Universidad
Francisco de Vitoria
UFV Madrid

# Fundamentos de la ingeniería informática

Ingeniería de sistemas industriales

Curso 2019-2020

# SQL

# 1. SQL

SQL (Structured Query Language) es un lenguaje de aplicación específica a la administración de bases de datos que permite añadir, recuperar, modificar y eliminar registros en tablas y tablas en bases de datos relacionales, en base a las definiciones del álgebra relacional.

Originalmente basado en el álgebra relacional y en el cálculo relacional, SQL consiste en un lenguaje de definición de datos, un lenguaje de manipulación de datos y un lenguaje de control de datos.

Desde su conversión en estándar por el ANSI 1986 han aparecido múltiples versiones (dialectos): SQL-88, SQL-91, SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011 y SQL:2014; que, aunque similares en lo fundamental, cada una de ellas tiene sus detalles particulares.

Aunque no sea coherente, cualquier sentencia completa individual de este lenguaje se denomina consulta, no sólo aquellas que recuperan información, sino también las que insertan datos, borran registros o los modifican.

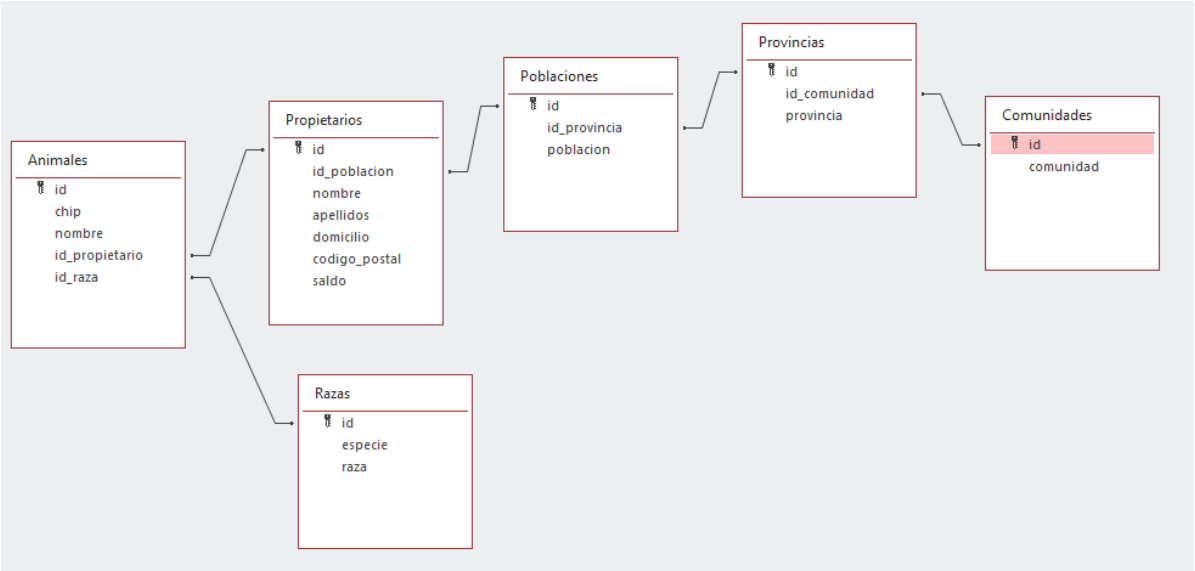
El resultado de una consulta tiene la forma de una tabla: filas para los registros recuperados y columnas para los campos solicitados de esos registros.

Por otra parte, vamos a utilizar el término “vista” para referirnos bien una tabla o al resultado de una consulta.

SQL no es sensible a mayúsculas y minúsculas, pero es una buena práctica tomar un criterio de etiquetado que nos facilite la lectura de diseños y consultas. Por ejemplo: Los nombres de las tablas serán siempre sustantivos con la primera letra mayúscula y las palabras separadas por “_”, los campos sustantivos con todas sus letras en minúsculas y “_” en vez de espacios. Las palabras clave de SQL en mayúsculas.

# 2. Base ejemplo

En los siguientes apuntes tomaremos como ejemplo la base de datos “Clínica”, la cual hemos construido en el tema anterior. Este es su diseño físico:


Animales
id
chip
nombre
id_propietario
id_raza
Propietarios
id
id_poblacion
nombre
apellidos
domicilio
codigo_postal
saldo
Poblaciones
id
id_provincia
poblacion
Provincias
id
id_comunidad
provincia
Comunidades
id
comunidad
Razas
id
especie
raza

# 3. SELECT

La operación fundamental en cualquier SGBD es la recuperación de información que en SQL corresponde a la sentencia SELECT. Su ejecución ofrece como resultado el grupo de campos especificados de un conjunto de registros, que cumplen ciertas condiciones, obtenidos de la composición de tablas y/o otras consultas.

Aún en su forma básica es muy compleja y corresponde al siguiente formato

```
SELECT [ALL | DISTINCT ] <campo> [{,<campo>}]

FROM <tabla>|<consulta> [{,<tabla>|<consulta>}]

[WHERE <condición> [{AND|OR <condición>}]]

[GROUP BY <campo> [{,<campo >}]]

[HAVING <condición>[{AND|OR <condición>}]]

[ORDER BY <campo> [ASC | DESC] [{,<campo> [ASC | DESC]}]]
```

En negrita se encuentran las palabras clave:

- etiquetas cláusulas: SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY)
- operadores: AND, OR, AS, LIKE...
- modificadores: ALL, DISTINC, ASC, DSC...

