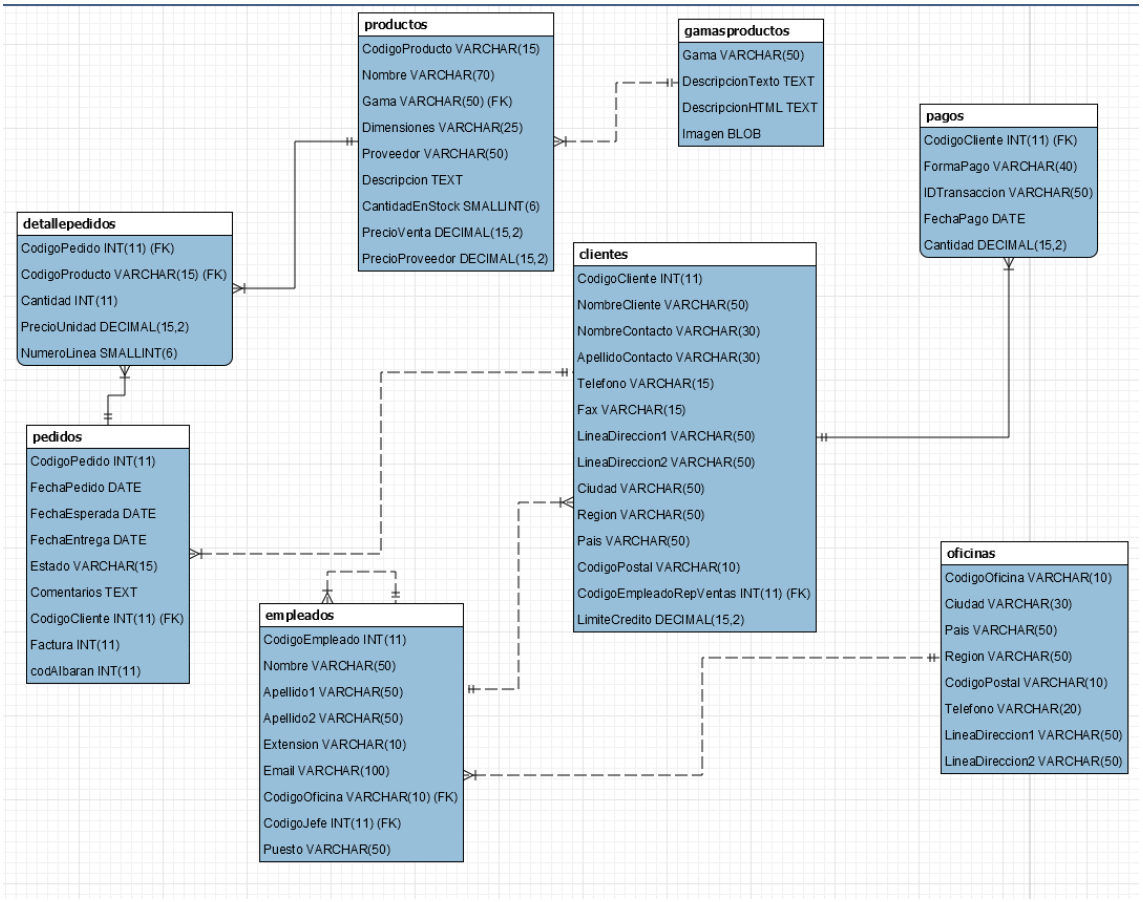
NOCIONES PREVIAS

Antes de empezar con las sentencias propias de SQL vamos a ver algunos conceptos que nos ayudarán a entender mejor el Modelo Relacional.

GRAFO RELACIONAL

Diagrama de la base de datos que muestra las tablas, los campos que las forman, las claves primarias y ajenas y las relaciones existentes entre las tablas.

Es importante, antes de empezar a manipular una base de datos mediante sentencias SQL, entender cómo está estructurada. Una forma sencilla es revisar el grafo relacional.



Entre las bondades del modelo relacional está evitar la redundancia y las inconsistencias.

Por ejemplo, en el grafo anterior vemos como la tabla **Pedidos** está relacionada con la tabla **Clientes**. Los datos de cada cliente están en la tabla clientes y solo se han metido una vez, sin embargo, un cliente puede hacer muchos pedidos, sino existiera esa relación entre la tabla clientes y la tabla pedidos, cada vez que el cliente hiciera un pedido tendríamos que volver a meter todos sus datos (nombre, dirección, nif, teléfono, etc), como se hace manualmente (a bolígrafo). Sin embargo, la relación nos evita tener que hacer esto, y ¿cómo se obtienen los datos del cliente, que son necesarios, en cada pedido? Guardando en la tabla pedidos un dato del cliente que lo identifique de forma única (que no lo pueda estar repetido en otro cliente) Ese dato corresponde a la clave principal de la tabla clientes y se guarda en un campo/s de la tabla pedidos que recibe el nombre de clave ajena. En este caso es el **CodigoCliente.** Que se llama igual en las dos tablas pero no es obligatorio.

Por otro lado, la relación garantiza que cada pedido de la tabla pedidos tenga asociado un cliente de la tabla clientes, evitando la inconsistencia que se produciría si el cliente de un pedido no existiera en la tabla clientes. No sabríamos a quién pertenece el pedido.

NOTACIÓN

La notación que seguiremos:

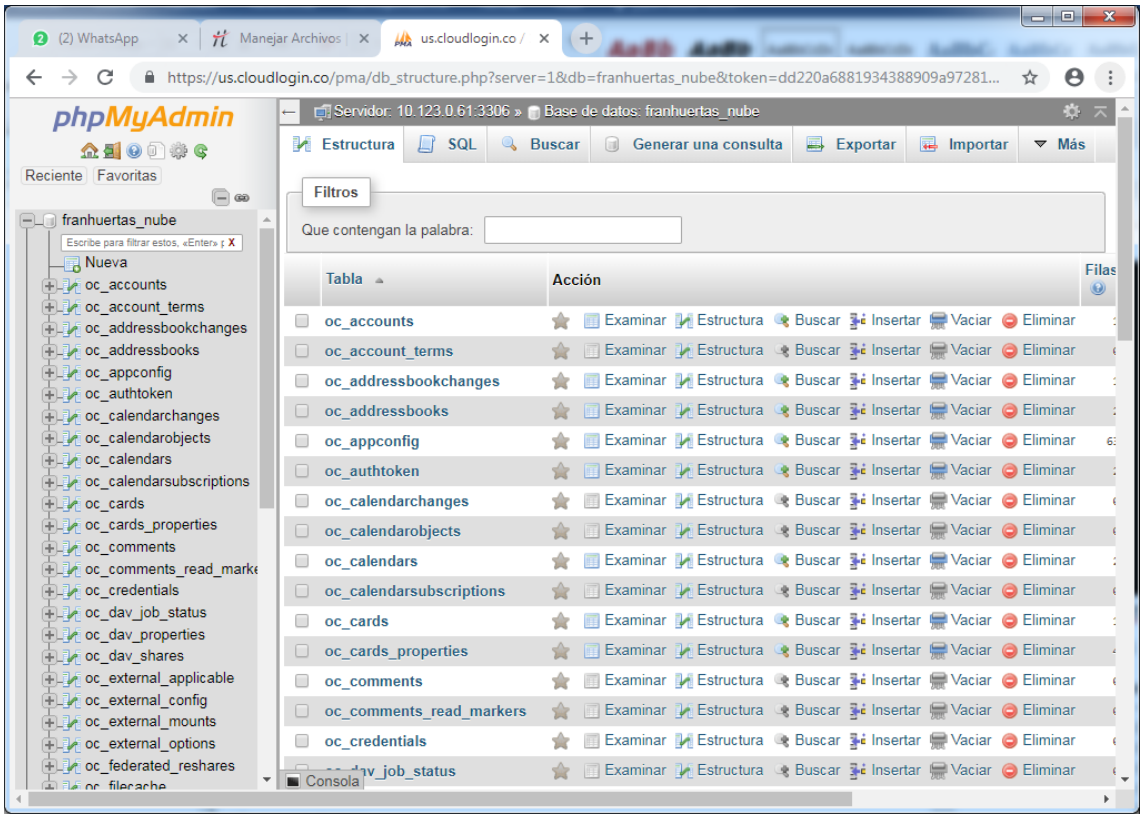
- **Palabras en mayúsculas**. Estas son las palabras reservadas del lenguaje. Por ejemplo, `SELECT, DROP, CREATE` son palabras reservadas, esto quiere decir que no puede utilizarse para nombrar objetos
- **Palabras en minúscula.** Se utiliza para realizar descripciones de sintaxis más en detalle. Por ejemplo, el token `especificación-de-filtro` se puede desplegaren más definición es para realizar filtros en las consultas.
- **Corchetes**. Un elemento sintáctico entre corchetes indica opcionalidad. Es decir, lo que está encerrado entre corchetes se puede incorporar a la sentencia o, dependiendo de lo que el programador quiera expresar.
- **Llaves**. Indica alternativa obligatoria. Se debe elegir entre los elementos separados mediante el token pipe "|". Por ejemplo, en la definición de sintaxis para crear una base de datos, `CREATE {DATABASE| SCHEMA} nombre-bd`, hay que escribir uno de los dos tokens entre llaves. Se puede optar bien por `CREATE DATABASE nombre_bd` o por `CREATESCHEMA nombre.bd.`
- **Puntos suspensivos**. Significa repetición, es decir, el último elemento sintáctico puede repetirse varias veces. Por ejemplo, para codificar una consulta se usa la definición `SELECT columna [,columna]...FROM tabla`. Los puntos suspensivos significan que se puede repetir el token `[,columna]` tantas veces como quiera.

MYSQL

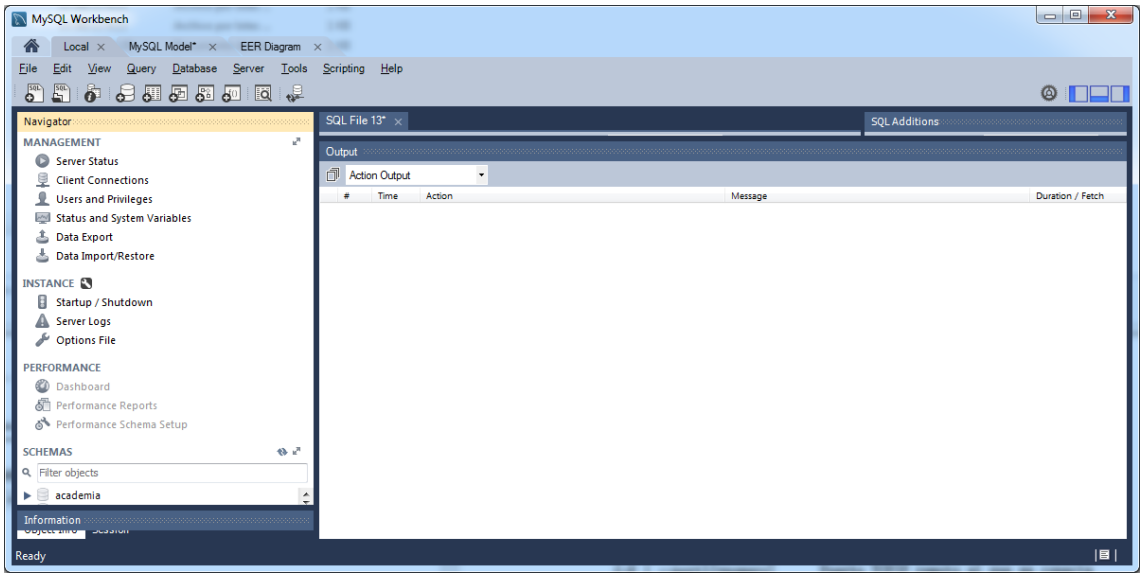
HERRAMIENTAS GRÁFICAS

Existen muchas herramientas que permiten la creación y administración de BBDD MySQL.

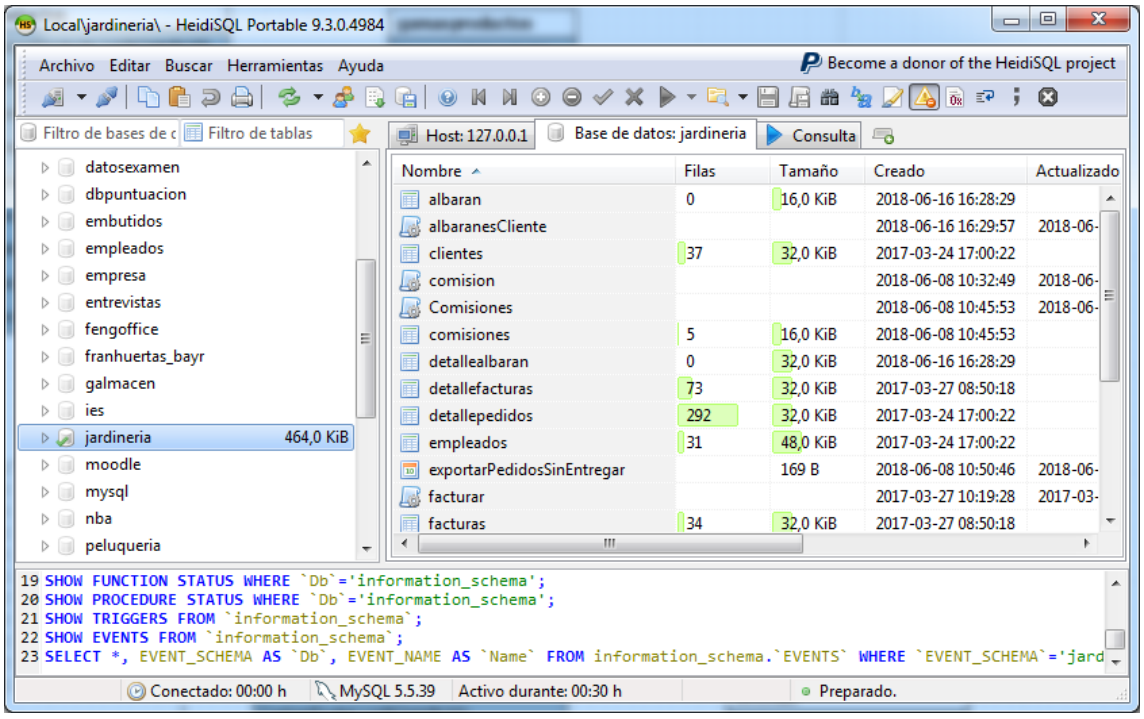
PhpMyAdmin. Se trata de una herramienta basada Web y desarrollada en PHP que permite la administración de bases de datos MySQL. Está presente en casi todos los servidores de hosting.



MySQL Workbench. Es una aplicación de escritorio bastante completa que permite la creación de bases de datos a partir del diagrama Entidad Relación. También permite ingeniería inversa, obtener el diagrama a partir de la base de datos.



HeidiSQL. Herramienta bastante ligera e intuitiva que permite administrar bases de datos MySQL sin necesidad instalación.



INTERPRETES

La utilidad principal de un SGBD es su intérprete de comandos. Es una aplicación cliente cuya única misión es enviar comandos al SGBD y mostrar los resultados devueltos por el SGBD en pantalla. El cliente del servidor MySQL (mysql-server) se llama *mysql*.

MySQL: El cliente de MySQL-Server

```
mysql [options] [database]

options:
     --help                        Visualiza la ayuda
     {-p | --password}[=frase]     Password con la que se conecta
     {-P | --port}[=numero]        Puerto TCPIP remoto al que se conecta
     {-h | --host}[=numero]        Nombre Host o IP al que se conecta
     {-u | --user}[=usuario]       Usuario con el que se conecta
     {-s | --socket}[=nombre_fich] Fichero socket con el que se conecta

#ejemplo típico:
mysql -u root -p
Enter password: *********

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 37
Server version: 5.0.75-0ubuntu10.2 (Ubuntu)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.


#conexión sin usuario y password (se conecta como anónimo y sin password)
mysql

#conexión con usuario y password (se conecta como root y su password )
mysql -u root -p
Enter password: *********

#conexión con usuario y password en claro a la base de datos jardineria
mysql -u root -pPasswordDelUsuario jardineria

#conexión con usuario y password en claro a la base de datos jardineria
# del host 192.168.3.100
mysql -u root -pPasswordDelUsuario -h 192.168.3.100 jardineria

#conexión con usuario y password en claro a la base de datos jardineria
# del host 192.168.3.100 con puerto 15300
mysql -u root -pPasswordDelUsuario -h 192.168.3.100 jardineria -P 15300
```

EJECUCIÓN DE CONSULTAS EN MYSQL

Las palabras clave no son sensibles a mayúsculas, los campos y tablas, lo son en Linux, no en Windows.

```
mysql> SELECT VERSION(),CURRENT_DATE();
mysql> SELect Version(),current_Date();
mysql> select version(),current_date();
```

```
#selecciona la version del gestor y la fecha actual
mysql> select version(),current_date();
+-------------------+----------------+
| version()         | current_date() |
+-------------------+----------------+
| 5.0.75-0ubuntu10.2 | 2009-8-20      |
+-------------------+----------------+
1 row in set (0.00 sec)
```

Siempre hay que terminar la línea de consulta con ; o *ctrl+c*

Una forma de ejecutar comandos SQL es desde un fichero de texto mediante la orden *source*.

```
#ejecución del script de creación crear_bbdd_startrek.sql
mysql> source /home/ivan/crear_bbdd_startrek.sql
```

También en segundo plano.

```
#ejecucion en modo batch
~$ mysql -u root -pPassWdUsuario <crear_bbdd_startrek.sql

#ejecucion en modo batch almacenando resultados
~$ mysql -u root -pPassWdUsuario <crear_bbdd_startrek.sql >resultado
```

EL LENGUAJE DML

Las sentencias DML (Lenguaje de Manipulación de Datos) del lenguaje SQL son las siguientes:

- La sentencia INSERT, cuyo cometido es insertar uno o varios registros en alguna tabla.
- La sentencia DELETE, que borra registros de una tabla.
- La sentencia UPDATE, que modifica registros de una tabla.
- La sentencia SELECT, que se utiliza para extraer información de la base de datos, ya sea de una tabla o de varias.

Grafo relacional de la base de datos JARDINERIA. Generado con Workbench.

SENTENCIA SELECT

Es la sentencia más versátil de todo SQL, y por tanto la más compleja de todas.