Los paréntesis cuadrados ( [ y ] ) señalan los elementos opcionales de la sintaxis. Las llaves ( { y } ) enmarcan fragmentos que se repiten una o más veces. La barra ( | ) separa dos alternativas para un único elemento. Los ángulos (< y >)

Considérese que SELECT es tanto una cláusula como el tipo de sentencia que comienza por ella.

Una clausula es una pieza de una sentencia. Comienza por una etiqueta (palabra reservada) a la que siguen algunos parámetros.

## 3.1. Sencillo (sencillo1)

Empecemos por lo más sencillo escribiendo sólo lo que no es opcional.

Toda sentencia de recuperación tiene que especificar al menos un campo a recuperar y al menos una tabla de la cual recuperarla. Ni que decir tiene que ese campo debe ser un campo de la tabla.

**SELECT** *<campo>* **FROM** *<tabla>*

Ejemplo:

```
SELECT nombre FROM Animales
```

Esto significa seleccionar el campo nombre de todos los registros de la tabla Animales.

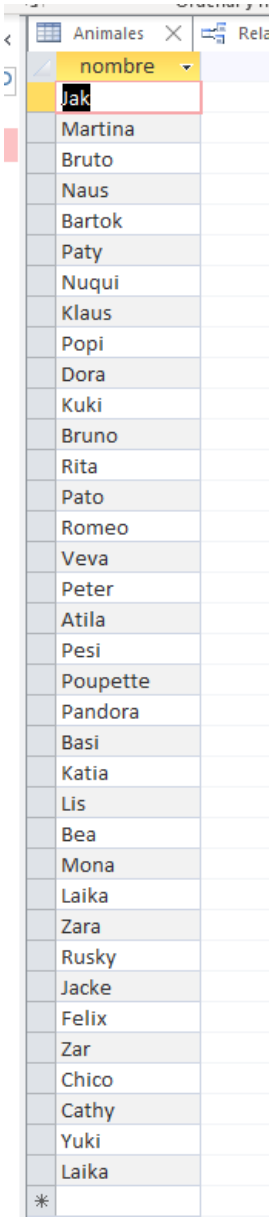
¿Por qué el campo nombre?

Porque es el único que aparece en la lista de campos que constituyen los parámetros de la cláusula SELECT. Si se referenciaran más campos (separados por comas) esos serían los recuperados.

¿Por qué la tabla Animales?

Porque es la indicada en la cláusula FROM. Sólo se permite hacer consulta (recuperar información) de una vista. Puede confundirnos el ver en alguna ocasión tablas separadas por comas en una cláusula FROM (FROM A, B, C), esa es la forma de expresar una consulta en la que se obtiene la combinación de todos los registros de A con todos los de B con todos los de C.

El resultado de ejecutar la sentencia anterior sobre la base Clínica es una vista como la siguiente:



## 3.2. Otro ejemplo (sencillo2)

Si queremos obtener todos los nombres y apellidos de los propietarios de la clínica la consulta a realizar es:

```
SELECT nombre, apellidos FROM Propietarios
```

Parte del resultado que se obtiene es:

| nombre | apellidos |
|---|---|
| LEANDRO | LOPEZ VEIGUELA |
| CONCHA | GOMEZ CARDIN |
| NOELIA | PERNAUTE |
| OLGA | PERNAUTE |
| NIEVES | RABASSA GARCIA |
| JOSE | CLAVIJO QUESADA |
| PILAR | BOUZAS VITURRO |
| DAVID | PALOMO RUIZ |
| MERCEDES | RODRIGUEZ ANDIA-ROMERO |
| ALFONSO | GOMEZ MARFIL |
| ADRIANA | VIDAL GARCIA-VALCARCEL |
| LUZ | DE HARO |

## 3.3. Ordenación (ordenado1)

La clausula ORDER BY permite que la vista tenga un orden determinado siguiendo alguno de los campos. Así:

```
SELECT nombre, apellidos FROM Propietarios ORDER BY nombre
```

nos proporcionará una vista de la misma información que en la sección anterior pero ordenada por el nombre del propietario de menor a mayor.

| nombre | apellidos |
|---|---|
| ABEL | CABRERA MARTINEZ |
| ADRIANA | VIDAL GARCIA-VALCARCEL |
| ALFONSO | GOMEZ MARFIL |
| ALFONSO | MORITZ |
| ANA | GARCIA DURAN |
| ASUNCION | KHALED |
| CARMEN | GUIBELALDE |
| CONCHA | GOMEZ CARDIN |
| DAVID | PALOMO RUIZ |
| DOLORES | PIZARRO |
| DOMINIQUE | GARCIA MADRUGA |
| ENRIQUE | LORENZ PÉREZ |

# 3.4. Ordenación inversa (ordenado2)

Pero si quisiéramos que el orden fuera el contrario sólo habría que añadir el modificador **DESC** (descendente) a la cláusula **ORDER BY**.