Ejemplos:

Consulta sencilla.

```
#esta consulta selecciona todos los campos y muestra todos los
#registros de la tabla empleados
SELECT * FROM empleados;
```

Consulta compleja

```
#esta consulta obtiene el total de los pedidos
#de los clientes de una tienda
      SELECT NombreCliente,tot_Cantidad
      FROM Clientes,Pedidos,
      (SELECT sum(Cantidad*PrecioUnidad) as Cantidad,NumeroPedido
      FROM DetallePedidos GROUP BY NumeroPedido) tot
      WHERE Clientes.NumeroCliente=Pedidos.NumeroCliente
      AND Pedidos.numeroPedido=tot.NumeroPedido ORDER BY Cantidad;
```

Formato básico.

```
SELECT [DISTINCT] select_expr [,select_expr]
... [FROM tabla]

select_expr:

nombre_columna [AS alias]
    | *
    | expresión
```

nombre-columna indica un nombre de columna, es decir, se puede seleccionar de una tabla una serie de columnas, o todas si se usa *, o una expresión algebraica compuesta por operadores, operandos y funciones.

El parámetro opcional DISTINCT fuerza que solo se muestren los registros con valores distintos, o, dicho de otro modo, que suprima las repeticiones.

Ejemplos

```
#consulta 1
SELECT * FROM vehiculos;
+-----------+----------------+---------+
| matricula | modelo         | marca   |
+-----------+----------------+---------+
| 1129FGT   | ibiza gt       | seat    |
| 1132GHT   | leon tdi 105cv | seat    |
| M6836YX   | corolla g6     | toyota  |
| 7423FZY   | coupe          | hyundai |
| 3447BYD   | a3 tdi 130cv   | audi    |
+-----------+----------------+---------+
```

```
#consulta 2
SELECT matricula, modelo
FROM vehiculos;
+-----------+----------------+
| matricula | modelo         |
+-----------+----------------+
| 1129FGT   | ibiza gt       |
| 1132GHT   | leon tdi 105cv |
| M6836YX   | corolla g6     |
| 7423FZY   | coupe          |
| 3447BYD   | a3 tdi 130cv   |
+-----------+----------------+
```

```
#consulta 3
SELECT matricula,
concat(marca,modelo) as coche
FROM vehiculos;
+-----------+---------------------+
| matricula | coche               |
+-----------+---------------------+
| 1129FGT   | seatibiza gt        |
| 1132GHT   | seatleon tdi 105cv  |
| M6836YX   | toyotacorolla g6    |
| 7423FZY   | hyundaicoupe        |
| 3447BYD   | audia3 tdi 130cv    |
+-----------+---------------------+
```

```
#consulta 4
SELECT matricula, modelo,1+5
FROM vehiculos;
+-----------+----------------+-----+
| matricula | modelo         | 1+5 |
+-----------+----------------+-----+
| 1129FGT   | ibiza gt       |   6 |
| 1132GHT   | leon tdi 105cv |   6 |
| M6836YX   | corolla g6     |   6 |
| 7423FZY   | coupe          |   6 |
| 3447BYD   | a3 tdi 130cv   |   6 |
+-----------+----------------+-----+
```

```
#consulta 5
SELECT 1+6;
+-----+
| 1+6 |
+-----+
|   7 |
+-----+
```

```
#consulta 6
SELECT marca FROM vehiculos;
+---------+
| marca   |
+---------+
| seat    |
| seat    |
| toyota  |
| hyundai |
| audi    |
+---------+
```

```
#consulta 7
SELECT DISTINCT marca
FROM vehiculos;
+---------+
| marca   |
+---------+
| seat    |
| toyota  |
| hyundai |
| audi    |
+---------+
```

FILTROS

Los filtros son condiciones que cualquier gestor de base de datos interpreta para seleccionar registros y mostrarlos como resultado de la consulta. En SQL la palabra clave para realizar filtros es la cláusula **WHERE**.

A continuación, se añade a la sintaxis de la cláusula SELECT la sintaxis de los filtros:

```
SELECT [DISTINCT] select_expr [,select_expr]
...
[FROM tabla] [WHERE filtro]
```

filtro es una expresión que indica la condición o condiciones que deben satisfacer los registros para ser seleccionados.

```
#selecciona los vehículos de la marca seat
SELECT * FROM vehiculos
WHERE marca='seat';
+-----------+---------------+-------+
| matricula | modelo        | marca |
+-----------+---------------+-------+
| 1129FGT   | ibiza gt      | seat  |
| 1132GHT   | leon tdi 105cv | seat  |
+-----------+---------------+-------+
```

EXPRESIONES PARA FILTROS

Una expresión, es una combinación de operadores, operandos y funciones que producen un resultado.

```
#expresión 1 (oracle): (2+3)*7
SELECT (2+3)*7 from dual;

   (2+3)*7
----------
        35

#expresión 2 (mysql): (2+3)>(6*2)
SELECT (2+3)>(6*2);
+-------------+
| (2+3)>(6*2) |
+-------------+
|           0 |  #0 = falso, es falso que 5>12
+-------------+

#expresión 3 (mysql): la fecha de hoy -31 años;
SELECT date_sub(now(), interval 31 year);
+-----------------------------------+
| date_sub(now(), interval 31 year) |
+-----------------------------------+
| 1977-10-30 13:41:40               |
+-----------------------------------+
```

Partes de la expresión:

Operandos. Los operandos pueden ser constantes, por ejemplo el número entero 3, el número real 2.3, la cadena de caracteres 'España' o la fecha '2010-01-02'; también pueden ser variables, por ejemplo el campo edad o el campo NombreMascota, y pueden ser también otras expresiones.

Operadores aritméticos: +, -, *, /, %. El operador + y el operador - se utilizan para sumar o restar dos operandos (binario) o para poner el signo positivo o negativo a un operando (unario). El operador * es la multiplicación de dos operandos y el operador / es para dividir. El operador % o resto de la división entera a%b devuelve el resto de dividir a entre b.

Operadores relaciónales: >, <, <>, >=, <=, =. Los operadores relaciónales sirven para comparar dos operandos. Así, es posible

preguntar si un campo es mayor que un valor, o si un valor es distinto de otro. Estos operadores devuelven un número entero, de tal manera que si el resultado de la expresión es cierto el resultado será 1, y si el resultado es falso el resultado será 0. Por ejemplo, la expresión **a >b** devuelve 1 si a es estrictamente mayor que b y 0 en caso contrario. La expresión **d<>e** devuelve 1 si d y e son valores distintos.

Operadores lógicos: AND, OR, NOT. Los operadores lógicos toman como operandos valores lógicos, esto es, cierto o falso, en caso de SQL, 1 o 0.

Paréntesis: (). Los operadores tienen una prioridad, para alterar esa prioridad se usan los paréntesis, que tienen la máxima prioridad.

Funciones: *date_add, concat, left, right*,... Cada SGBD incorpora su propio repertorio de funciones que en pocas ocasiones coincide con el de otros SGBD. Para ver todas las funciones que incorpora MySQL consulta el manual. También puedes consultar la web: http://mysql.conclase.net/curso/?cap=011

Ejemplo de operaciones y sus resultados.

Operación	Resultado
7+2*3	13
(7-2)*3	15
7>2	1
9<2	0
7>2 AND 4<3	0
7>2 OR 4<3	1
(10>=10 AND 0<=1)+2	3

CREACIÓN DE FILTROS

Vamos a ver como se crean los filtros para obtener los datos deseados a través de ejemplos.

```
#la tabla jugadores contiene todos los jugadores de la nba
describe jugadores;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| codigo        | int(11)     | NO   | PRI | NULL    |       |
| Nombre        | varchar(30) | YES  |     | NULL    |       |
| Procedencia   | varchar(20) | YES  |     | NULL    |       |
| Altura        | varchar(4)  | YES  |     | NULL    |       |
| Peso          | int(11)     | YES  |     | NULL    |       |
| Posicion      | varchar(5)  | YES  |     | NULL    |       |
| Nombre_equipo | varchar(20) | YES  | MUL | NULL    |       |
+---------------+-------------+------+-----+---------+-------+

#Consulta que selecciona los jugadores españoles de los Lakers
SELECT codigo,Nombre,Altura
FROM jugadores WHERE Nombre_equipo='Lakers'
   and Procedencia='Spain';
+--------+----------+--------+
| codigo | Nombre   | Altura |
+--------+----------+--------+
|     66 | Pau Gasol | 7-0    |
+--------+----------+--------+
#Consulta que selecciona los jugadores españoles y eslovenos de los lakers
SELECT Nombre, Altura,Procedencia FROM jugadores
WHERE Nombre_equipo='Lakers'
        AND (Procedencia='Spain' OR Procedencia='Slovenia');
+--------------+--------+-------------+
| Nombre       | Altura | Procedencia |
+--------------+--------+-------------+
| Pau Gasol    | 7-0    | Spain       |
| Sasha Vujacic | 6-7    | Slovenia    |
+--------------+--------+-------------+
```

PERTENENCIA A CONJUNTOS

Además de los operadores vistos, se puede usar el operador de pertenencia a conjuntos IN.

```
nombre_columna IN (Valuel, Value2, ...)
```

Este operador permite comprobar si una columna tiene un valor igual que cualquier de los que están incluidos dentro del paréntesis.

```
# versión larga
SELECT Nombre, Altura,Procedencia
FROM jugadores WHERE Nombre_equipo='Lakers' AND
(Procedencia='Spain'
OR Procedencia='Slovenia'
OR Procedencia='Serbia & Montenegro');

#versión corta (con el operador IN)
SELECT Nombre, Altura,Procedencia FROM jugadores
WHERE Nombre_equipo='Lakers' AND
Procedencia IN ('Spain','Slovenia','Serbia &
Montenegro');
```

```
+--------------------+--------+--------------------+
| Nombre             | Altura | Procedencia        |
+--------------------+--------+--------------------+
| Pau Gasol          | 7-0    | Spain              |
| Vladimir Radmanovic | 6-10   | Serbia & Montenegro |
| Sasha Vujacic      | 6-7    | Slovenia           |
+--------------------+--------+--------------------+
```

FILTROS CON OPERADOR DE RANGO

El operador de rango BETWEEN permite seleccionar los registros que estén incluidos en un rango.

```
nombre.columna BETWEEN Valuel AND Value2
```

```
SELECT Nombre,Nombre_equipo,Peso FROM jugadores
WHERE Peso BETWEEN 270 AND 300;
+---------------+--------------+------+
| Nombre        | Nombre_equipo | Peso |
+---------------+--------------+------+
| Chris Richard | Timberwolves |  270 |
| Paul Davis    | Clippers     |  275 |
| ....          | ....         |  ... |
| David Harrison | Pacers       |  280 |
+---------------+--------------+------+
```

Ejercicio 4.2: Saca el peso en kilogramos redondeado a un decimal de los jugadores de la NBA que pesen entre 120 y 150 kilos. Una libra equivale a 0.4535 kilos.

FILTROS CON TEST DE VALOR NULO

Los operadores IS e IS NOT permiten verificar si un campo es o no es nulo respectivamente.

```
SELECT nombre,Nombre_equipo
FROM jugadores WHERE Procedencia IS null;
+----------------+---------------+
| nombre         | Nombre_equipo |
+----------------+---------------+
| Anthony Carter | Nuggets       |
+----------------+---------------+

#la query contraria saca el resto de jugadores
SELECT nombre,Nombre_equipo
FROM jugadores WHERE Procedencia IS NOT null;
+---------------------+---------------+
| nombre              | Nombre_equipo |
+---------------------+---------------+
| Corey Brever        | Timberwolves  |
| Greg Buckner        | Timberwolves  |
| Michael Doleac      | Timberwolves  |
......
| C.J. Watson         | Warriors      |
| Brandan Wright      | Warriors      |
+---------------------+---------------+
```

FILTROS CON TEST DE PATRÓN

Los filtros con test patrón seleccionan los registros que cumplan una serie de características. Se pueden usar los caracteres comodines % y _ para buscar una cadena de caracteres.

```
SELECT * FROM vehículos where modelo like '%tdi%';

| matricula | modelo         | marca |
+-----------+----------------+-------+
| 1132GHT   | leon tdi 105cv | seat  |
| 3447BYD   | a3 tdi 130cv   | audi  |
+-----------+----------------+-------+
```

```
SELECT Nombre,Conferencia
FROM equipos WHERE Nombre like 'R_____s';
+---------+-------------+
| Nombre  | Conferencia |
+---------+-------------+
| Raptors | East        |
| Rockets | West        |
+---------+-------------+
SELECT Nombre,Conferencia
FROM equipos WHERE Nombre like '_o%';
+---------+-------------+
| Nombre  | Conferencia |
+---------+-------------+
| Bobcats | East        |
| Hornets | West        |
| Rockets | West        |
+---------+-------------+
```

FILTROS CON LÍMITE DE REGISTROS

Este tipo de filtros no es estándar y su funcionamiento varía con el SGBD. Consiste en limitar el número de registros devuelto por una consulta.

```
[LIMIT [desplazamiento,] nfilas]
```

nfilas especifica el número de filas a devolver y desplazamiento especifica a partir de qué fila se empieza a contar.

```
#devuelve las 4 primeras filas
SELECT nombre,Nombre_equipo
FROM jugadores limit 4;
+---------------+--------------+
| nombre        | Nombre_equipo |
+---------------+--------------+
| Corey Brever  | Timberwolves  |
| Greg Buckner  | Timberwolves  |
| Michael Doleac | Timberwolves  |
| Randy Foye    | Timberwolves  |
+---------------+--------------+

#devuelve 3 filas a partir de la sexta
SELECT nombre,Nombre_equipo
FROM jugadores LIMIT 5,3;
+------------------+--------------+
| nombre           | Nombre_equipo |
+------------------+--------------+
| Marko Jaric      | Timberwolves  |
| Al Jefferson     | Timberwolves  |
| Mark Madsen      | Timberwolves  |
+------------------+--------------+
```

En Oracle.

```
--Saca los 25 primeros jugadores
SELECT *
FROM jugadores
WHERE  rownum <= 25;
```

ORDENACIÓN

Para mostrar ordenados un conjunto de registros se utiliza la cláusula **ORDER BY** de la sentencia **SELECT**.

```
SELECT [DISTINCT] select_expr [,select_expr] ...

[FROM tabla]

[WHERE filtro]

[ORDER BY {nombre.columna | expr | posición} [ASC | DESC],
...]
```

```
#estructura de la tabla equipos
DESCRIBE equipos;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| Nombre      | varchar(20) | NO   | PRI | NULL    |       |
| Ciudad      | varchar(20) | YES  |     | NULL    |       |
| Conferencia | varchar(4)  | YES  |     | NULL    |       |
| Division    | varchar(9)  | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+

#obtener los equipos de la conferencia oeste de la nba ordenados por división
SELECT Nombre,Division
FROM equipos WHERE Conferencia='West'
ORDER BY Division ASC;
+---------------+-----------+
| Nombre        | Division  |
+---------------+-----------+
| Jazz          | NorthWest |
| Nuggets       | NorthWest |
| Trail Blazers | NorthWest |
| Timberwolves  | NorthWest |
| Supersonics   | NorthWest |
| Clippers      | Pacific   |
| Kings         | Pacific   |
| Warriors      | Pacific   |
| Suns          | Pacific   |
| Lakers        | Pacific   |
| Hornets       | SouthWest |
| Spurs         | SouthWest |
| Rockets       | SouthWest |
| Mavericks     | SouthWest |
| Grizzlies     | SouthWest |
+---------------+-----------+
```

```
#se puede ordenar por varios campos, p.ej: además de que cada
#división esté ordenada ascendentemente se ordene por nombre
#de equipo
SELECT Division,Nombre FROM equipos
WHERE Conferencia='West'
ORDER BY Division ASC,Nombre DESC;
+-----------+---------------+
| Division  | Nombre        |
+-----------+---------------+
| NorthWest | Trail Blazers |
| NorthWest | Timberwolves  |
| NorthWest | Supersonics   |
| NorthWest | Nuggets       |
| NorthWest | Jazz          |
| Pacific   | Warriors      |
| Pacific   | Suns          |
| Pacific   | Lakers        |
| Pacific   | Kings         |
| Pacific   | Clippers      |
| SouthWest | Spurs         |
| SouthWest | Rockets       |
| SouthWest | Mavericks     |
| SouthWest | Hornets       |
| SouthWest | Grizzlies     |
+-----------+---------------+
```

CONSULTAS DE RESUMEN

En SQL se pueden generar consultas con información calculada sobre varios registros.

```
SELECT count(*) FROM vehiculos;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
```

Las funciones disponibles para realizar estos cálculos son:

```
SUM (Expresión) #Suma los valores indicados en el argumento

AVG (Expresión) #Calcula la media de los valores

MIN (Expresión) #Calcula el mínimo

MAX (Expresión) #Calcula el máximo

COUNT (nbColumna) #Cuenta el número de valores de una
columna    #(excepto los nulos)

COUNT (*) #Cuenta el número de valores de una fila
```

#Incluyendo los nulos.

```
#consulta 1
#¿Cuánto pesa el jugador más pesado de la nba?
SELECT max(peso) FROM jugadores;

#consulta 2
#¿Cuánto mide el jugador más bajito de la nba?
SELECT min(altura) FROM jugadores;

#consulta 3
#¿Cuántos jugadores tienen los Lakers?
SELECT count(*) FROM jugadores WHERE Nombre_equipo='Lakers';

#consulta 4
#¿Cuánto pesan de media los jugadores de los Blazers?
SELECT avg(peso) FROM jugadores WHERE Nombre_equipo='Blazers';
```

Agrupando (GROUP BY)

```
SELECT [DISTINCT] select_expr [,select_expr] ...

[FROM tabla]

[WHERE filtro]

[GROUP BY expr [, expr].... ]

[ORDER BY {nombre.columna | expr | posición} [ASC | DESC] ,
...]
```

```
#consulta 1
#¿Cuánto pesa el jugador más pesado de cada equipo?
 SELECT Nombre_equipo, max(peso)
 FROM jugadores GROUP BY Nombre_equipo;
+---------------+-----------+
| Nombre_equipo | max(peso) |
+---------------+-----------+
| 76ers         |       250 |
| Bobcats       |       266 |
| Bucks         |       260 |
.....
| Trail Blazers |       255 |
| Warriors      |       250 |
| Wizards       |       263 |
+---------------+-----------+

#consulta 2
#¿Cuántos equipos tiene cada conferencia en la nba?
SELECT count(*),conferencia FROM equipos GROUP BY conferencia;
+----------+-------------+
| count(*) | conferencia |
+----------+-------------+
|       15 | East        |
|       15 | West        |
+----------+-------------+

#query 3
#¿Cuánto pesan de media los jugadores de españa, francia e italia?
SELECT avg(peso),procedencia FROM jugadores
WHERE procedencia IN ('Spain','Italy','France') GROUP BY procedencia;
+-----------+-------------+
| avg(peso) | procedencia |
+-----------+-------------+
|  218.4000 | France      |
|  221.0000 | Italy       |
|  208.6000 | Spain       |
+-----------+-------------+
```