**SELECT** nombre, apellidos **FROM** Propietarios **ORDER BY** nombre **DESC**

| nombre | apellidos |
|---|---|
| VIRGINIA | MUÑOZ BLANCO |
| TERESA | CANO MEDINA |
| SAAD | MARINA CUELI |
| RAQUEL | TAMAYO ORIOL LUIS |
| RAQUEL | CABRERA TORRES |
| RAFAEL | ELSER |
| PILAR | BOUZAS VITURRO |
| OLGA | PERNAUTE |
| NOELIA | PERNAUTE |

El modificador **ASC** (ascendente) es el modificador predeterminado de la cláusula **ORDER BY**, por lo que no es necesario ponerlo en la consulta de la sección anterior.

# 3.5. Ordenación oculta (ordenado3)

No es necesario que el campo por el que se ordena forme parte de los campos recuperados.

**SELECT** nombre, apellidos **FROM** Propietarios **ORDER BY** codigo_postal

Ordenado3

| nombre | apellidos |
|---|---|
| ENRIQUE | LORENZ PÉREZ |
| ESTHER | FERNANDEZ DE LA ALDEA |
| ABEL | CABRERA MARTINEZ |
| NIEVES | RABASSA GARCIA |
| RAFAEL | ELSER |
| MARIA | GARCIA DIAZ |
| MARTINEZ | GUERRERO |
| LEANDRO | LOPEZ VEIGUELA |
| DOLORES | PIZARRO |
| RAQUEL | CABRERA TORRES |
| FELIPE | GARCIA GARCIA |
| NOELIA | PERNAUTE |
| GUILLER | MURCIA ALCALA |
| ASUNCION | KHALED |

Parece no estar ordenado, pero si añadimos el campo a la consulta...

```
SELECT nombre, apellidos, codigo_postal
    FROM Propietarios ORDER BY codigo_postal
```

Ordenado3

| nombre | apellidos | codigo_post |
|---|---|---|
| ENRIQUE | LORENZ PÉREZ | 10301 |
| ESTHER | FERNANDEZ DE LA ALDEA | 10302 |
| ABEL | CABRERA MARTINEZ | 16001 |
| NIEVES | RABASSA GARCIA | 16001 |
| RAFAEL | ELSER | 16002 |
| MARIA | GARCIA DIAZ | 16002 |
| MARTINEZ | GUERRERO | 16003 |
| LEANDRO | LOPEZ VEIGUELA | 19001 |
| DOLORES | PIZARRO | 19001 |
| RAQUEL | CABRERA TORRES | 19001 |
| FELIPE | GARCIA GARCIA | 19002 |
| NOELIA | PERNAUTE | 20303 |
| GUILLER | MURCIA ALCALA | 25007 |
| ASUNCION | KHALED | 25007 |

## 3.6. Ordenación múltiple (ordenado4)

Pero podemos ordenar por varios campos. Por ejemplo, queremos ordenar por código postal (de mayor a menor), pero aquellos registros con el mismo código queremos que aparezcan ordenados por nombre, también de mayor a menor.

```
SELECT Propietarios.nombre, Propietarios.apellidos, Propietarios.codigo_postal
    FROM Propietarios
    ORDER BY Propietarios.codigo_postal DESC, Propietarios.nombre DESC;
```

Ordenado4

| nombre | apellidos | codigo_post |
|---|---|---|
| VIRGINIA | MUÑOZ BLANCO | 45022 |
| JAVIER | GALDON FERNANDEZ | 45022 |
| RAQUEL | TAMAYO ORIOL LUIS | 45001 |
| JOSE | CLAVIJO QUESADA | 45001 |
| Mª ANGELES | SANCHEZ GUERRA MARI | 28320 |
| FRANCISCO | RENNOTTE | 28232 |
| CARMEN | GUIBELALDE | 28232 |
| ADRIANA | VIDAL GARCIA-VALCARCEL | 28232 |
| TERESA | CANO MEDINA | 28230 |
| LOLA | RENNOTTE | 28230 |
| DAVID | PALOMO RUIZ | 28230 |
| PILAR | BOUZAS VITURRO | 28109 |
| OLGA | PERNAUTE | 28100 |
| ALFONSO | GOMEZ MARFIL | 28100 |
| ANA | GARCIA DURAN | 28045 |
| DOMINIQUE | GARCIA MADRUGA | 28022 |
| CONCHA | GOMEZ CARDIN | 28022 |
| MERCEDES | RODRIGUEZ ANDIA-ROMERO | 28012 |
| Mª CARMEN | EGUIGUREN HERMOSO | 28012 |

En este caso he optado por escribir el nombre completo del campo (<tabla>.<campo>). La tabla es opcional en la mayoría de los casos, aquellos en que no hay duda a que campo se refiere.

# 3.7. Ordenación calculada (ordenado5)

El criterio de ordenación puede ser un cálculo (operación de campos).

```
SELECT nombre, apellidos, Mid([codigo_postal],3,3) AS [o]
FROM Propietarios
ORDER BY Mid([codigo_postal],3,3)
```

La función mid(txt, i, l) aplicada a un texto (txt) devuelve el substring que comienza en la posición i y tiene una longitud l. Por tanto, Mid([codigo_postal],3,3) devuelve los tres últimos dígitos del código postal.