IMPORTANTE: Se observa que para cada agrupación, se ha seleccionado también el nombre de la columna por la cual se agrupa. Esto no es posible si no se incluye el GROUP BY

```
mysql> SELECT count(*),conferencia FROM equipos;
ERROR 1140 (42000): Mixing of GROUP columns
 (MIN(),MAX(),COUNT(),...) with no GROUP columns is illegal
if there is no GROUP BY clause
```

FILTROS POR GRUPOS

Los filtros de grupos deben realizarse mediante el uso de la cláusula HAVING puesto que WHERE actúa antes de agrupar los registros.

```
SELECT [DISTINCT] select_expr [,select_expr] ...
[FROM tabla]
[WHERE filtro]
[GROUP BY expr [, expr].... ]
[HAVING filtro_grupos]
[ORDER BY {nombre.colunma I expr | posición} [ASC | DESC] , ...]
```

```
#query 1:
#Seleccionar los equipos de la nba cuyos jugadores
#pesen de media más de 228 libras
SELECT Nombre_equipo,avg(peso)
FROM jugadores
GROUP BY  Nombre_equipo
HAVING avg(peso)>228 ORDER BY avg(peso);
+---------------+-----------+
| Nombre_equipo | avg(peso) |
+---------------+-----------+
| Suns          |  228.8462 |
| Wizards       |  229.6923 |
| Lakers        |  230.0000 |
| Jazz          |  230.0714 |
| Knicks        |  235.4667 |
+---------------+-----------+
```

```
#query 2
#seleccionar qué equipos de la nba tienen más de 1 jugador español
 SELECT Nombre_equipo,count(*)
    FROM jugadores
    WHERE procedencia='Spain'
    GROUP BY Nombre_equipo
    HAVING count(*)>1;
+---------------+----------+
| Nombre_equipo | count(*) |
+---------------+----------+
| Raptors       |        2 |
+---------------+----------+
```

SUBCONSULTAS

Las subconsultas se utilizan para realizar filtrados con los datos de otra consulta.

Ej. Selecciona el nombre de todos los jugadores que juegan en la división SouthWest.

```
SELECT nombre FROM jugadores
WHERE Nombre_equipo IN
(SELECT Nombre FROM equipos WHERE division='SouthWest');
+---------------------+
| nombre              |
+---------------------+
| Andre Brown         |
| Kwame Brown         |
| Brian Cardinal      |
| Jason Collins       |
| ...                 |
+---------------------+
```

TEST DE COMPARACIÓN

Consiste en usar los operadores de comparación =, >=, <=, <>, > y < para comparar el valor producido con **un valor único** generado por una subconsulta.

```
SELECT nombre FROM jugadores
WHERE altura =
        (SELECT max(altura) FROM jugadores);
+----------+
| nombre   |
+----------+
| Yao Ming |
+----------+
```

Fallaría si hay más de un valor en la subconsulta.

TEST DE PERTENENCIA A CONJUNTO

```
SELECT division FROM equipos WHERE nombre in
(SELECT Nombre_equipo FROM jugadores WHERE procedencia='Spain');
+-----------+
| division  |
+-----------+
| Atlantic  |
| NorthWest |
| Pacific   |
| SouthWest |
+-----------+
```

TEST DE EXISTENCIA

Permite filtrar los resultados si existen filas en la subconsulta. Se usa el operador **EXISTS**.

```
SELECT columnas FROM tabla

WHERE [NOT ]EXISTS (subconsulta)
```

Ejemplo. Equipos que no tengan jugadores españoles.

```
SELECT Nombre FROM equipos WHERE NOT EXISTS
        (SELECT  Nombre FROM jugadores
        WHERE equipos.Nombre = jugadores.Nombre_Equipo
        AND procedencia='Spain');
+--------------+
| Nombre       |
+--------------+
| 76ers        |
| Bobcats      |
| Bucks        |
| ...          |
+--------------+
```

Para comprender la lógica de esta **query**, se puede asumir que cada registro devuelto por la consulta principal provoca la ejecución de la subconsulta, así, si la consulta principal (*SELECT Nombre FROM Equipos*) devuelve 30 registros, se entenderá que se ejecutan 30 subconsultas, una por cada nombre de equipo que retorne la consulta principal.

```
 SELECT Nombre from equipos;
+---------------+
| Nombre        |
+---------------+
| 76ers         | -> subconsulta ejecutada #1
| Bobcats       | -> subconsulta ejecutada #2
| ...           | ...
| Raptors       | -> subconsulta ejecutada #22
| ...           | -> ....
+---------------+


#subconsulta ejecutada #1
SELECT  Nombre FROM jugadores
        WHERE '76ers' = jugadores.Nombre_Equipo
        AND procedencia='Spain';
```

Esta subconsulta no retorna resultados, por tanto, el equipo `'76ers'` cumple el filtro. `(NOT EXISTS)`.

TEST CUANTIFICADOS ALL Y ANY

Los test cuantificados sirven para calcular la relación entre una expresión y todos los registros de la subconsulta (ALL) o algunos de los registros de la subconsulta

(ANY).

```
SELECT nombre,peso from jugadores
WHERE peso > ALL
(SELECT peso FROM jugadores WHERE procedencia='Spain');
+---------------------+------+
| nombre              | peso |
+---------------------+------+
| Michael Doleac      |  262 |
| Al Jefferson        |  265 |
| Chris Richard       |  270 |
| ...                 |  ... |
+---------------------+------+
SELECT nombre,peso from jugadores
WHERE posicion='G' AND
peso > ANY
(SELECT peso FROM jugadores where posicion='C');
+------------+------+
| nombre     | peso |
+------------+------+
| Joe Johnson |  235 |
+------------+------+
```

SUBCONSULTAS ANIDADAS

Se puede usar una subconsulta para filtrar los resultados de otra subconsulta. De esta manera se anidan subconsultas.

Obtener la ciudad donde juega el jugador más alto.

```
SELECT ciudad FROM equipos WHERE nombre =
  (SELECT Nombre_equipo FROM jugadores WHERE altura =
      (SELECT MAX(altura) FROM jugadores));
+----------+
| ciudad   |
+----------+
| Houston  |
+----------+
```

CONSULTAS MULTITABLA

Una consulta multitabla es aquella en la que se puede consultar información de más de una tabla. Se utilizan los campos relacionados para unirlas. **Es muy importante comprender cómo están relacionadas las tablas de la base de datos mediante el grafo relacional e identificar los campos que se usan para unirlas.**

```
SELECT [DISTINCT] select_expr [,select_expr] ...
[FROM referencias.tablas]
[WHERE filtro]
[GROUP BY expr [, expr].... ]
[HAVING filtro.grupos]
[ORDER BY {nombre.columnas I expr I posición} [ASC I DESC] , ...]
```

```
referencias_tablas;
referencia_tabla[, referencia_tabla] ...
| referencia_tabla [INNER | CROSS] JOIN referencia_tabla
[ON condición]
| referencia_tabla LEFT [OUTER] JOIN referencia_tabla ON condición
| referencia.tabla RIGHT [OUTER] JOIN referencia_tabla ON
condición
```

referencia_tabla:

```
nombre_tabla [[AS] alias]
```

SQL1 (86)

Realiza el producto cartesiano de dos tablas, que son todas las combinaciones de las filas de una tabla unidas a las filas de la otra tabla.

Ejemplo. Tenemos dos tablas: animales y propietarios

```
SELECT * FROM propietarios;
+-----------+--------------------+
| dni       | nombre             |
+-----------+--------------------+
| 51993482Y | José Pérez         |
| 2883477X  | Matías Fernández   |
| 37276317Z | Francisco Martínez |
+-----------+--------------------+

SELECT * FROM animales;
+--------+----------+-------+-------------+
| codigo | nombre   | tipo  | propietario |
+--------+----------+-------+-------------+
|      1 | Cloncho  | gato  | 51993482Y   |
|      2 | Yoda     | gato  | 51993482Y   |
|      3 | Sprocket | perro | 37276317Z   |
+--------+----------+-------+-------------+
```

El producto cartesiano resultante sería:

```
SELECT * FROM animales,propietarios;
+--------+----------+-------+-------------+-----------+--------------------+
| codigo | nombre   | tipo  | propietario | dni       | nombre             |
+--------+----------+-------+-------------+-----------+--------------------+
|      1 | Cloncho  | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      2 | Yoda     | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      3 | Sprocket | perro | 37276317Z   | 51993482Y | José Pérez         |
|      1 | Cloncho  | gato  | 51993482Y   | 2883477X  | Matías Fernández   |
|      2 | Yoda     | gato  | 51993482Y   | 2883477X  | Matías Fernández   |
|      3 | Sprocket | perro | 37276317Z   | 2883477X  | Matías Fernández   |
|      1 | Cloncho  | gato  | 51993482Y   | 37276317Z | Francisco Martínez |
|      2 | Yoda     | gato  | 51993482Y   | 37276317Z | Francisco Martínez |
|      3 | Sprocket | perro | 37276317Z   | 37276317Z | Francisco Martínez |
+--------+----------+-------+-------------+-----------+--------------------+
```

Si se aplica un filtro

```
SELECT * FROM animales,propietarios
    WHERE propietarios.dni=animales.propietario;
+--------+----------+-------+-------------+-----------+--------------------+
| codigo | nombre   | tipo  | propietario | dni       | nombre             |
+--------+----------+-------+-------------+-----------+--------------------+
|      1 | Cloncho  | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      2 | Yoda     | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      3 | Sprocket | perro | 37276317Z   | 37276317Z | Francisco Martínez |
+--------+----------+-------+-------------+-----------+--------------------+
```

SQL2 (92)

En SQL2 se introduce una nueva sintaxis para las consultas multitabla: las **joins** (o composiciones) internas, externas y productos cartesianos (también llamadas composiciones cruzadas):

1. **Join** interna.
   - De equivalencia (INNER JOIN)
   - Natural (NATURAL JOIN)

2. Producto Cartesiano (CROSS JOIN)
3. **Join** Externa
   De tabla derecha (RIGHT OUTER JOIN)
   De tabla izquierda (LEFT OUTER JOIN)
   Completa (FULL OUTER JOIN)

Composiciones internas. INNER JOIN

Hay **dos formas** diferentes para expresar las INNER JOIN o composiciones internas. La primera, usa la palabra reservada JOIN, mientras que la segunda usa "," para separar las tablas a combinar en la sentencia FROM, es decir, las de SQL 1.