Ordenado5

| nombre | apellidos | codigo_post | o |
|---|---|---|---|
| RAQUEL | TAMAYO ORIOL LUIS | 45001 | 001 |
| RAQUEL | CABRERA TORRES | 19001 | 001 |
| NIEVES | RABASSA GARCIA | 16001 | 001 |
| JOSE | CLAVIJO QUESADA | 45001 | 001 |
| DOLORES | PIZARRO | 19001 | 001 |
| LEANDRO | LOPEZ VEIGUELA | 19001 | 001 |
| ABEL | CABRERA MARTINEZ | 16001 | 001 |
| RAFAEL | ELSER | 16002 | 002 |
| MARIA | GARCIA DIAZ | 16002 | 002 |
| FELIPE | GARCIA GARCIA | 19002 | 002 |
| MARTINEZ | GUERRERO | 16003 | 003 |
| GUILLER | MURCIA ALCALA | 25007 | 007 |
| ASUNCION | KHALED | 25007 | 007 |
| LUZ | DE HARO | 28008 | 008 |
| ALFONSO | MORITZ | 28008 | 008 |

Hemos utilizado por primera vez los paréntesis cuadrados ([ y ]) para delimitar nombres de campos, su uso es optativo salvo que el nombre contenga espacios.

La cláusula **AS** se utiliza para renombrar un campo de la vista, en este caso

Mid([codigo_postal],3,3) **AS** [o]

hace que el campo que contiene el resultado de Mid([codigo_postal],3,3) pueda ser referido como o en consultas que se hagan sobre ésta, pero no en la propia. El renombrado es opcional, pero si no se hace no podrá tenerse acceso al resultado de ese nuevo campo.

Mid([codigo_postal],3,3) es un ejemplo de como obtener información cuyos datos no son explícitos en las tablas.

# 3.8. Filtrado simple (filtrado1)

Hemos visto como obtener todos los registros de una vista, pero si lo que busco son aquellos que cumplen cierta condición he de añadir la cláusula **WHERE** a la consulta.

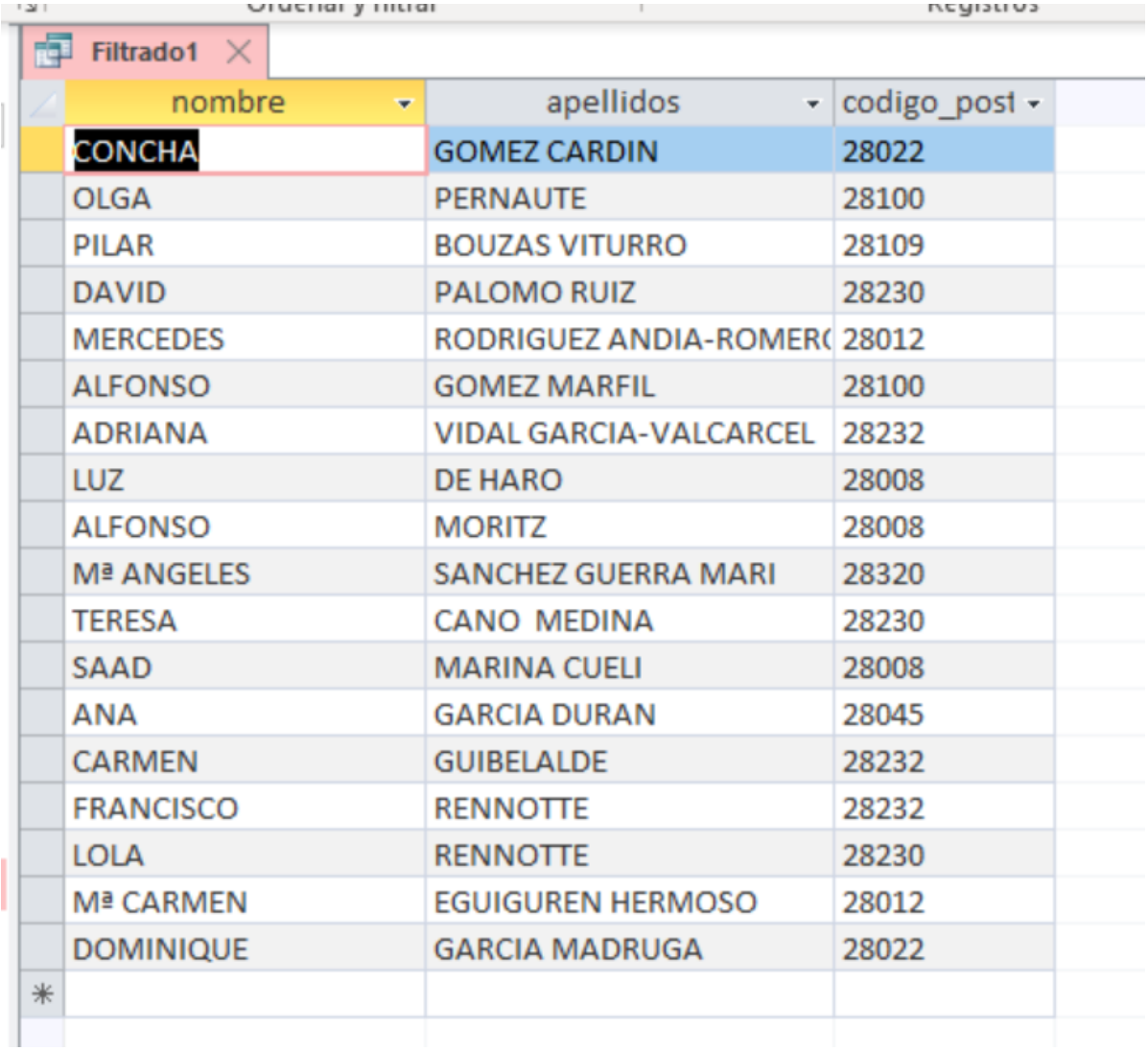
```
SELECT nombre, apellidos, codigo_postal
FROM Propietarios
WHERE (Left([codigo_postal],2)=28)
```