Con la operación **INNER JOIN** se calcula el producto cartesiano de todos los registros, después, cada registro en la primera tabla es combinado con cada registro de la segunda tabla, y solo se seleccionan aquellos registros que satisfacen las condiciones que se especifiquen en la cláusula **ON**. Hay que tener en cuenta que **los valores Nulos no se combinan**.

```
SELECT * FROM animales INNER JOIN propietarios
  ON animales.propietario = propietarios.dni;
+--------+----------+-------+-------------+-----------+--------------------+
| codigo | nombre   | tipo  | propietario | dni       | nombre             |
+--------+----------+-------+-------------+-----------+--------------------+
|      1 | Cloncho  | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      2 | Yoda     | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      3 | Sprocket | perro | 37276317Z   | 37276317Z | Francisco Martínez |
+--------+----------+-------+-------------+-----------+--------------------+

# Nótese que es una consulta equivalente a la vista en el apartado anterior
# select * from animales,propietarios
#   where animales.propietario=propietarios.dni;
```

si hubiera algún animal sin propietario, no saldría.

```
INSERT INTO animales VALUES (null,'Arco','perro',null);
SELECT * FROM animales;
+--------+----------+-------+-------------+
| codigo | nombre   | tipo  | propietario |
+--------+----------+-------+-------------+
|      1 | Cloncho  | gato  | 51993482Y   |
|      2 | Yoda     | gato  | 51993482Y   |
|      3 | Sprocket | perro | 37276317Z   |
|      4 | Arco     | perro | NULL        |  #nueva mascota sin propietario
+--------+----------+-------+-------------+

SELECT * FROM animales INNER JOIN propietarios
   ON animales.propietario = propietarios.dni;
+--------+----------+-------+-------------+-----------+--------------------+
| codigo | nombre   | tipo  | propietario | dni       | nombre             |
+--------+----------+-------+-------------+-----------+--------------------+
|      1 | Cloncho  | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      2 | Yoda     | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      3 | Sprocket | perro | 37276317Z   | 37276317Z | Francisco Martínez |
+--------+----------+-------+-------------+-----------+--------------------+
```

Variantes con el filtro

```
SELECT * FROM animales INNER JOIN propietarios
   ON propietarios.dni >= animales.propietario;
+--------+----------+-------+------------+-----------+--------------------+
| codigo | nombre   | tipo  | propietario | dni       | nombre             |
+--------+----------+-------+------------+-----------+--------------------+
|      1 | Cloncho  | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      2 | Yoda     | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      3 | Sprocket | perro | 37276317Z   | 51993482Y | José Pérez         |
|      3 | Sprocket | perro | 37276317Z   | 37276317Z | Francisco Martínez |
+--------+----------+-------+------------+-----------+--------------------+
```

Composiciones naturales. NATURAL JOIN

Es una especialización de la INNER JOIN. En este caso se comparan todas las columnas que tengan el mismo nombre en ambas tablas. La tabla resultante contiene solo una columna por cada par de columnas con el mismo nombre.

```
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| CodigoEmpleado | int(11)     | NO   | PRI | NULL    |       |
| Nombre         | varchar(50) | NO   |     | NULL    |       |
| Apellido1      | varchar(50) | NO   |     | NULL    |       |
| Apellido2      | varchar(50) | YES  |     | NULL    |       |
| Extension      | varchar(10) | NO   |     | NULL    |       |
| Email          | varchar(100)| NO   |     | NULL    |       |
| CodigoOficina  | varchar(10) | NO   |     | NULL    |       | #relación
| CodigoJefe     | int(11)     | YES  |     | NULL    |       |
| Puesto         | varchar(50) | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
DESCRIBE Oficinas;
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| CodigoOficina   | varchar(10) | NO   | PRI | NULL    |       | #relación
| Ciudad          | varchar(30) | NO   |     | NULL    |       |
| Pais            | varchar(50) | NO   |     | NULL    |       |
| Region          | varchar(50) | YES  |     | NULL    |       |
| CodigoPostal    | varchar(10) | NO   |     | NULL    |       |
| Telefono        | varchar(20) | NO   |     | NULL    |       |
| LineaDireccion1 | varchar(50) | NO   |     | NULL    |       |
| LineaDireccion2 | varchar(50) | YES  |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+

#NATURAL JOIN coge los mismos nombres de campo, en este caso CodigoOficina

SELECT CodigoEmpleado,Empleados.Nombre,
   Oficinas.CodigoOficina,Oficinas.Ciudad
      FROM Empleados NATURAL JOIN Oficinas;
+----------------+-----------------+---------------+---------------------+
| CodigoEmpleado | Nombre          | CodigoOficina | Ciudad              |
+----------------+-----------------+---------------+---------------------+
|              1 | Marcos          | TAL-ES        | Talavera de la Reina |
|              2 | Ruben           | TAL-ES        | Talavera de la Reina |
....
|             31 | Mariko          | SYD-AU        | Sydney              |
+----------------+-----------------+---------------+---------------------+
```

Producto cartesiano. CROSS JOIN

```
#equivalente a SELECT * FROM animales,propietarios;
SELECT * FROM animales CROSS JOIN propietarios;
+--------+----------+-------+-------------+-----------+--------------------+
| codigo | nombre   | tipo  | propietario | dni       | nombre             |
+--------+----------+-------+-------------+-----------+--------------------+
|      1 | Cloncho  | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      1 | Cloncho  | gato  | 51993482Y   | 2883477X  | Matías Fernández   |
|      1 | Cloncho  | gato  | 51993482Y   | 37276317Z | Francisco Martínez |
|      2 | Yoda     | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      2 | Yoda     | gato  | 51993482Y   | 2883477X  | Matías Fernández   |
|      2 | Yoda     | gato  | 51993482Y   | 37276317Z | Francisco Martínez |
|      3 | Sprocket | perro | 37276317Z   | 51993482Y | José Pérez         |
|      3 | Sprocket | perro | 37276317Z   | 2883477X  | Matías Fernández   |
|      3 | Sprocket | perro | 37276317Z   | 37276317Z | Francisco Martínez |
|      4 | Arco     | perro | NULL        | 51993482Y | José Pérez         |
|      4 | Arco     | perro | NULL        | 2883477X  | Matías Fernández   |
|      4 | Arco     | perro | NULL        | 37276317Z | Francisco Martínez |
+--------+----------+-------+-------------+-----------+--------------------+
```

Composiciones externas. OUTER JOIN.

Las tablas relacionadas no requieren que haya una equivalencia. El registro es seleccionado para ser mostrado aunque no haya otro registro que le corresponda.

OUTER JOIN se subdivide dependiendo de la tabla a la cual se le admitirán los registros que no tienen correspondencia, ya sean de tabla izquierda, de tabla derecha, o combinación completa.

```
SELECT * FROM animales LEFT OUTER JOIN propietarios
  ON animales.propietario = propietarios.dni;
+--------+----------+-------+-------------+-----------+--------------------+
| codigo | nombre   | tipo  | propietario | dni       | nombre             |
+--------+----------+-------+-------------+-----------+--------------------+
|      1 | Cloncho  | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      2 | Yoda     | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      3 | Sprocket | perro | 37276317Z   | 37276317Z | Francisco Martínez |
|      4 | Arco     | perro | NULL        | NULL      | NULL               |
+--------+----------+-------+-------------+-----------+--------------------+
```

Si los registros que admiten no tener correspondencia son los que aparecen en la tabla de la derecha, se llama composición de tabla derecha RIGHT JOIN (o RIGHT OUTER JOIN):

```
#ejemplo de RIGHT OUTER JOIN
#animales RIGHT OUTER JOIN propietarios
#animales está a la izquierda
#propietarios está a la derecha
SELECT * FROM animales RIGHT OUTER JOIN propietarios
      ON animales.propietario = propietarios.dni;
+--------+----------+-------+-------------+-----------+--------------------+
| codigo | nombre   | tipo  | propietario | dni       | nombre             |
+--------+----------+-------+-------------+-----------+--------------------+
|      1 | Cloncho  | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      2 | Yoda     | gato  | 51993482Y   | 51993482Y | José Pérez         |
|   NULL | NULL     | NULL  | NULL        | 2883477X  | Matías Fernández   |
|      3 | Sprocket | perro | 37276317Z   | 37276317Z | Francisco Martínez |
+--------+----------+-------+-------------+-----------+--------------------+
```

La operación que admite registros sin correspondencia tanto para la tabla izquierda como para la derecha, por ejemplo, animales sin propietario y propietarios sin animales, se llama composición externa completa o FULL JOIN (FULL OUTER JOIN)

```
#ejemplo de FULL OUTER JOIN
#animales FULL OUTER JOIN propietarios
#animales está a la izquierda
#propietarios está a la derecha
SELECT * FROM animales FULL OUTER JOIN propietarios
       ON animales.propietario = propietarios.dni;
+--------+----------+-------+-------------+-----------+--------------------+
| codigo | nombre   | tipo  | propietario | dni       | nombre             |
+--------+----------+-------+-------------+-----------+--------------------+
|      1 | Cloncho  | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      2 | Yoda     | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      3 | Sprocket | perro | 37276317Z   | 37276317Z | Francisco Martínez |
|      4 | Arco     | perro | NULL        | NULL      | NULL               |
|   NULL | NULL     | NULL  | NULL        | 2883477X  | Matías Fernández   |
+--------+----------+-------+-------------+-----------+--------------------+
```

¿Sabías que . . . ? En SQL existe el operador UNION, que añade al conjunto de resultados producidos por una SELECT, los resultados de otra SELECT. La sintaxis es:

```
SELECT .... FROM ....
UNION [ALL]
SELECT .... FROM ....
```

El parámetro ALL incluye todos los registros de las dos SELECT, incluyendo los que son iguales. Si no se indica ALL, se excluyen los duplicados.

```
mysql> SELECT * FROM animales LEFT OUTER JOIN propietarios
    ->    ON animales.propietario = propietarios.dni
    -> UNION
    -> SELECT * FROM animales RIGHT OUTER JOIN propietarios
    ->    ON animales.propietario = propietarios.dni;
+--------+----------+-------+-------------+-----------+--------------------+
| codigo | nombre   | tipo  | propietario | dni       | nombre             |
+--------+----------+-------+-------------+-----------+--------------------+
|      1 | Cloncho  | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      2 | Yoda     | gato  | 51993482Y   | 51993482Y | José Pérez         |
|      3 | Sprocket | perro | 37276317Z   | 37276317Z | Francisco Martínez |
|      4 | Arco     | perro | NULL        | NULL      | NULL               |
|   NULL | NULL     | NULL  | NULL        | 2883477X  | Matías Fernández   |
+--------+----------+-------+-------------+-----------+--------------------+
```

CONSULTAS REFLEXIVAS

A veces, es necesario obtener información de relaciones reflexivas, por ejemplo, un informe de empleados con su nombre y apellidos y el nombre y apellidos de su jefe.

```
mysql> desc Empleados;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| CodigoEmpleado | int(11)      | NO   | PRI | NULL    |       |
| Nombre         | varchar(50)  | NO   |     | NULL    |       |
| Apellido1      | varchar(50)  | NO   |     | NULL    |       |
| Apellido2      | varchar(50)  | YES  |     | NULL    |       |
| Extension      | varchar(10)  | NO   |     | NULL    |       |
| Email          | varchar(100) | NO   |     | NULL    |       |
| CodigoOficina  | varchar(10)  | NO   |     | NULL    |       |
| CodigoJefe     | int(11)      | YES  |     | NULL    |       | #autorelación
| Puesto         | varchar(50)  | YES  |     | NULL    |       |
+----------------+--------------+------+-----+---------+-------+
```

```
SELECT concat(emp.Nombre,', ' ,emp.Apellido1) as Empleado,
concat(jefe.Nombre,', ',jefe.Apellido1) as jefe

FROM Empleados emp INNER JOIN Empleados jefe ON emp.CodigoEmpleado =
jefe.CodigoJefe;
```

```
+------------------+-------------------------+
| Empleado         | jefe                    |
+------------------+-------------------------+
| Marcos Magaña    | Ruben López             |
| Ruben López      | Alberto Soria           |
....
| Alberto Soria    | Kevin Fallmer           |
| Kevin Fallmer    | Julian Bellinelli       |
| Kevin Fallmer    | Mariko Kishi            |
+------------------+-------------------------+
```

CONSULTAS CON TABLAS DERIVADAS

Las consultas con tablas derivadas, *o inline views,* son aquellas que utilizan sentencias SELECT en la cláusula FROM en lugar de nombres de tablas.

```
SELECT * FROM
        (SELECT CodigoEmpleado, Nombre FROM Empleados
         WHERE CodigoOficina='TAL-ES') as tabla_derivada;
```

En este caso se ha de distinguir, por un lado la tabla derivada, (*SELECT CodigoEmpleado, Nombre FROM Empleados)* que tiene un alias *tabla_derivada*, es decir, una especie de tabla temporal cuyo contenido es el resultado de ejecutar la consulta, su nombre es *tabla_derivada* y tiene dos columnas, una *CodigoEmpleado* y otra Nombre. Este tipo de consultas ayudará a obtener información relacionada de forma mucho más avanzada.

Ejemplo. Sacar el pedido de menor importe.

```
#1: Para calcular el total de cada pedido, hay que codificar esta query
SELECT SUM(Cantidad*PrecioUnidad) as total,CodigoPedido
   FROM DetallePedidos
   GROUP BY CodigoPedido;
+-------+--------------+
| total | CodigoPedido |
+-------+--------------+
|  1567 |            1 |
|  7113 |            2 |
| 10850 |            3 |
....
|   154 |          117 |
|    51 |          128 |
+-------+--------------+

#2: Para calcular el menor pedido, se puede hacer una tabla
#  derivada de la consulta anterior y con la función MIN
#  obtener el menor de ellos:
SELECT MIN(total) FROM (
    SELECT SUM(Cantidad*PrecioUnidad) as total,CodigoPedido
       FROM DetallePedidos
       GROUP BY CodigoPedido
    ) AS TotalPedidos;
+------------+
| MIN(total) |
+------------+
|          4 |
+------------+
#TotalPedidos es la tabla derivada formada
#por el resultado de la consulta entre paréntesis
```

Se pueden concatenar más tablas derivadas.

LAS VISTAS

Una vista es una tabla virtual que devuelve filas obtenidas de una consulta SQL. La diferencia entre una sentencia SQL y una vista, estriba en que en cada sentencia SQL se hace una petición al SGBD y una compilación de la sentencia, mientras que en una vista la consulta está almacenada y compilada en la BBDD. También puede que un usuario no pueda acceder a todas las tablas que están implicadas en la vista pero sí a la vista que contiene algunas columnas de la tabla prohibida.

SINTAXIS

```
CREATE [OR REPLACE] VIEW [esquema.]nombre_vista [(lista_columnas)] AS
sentencia_select
```

EJEMPLO.

```
mysql> CREATE VIEW nba.jugadoresMiami AS
    -> SELECT Nombre, Posicion FROM nba.jugadores WHERE Nombre_equipo='HEAT';
Query OK, 0 rows affected (0.01 sec)
```

Una vez creada se puede usar como una tabla normal.

```
mysql> SELECT * FROM jugadoresMiami;
+-------------------+----------+
| Nombre            | Posicion |
+-------------------+----------+
| Blake Ahearn      | G        |
| joel Anthony      | C        |
| Marcus Banks      | G        |
| Earl Barron       | C-F      |
| Mark Blount       | C-F      |
| Daequan Cook      | G        |
| Ricky Davis       | F-G      |
| Udonis Haslem     | F        |
| Alexander Johnson | F        |
| Stephane Lasme    | F        |
| Shawn Marion      | F        |
| Alonzo Mourning   | C        |
| Chris Quinn       | G        |
| Dwyane Wade       | G        |
| Jason Williams    | G        |
| Dorell Wright     | F        |
+-------------------+----------+
16 rows in set (0.00 sec)
```

Para usuarios menos expertos, se suele crear vistas útiles de consultas complejas.

```
mysql> USE JARDINERIA;
Database changed
mysql> CREATE VIEW VistaPedidos (CodigoPedido, Cliente,Total) AS
    -> SELECT CodigoPedido, NombreCliente, SUM(Cantidad*PrecioUnidad)
    -> FROM Clientes NATURAL JOIN Pedidos NATURAL JOIN DetallePedidos
    -> GROUP BY CodigoPedido;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> SELECT * FROM VISTAPEDIDOS;
+--------------+--------------------------------+----------+
| CodigoPedido | Cliente                        | Total    |
+--------------+--------------------------------+----------+
|            1 | Tendo Garden                   |  1567.00 |
|            2 | Tendo Garden                   |  7113.00 |
|            3 | Tendo Garden                   | 10850.00 |
|            4 | Tendo Garden                   |  2624.00 |
|            8 | DGPRODUCTIONS GARDEN           |  1065.00 |
|            9 | DGPRODUCTIONS GARDEN           |  2535.00 |
|           10 | Gardening Associates           |  2920.00 |
|           11 | DGPRODUCTIONS GARDEN           |   820.00 |
|           12 | DGPRODUCTIONS GARDEN           |   290.00 |
```

Para eliminar una vista usamos el comando **DROP VIEW**

```
DROP VIEW [esquema.]nombre_vista;
```

Para modificar una vista usamos el comando **ALTER VIEW**

```
ALTER VIEW [esquema.]nombre_vista [(lista_columnas)] AS
sentencia_select
```

Para mostrar las vistas tenemos dos opciones:

```
SELECT table_name FROM information_schema.tables WHERE table_type='VIEW';
```

```
SHOW FULL TABLES;
```

SENTENCIA INSERT

La sentencia INSERT de SQL permite insertar una fila en una tabla, es decir, añadir un registro de información a una tabla.