La función Left de un texto devuelve los n caracteres más a la izquierda de un texto (en este caso n=2), por lo que x tiene los dos primeros dígitos del código postal.

Por otra parte 28 es el código postal de la provincia de Madrid.

La cláusula **WHERE** va acompañada de una expresión booleana (condición) que cuando es cierta el registro al que pertenece se recupera y cuando es falsa el registro se ignora.

Así la sentencia anterior recupera los nombres, apellidos y códigos postales de todos los propietarios de Madrid, y el resultado es:



Filtrado1

| nombre | apellidos | codigo_post |
|---|---|---|
| CONCHA | GOMEZ CARDIN | 28022 |
| OLGA | PERNAUTE | 28100 |
| PILAR | BOUZAS VITURRO | 28109 |
| DAVID | PALOMO RUIZ | 28230 |
| MERCEDES | RODRIGUEZ ANDIA-ROMERO | 28012 |
| ALFONSO | GOMEZ MARFIL | 28100 |
| ADRIANA | VIDAL GARCIA-VALCARCEL | 28232 |
| LUZ | DE HARO | 28008 |
| ALFONSO | MORITZ | 28008 |
| Mª ANGELES | SANCHEZ GUERRA MARI | 28320 |
| TERESA | CANO MEDINA | 28230 |
| SAAD | MARINA CUELI | 28008 |
| ANA | GARCIA DURAN | 28045 |
| CARMEN | GUIBELALDE | 28232 |
| FRANCISCO | RENNOTTE | 28232 |
| LOLA | RENNOTTE | 28230 |
| Mª CARMEN | EGUIGUREN HERMOSO | 28012 |
| DOMINIQUE | GARCIA MADRUGA | 28022 |

## 3.9. Filtrado más complejo

Las condiciones se pueden complicar todo lo que sea necesario haciendo uso de los operadores AND, OR y NOT, y los comparadores >, < , = ,<>, <=, >=. Documéntate en la web sobre los mismos.

## 3.10. Comodines

Otra forma de realizar la consulta anterior sería

```
SELECT Propietarios.nombre, Propietarios.apellidos, Propietarios.codigo_postal
FROM Propietarios
WHERE (Propietarios.codigo_postal LIKE "28*").
```

El operador **LIKE** comprueba que el dato cumpla el patrón dado, en este caso, que el código postal comience por 28 seguido de otros caracteres cualquiera o ninguno. La función del * en esta condición la llamamos “comodín”.

Tal y como se ha expresado aquí, es para el SQL que funciona en ACCESS, en otros es distinto, por ejemplo, en SQL SERVER, el carácter comodín es el %.

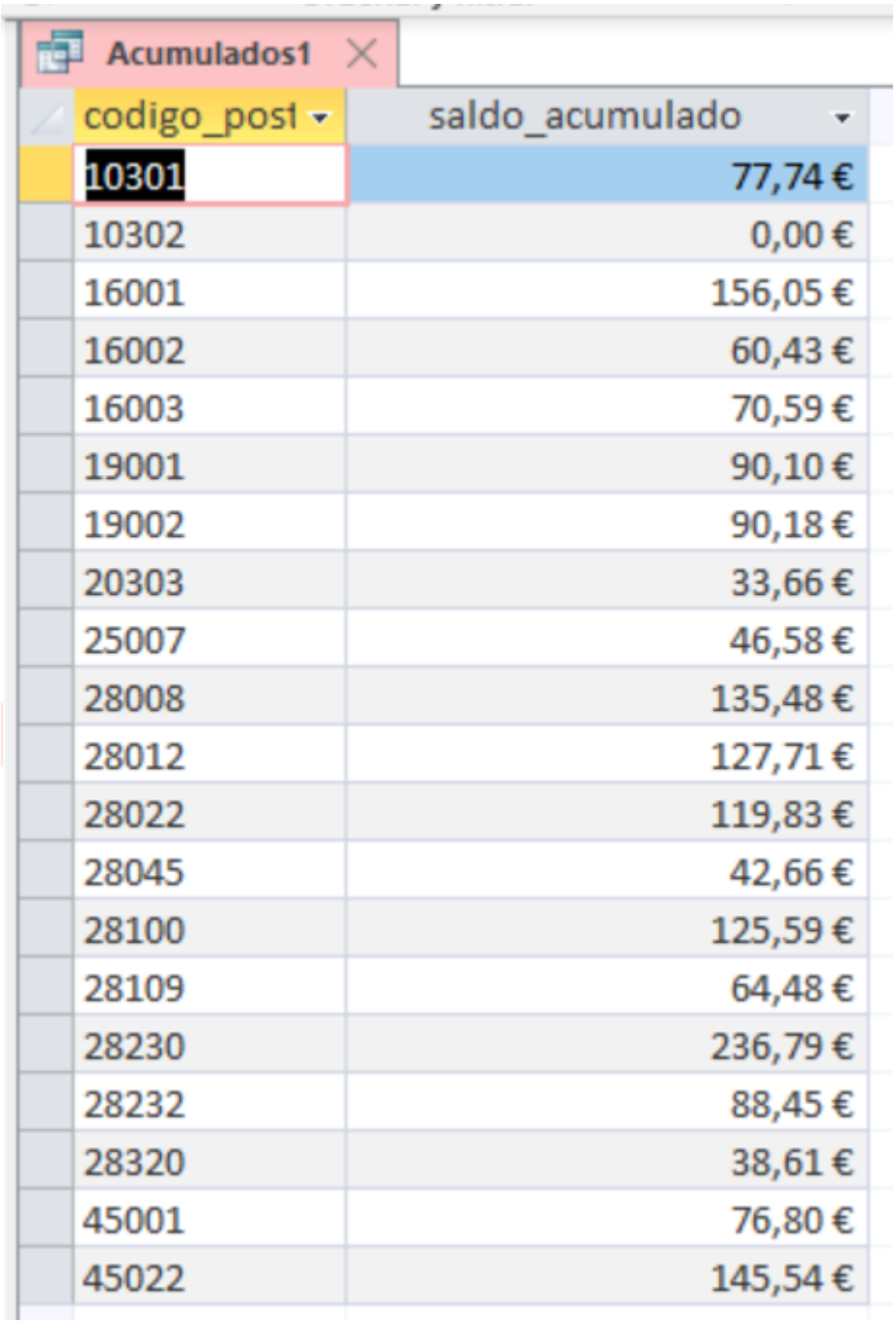
# 3.11. Acumulado

SQL nos permite agrupar registros siguiendo un criterio y obtener información acumulada de dichos grupos.

Si queremos saber cuál es el acumulado del saldo por distrito postal, la consulta a realizar será:

```
SELECT codigo_postal, Sum(saldo) AS saldo_acumulado
FROM Propietarios
GROUP BY codigo_postal
```

La cláusula GROUP BY reúne todos los registros (que cumplan los filtros si los hay) en grupos con el mismo valor del atributo argumento; en este caso del mismo código postal, y de cada grupo construye un registro con los datos solicitados del grupo: el código postal que es el mismo en todos los miembros y la suma de sus saldos (con la función SUM).



Acumulados1

| codigo_posi | saldo_acumulado |
|---|---|
| 10301 | 77,74 € |
| 10302 | 0,00 € |
| 16001 | 156,05 € |
| 16002 | 60,43 € |
| 16003 | 70,59 € |
| 19001 | 90,10 € |
| 19002 | 90,18 € |
| 20303 | 33,66 € |
| 25007 | 46,58 € |
| 28008 | 135,48 € |
| 28012 | 127,71 € |
| 28022 | 119,83 € |
| 28045 | 42,66 € |
| 28100 | 125,59 € |
| 28109 | 64,48 € |
| 28230 | 236,79 € |
| 28232 | 88,45 € |
| 28320 | 38,61 € |
| 45001 | 76,80 € |
| 45022 | 145,54 € |

Evidentemente la función Sum y otras que veremos aquí sólo son invocables en consultas de acumulados como esta.

Otras funciones de acumulados son: Avg, Min, Max, Count, StDev, First, Last

Documéntate en la web sobre ellas.

# 4. Consulta multitabla (mtbl1)

La ventaja de las bases de datos relacionales es que se pueden programar sentencias que utilicen la combinación de varias tablas de forma sencilla (para los casos comunes).

La cuestión es indicar en la clausula FROM cuales y en que forma hay que combinar tablas.