El formato de uso es muy sencillo:

```
INSERT [INTO] nombre_tabla [(nombre_columna, . . . )]
VALUES ({expr | DEFAULT},...)
```

nombre_tabla es el nombre de la tabla donde se quiere insertar la fila. Después del nombre de la tabla, de forma optativa, se pueden indicar las columnas donde se va a insertar la información. Si se especifican las columnas, la lista de valores (VALUES) a insertar se asociará correlativamente con los valores a las columnas indicadas. Si no se especifican las columnas, la lista de valores se escribirá conforme al orden de las columnas en la definición de la tabla. A continuación se muestran unos cuantos ejemplos:

```
#INSERT especificando la lista de columnas
INSERT INTO mascotas (Codigo, Nombre, Raza)
VALUES
(1,'Paquitas','Gato Común Europeo')
```

```
#INSERT sin especificar la lista de columnas.
INSERT INTO mascotas VALUES
(2, 'Calcetines', 'Gato Común Europeo', '59932387L')
```

En este caso hay pasar un valor para cada una de las columnas de la tabla.

```
#INSERT con columnas con valores por defecto
INSERT INTO vehiculos VALUES ('1215 BCD','Toledo TDI', DEFAULT);
```

Si el número de columnas especificadas no coincide con el número de valores o no coinciden los tipos de datos, dará un error.

```
#INSERT con columnas con valores por defecto
INSERT INTO vehiculos VALUES ('1215 BCD','Toledo TDI', DEFAULT);
```

SENTENCIA INSERT EXTENDIDA

La sintaxis extendida de INSERT para gestores tipo MySQL es la siguiente:

```
INSERT [INTO] nombre_tabla [(nombre.columna,. . .)]
VALUES ({expr | DEFAULT},...),(...),..
```

Ejemplo

```
insert into vehiculos (Matricula,Modelo,Marca)
VALUES ('4123 BFH','Ibiza','Seat'),
('1314 FHD','Toledo','Seat'),
('3923 GJS','León','Seat');
```

INSERT Y SELECT

Una variante de la sentencia INSERT consiste en una utilizar la sentencia SELECT para obtener un conjunto de datos y, posteriormente, insertarlos en la tabla.

```
INSERT
[INTO] noinbre_tabla [(nombre_columna,.,)]
```

```
SELECT ... FROM ...
```

Ejemplo

```
#Inserta en una tabla Backup todos los vehículos
INSERT INTO BackupVehiculos
SELECT * FROM vehiculos;
```

La sentencia SELECT debe devolver tantas columnas como columnas tenga la

tabla donde se introduce la información. La sentencia SELECT puede ser tan compleja como se desee, con tablas derivas, multitablas, agrupaciones , ordenaciones, filtros, etc

SENTENCIA UPDATE

La sentencia UPDATE de SQL permite modificar el contenido de cualquier columna y de cualquier fila de una tabla. Su sintaxis es la siguiente:

```
UPDATE nombre_tabla
SET nombre_col1=exprl [, nombre_col2=expr2 ] ...
[WHERE filtro]
```

filtro es una expresión booleana que indica la condición o condiciones que deben satisfacer los registros para ser actualizados.

Ejemplo

```
#Actualiza una de columna de una sola fila
UPDATE jugadores SET Nombre_equipo='Knicks'
WHERE Nombre='Pau Gasol';

#Actualiza dos columnas de una misma fila
UPDATE jugadores SET Nombre_equipo='Knicks', Peso=210
WHERE Nombre='Pau Gasol';
```

Si no se pone filtro, actualizará toda la tabla.

```
UPDATE jugadores SET Peso=Peso*0.4535;
```

SENTENCIA DELETE

Se usa para eliminar filas de una tabla

```
DELETE FROM nombre_tabla
[WHERE filtro]
```

Ejemplo

```
DELETE FROM jugadores WHERE Nombre='Jorge Garbajosa';
```

Si se omite el filtro, elimina todas las filas de la tabla, pero no la tabla.

```
DELETE FROM jugadores;
```

BORRADO Y MODIFICACIÓN CON RELACIONES

Habrá situaciones donde el borrado o la modificación entre en conflicto con las restricciones establecidas en la creación de la BBDD. Por ejemplo, no podremos eliminar clientes con pagos pendientes, si se ha establecido la restricción de clave foránea en la tabla de pagos. Si en la restricción establecemos alguna de las acciones : CASCADE o SET NULL, se realizará la actualización correspondiente.

```
#dos tablas relacionadas en mysql
#han de ser innodb para soportar FOREING KEYS
CREATE TABLE clientes (
   dni varchar(15) PRIMARY KEY,
   nombre varchar(50),
   direccion varchar(50)
) engine=innodb;

CREATE TABLE pagos_pendientes(
   dni varchar(15),
   importe double,
   FOREIGN KEY(dni) REFERENCES clientes(dni)
            on delete NO ACTION
         on update NO ACTION
) engine=innodb;

#un cliente y dos pagos pendientes
INSERT INTO clientes
  VALUES ('5555672L','Pepe Cifuentes','C/Los almendros,23');
INSERT INTO pagos_pendientes VALUES ('5555672L',500);
INSERT INTO pagos_pendientes VALUES ('5555672L',234.5);

#Se intenta borrar el cliente y no es posible
DELETE FROM clientes WHERE dni='5555672L';
ERROR 1451 (23000): Cannot delete or update a parent row:
 a foreign key constraint fails (`gestion/pagos_pendientes`,
  CONSTRAINT `pagos_pendientes_ibfk_1`
  FOREIGN KEY (`dni`) REFERENCES `clientes` (`dni`))

#Se intenta modificar el dni del cliente y no lo permite
UPDATE clientes set dni='55555555L' WHERE dni='5555672L';
ERROR 1451 (23000): Cannot delete or update a parent row:
a foreign key constraint fails (`gestion/pagos_pendientes`,
  CONSTRAINT `pagos_pendientes_ibfk_1`
  FOREIGN KEY (`dni`) REFERENCES `clientes` (`dni`))

#de igual modo si se intenta borrar la tabla clientes,
#tampoco podemos
DROP TABLE clientes;
ERROR 1217 (23000): Cannot delete or update
a parent row: a foreign key constraint fails
```

En este otro caso, se elimina y actualiza en cascada.

```
#dos tablas relacionadas en mysql
create table clientes (
   dni varchar(15) primary key,
   nombre varchar(50),
   direccion varchar(50)
) engine=innodb;
create table pagos_pendientes(
   dni varchar(15),
   importe double,
   foreign key (dni) references clientes(dni)
            on delete CASCADE on update CASCADE
) engine=innodb;

#un cliente y dos pagos pendientes
INSERT INTO clientes
        values ('5555672L','Pepe Cifuentes','C/Los almendros,23');
INSERT INTO pagos_pendientes VALUES ('5555672L',500);
INSERT INTO pagos_pendientes VALUES ('5555672L',234.5);

#se borra el cliente...
DELETE FROM clientes WHERE dni='5555672L';
Query OK, 1 row affected (0.00 sec)

#además, se verifica que ha borrado en cascada sus pagos pendientes.
SELECT * FROM pagos_pendientes;
Empty set (0.00 sec)

#si en lugar de borrar el cliente, se hubiera cambiado el dni:
UPDATE clientes set dni='55555555L' WHERE dni='5555672L';
Query OK, 1 row affected (0.02 sec)

#ha cambiado el dni de los pagos en cascada.
SELECT  * FROM pagos_pendientes;
+-----------+---------+
| dni       | importe |
+-----------+---------+
| 55555555L |     500 |
| 55555555L |   234.5 |
+-----------+---------+
```

Actividad 1. Prueba las opciones CASCADE y SET NULL en la base de datos de mascotas.

UPDATE Y DELETE CON SUBCONSULTAS

Es posible actualizar o borrar registros de una tabla filtrando a través de una subconsulta. La única limitación es que hay gestores que no permiten realizar cambios en la tabla que se está leyendo a través de la subconsulta.

Ejemplo

```
#Elimina los representantes que no tengan clientes
DELETE FROM Empleados
WHERE CodigoEmpleado Not in
(SELECT CodigoEmpleadoRepVentas
FROM Clientes)
AND Puesto='Representante Ventas';
```

Sin embargo, la siguiente consulta dará error, porque la subconsulta incluye la tabla de donde se van a eliminar los registros.

```
DELETE FROM Clientes
WHERE CodigoCliente in
(SELECT CodigoCliente
FROM Clientes WHERE LimiteCredito=0);
ERROR 1093 (HYOOO): Yon can't specify taxget table 'Clientes' for
update in FROM clanse
```

SCRIPT EN MYSQL

Los script de MySQL son archivos de texto, con la extensión *.sql*, que contienen sentencias SQL de edición y/o creación. Se usan para automatizar tareas .

Para ejecutar un script de MySQL usamos la orden:

```
source archivo_script;
```