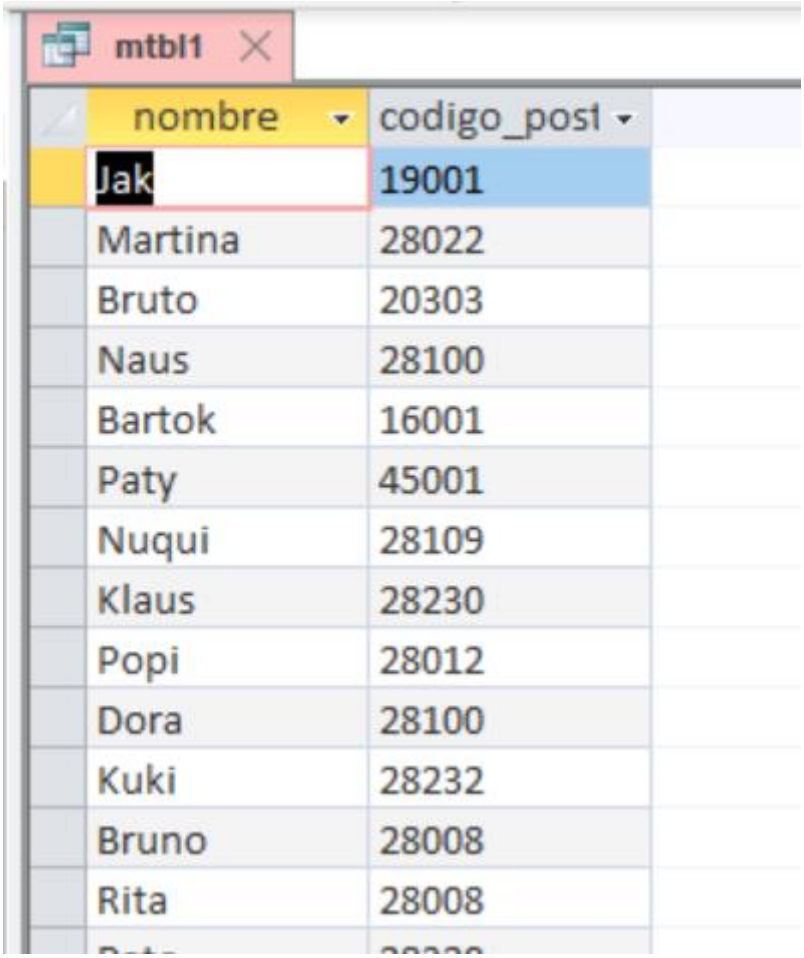
Por ejemplo, queremos obtener la lista de los nombres de los perros y su código postal. El código postal sólo se encuentra en la tabla de propietarios, pero existe una relación 1 a n entre ellos y sus mascotas.

```
SELECT Animales.nombre, Propietarios.codigo_postal
FROM Propietarios INNER JOIN Animales ON Propietarios.id = Animales.id_propietario;
```

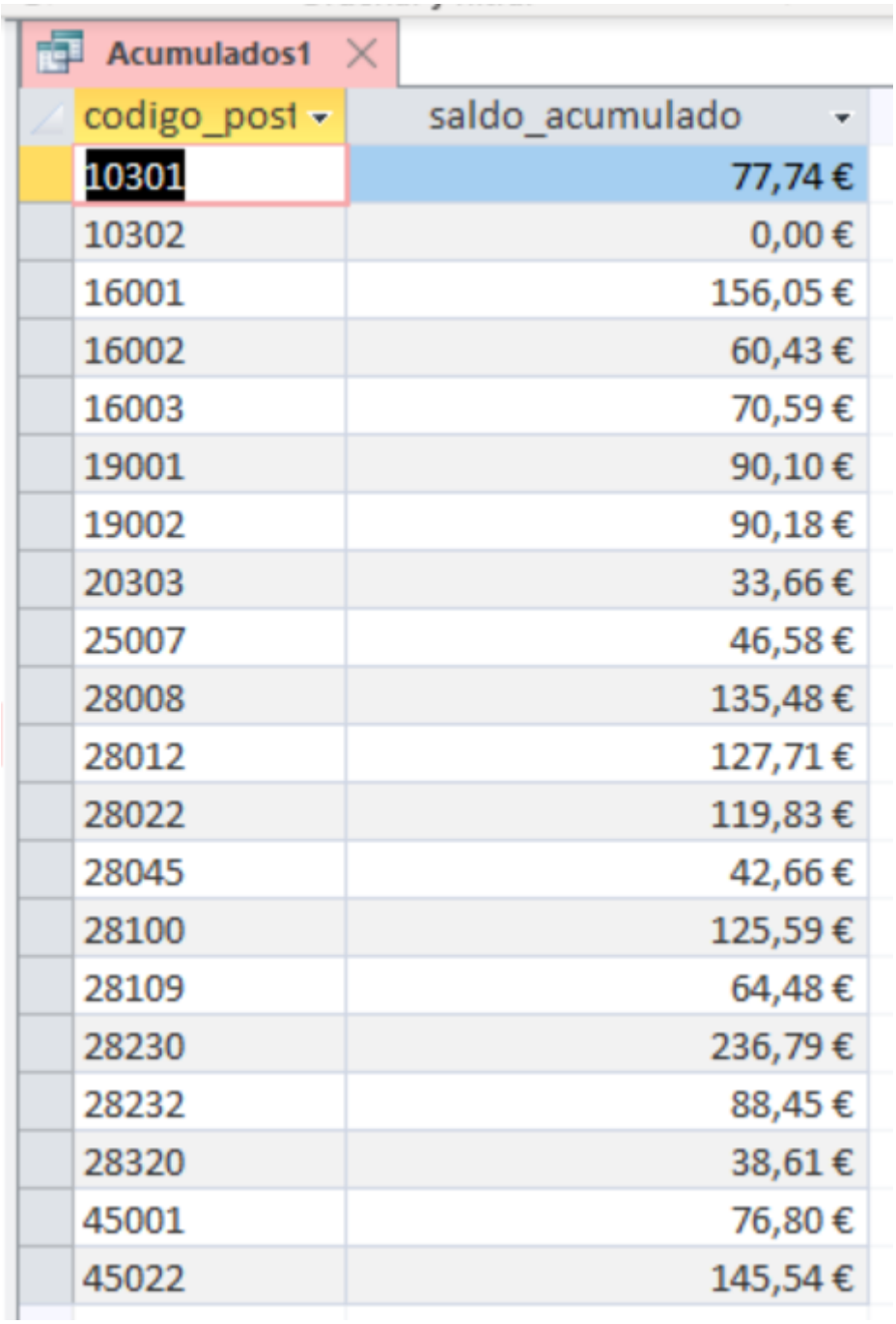
El operador INNER JOIN indica que dos tablas hay que combinar: Propietarios a la izquierda y Animales a la derecha. El modificador ON indica cual es la condición para combinar dos registros.

En definitiva, el INNER JOIN reúne cada uno de los registros de Propietarios con cada uno de los registros de animales, siempre que se cumpla la condición ON para crear nuevos registros virtuales y sobre ellos realizar el resto de la sentencia SELECT.

La sentencia antes escrita recupera el nombre del animal y el código postal del propietario (que en definitiva será el del animal) siempre y cuando el animal pertenezca al propietario.


mtbl1

| nombre | codigo_posl |
|---|---|
| Jak | 19001 |
| Martina | 28022 |
| Bruto | 20303 |
| Naus | 28100 |
| Bartok | 16001 |
| Paty | 45001 |
| Nuqui | 28109 |
| Klaus | 28230 |
| Popi | 28012 |
| Dora | 28100 |
| Kuki | 28232 |
| Bruno | 28008 |
| Rita | 28008 |



Acumulados1

| codigo_post | saldo_acumulado |
|---|---|
| 10301 | 77,74 € |
| 10302 | 0,00 € |
| 16001 | 156,05 € |
| 16002 | 60,43 € |
| 16003 | 70,59 € |
| 19001 | 90,10 € |
| 19002 | 90,18 € |
| 20303 | 33,66 € |
| 25007 | 46,58 € |
| 28008 | 135,48 € |
| 28012 | 127,71 € |
| 28022 | 119,83 € |
| 28045 | 42,66 € |
| 28100 | 125,59 € |
| 28109 | 64,48 € |
| 28230 | 236,79 € |
| 28232 | 88,45 € |
| 28320 | 38,61 € |
| 45001 | 76,80 € |
| 45022 | 145,54 € |

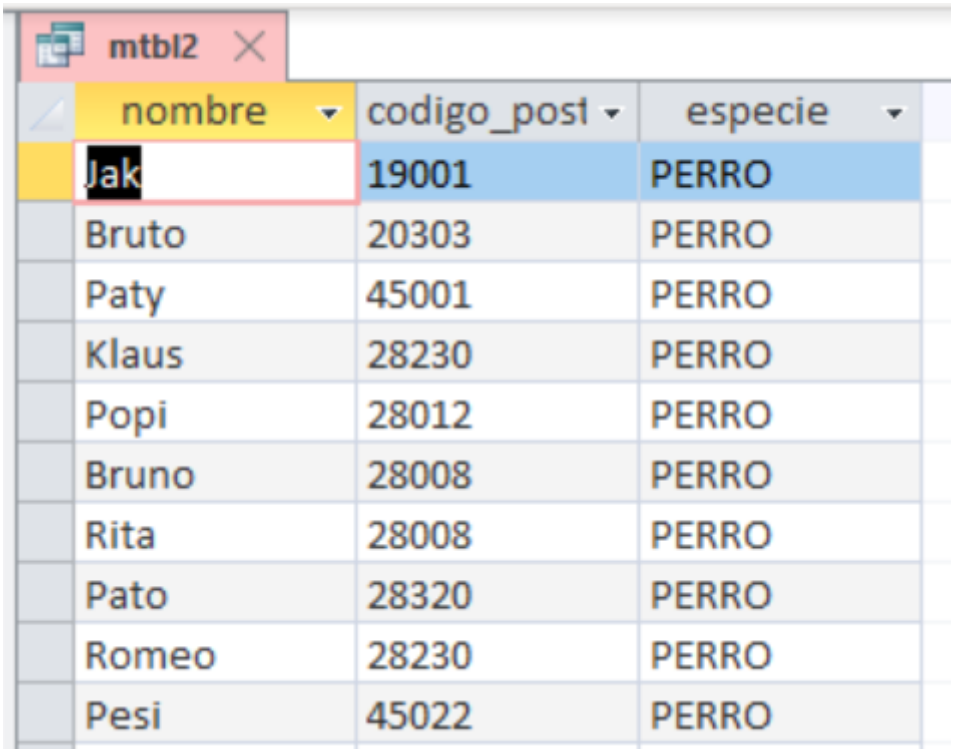
En la consulta ha sido imprescindible utilizar la forma Animales.nombre, ya que nombre es un título de campo repetido en las dos tablas que se utilizan.

Existen otros dos operadores similares a INNER JOIN; son LEFT JOIN y RIGHT JOIN, tienes que consultar en el web su utilidad, en que se diferencian y cuando son necesarios.

## 4.1. Algo más (mtbl2)

Si queremos que la consulta anterior se realice sólo sobre los perros, necesitamos incorporar la tabla razas que es la que tiene indicación de la especie, de la siguiente forma. SELECT Animales.nombre, Propietarios.codigo_postal, Razas.especie

```
FROM Razas
        INNER JOIN (Propietarios
                INNER JOIN Animales
                ON Propietarios.id = Animales.id_propietario)
        ON Razas.id = Animales.id_raza
        WHERE (((Razas.especie)="perro"))
```



mtbl2

| nombre | codigo_post | especie |
|---|---|---|
| Jak | 19001 | PERRO |
| Bruto | 20303 | PERRO |
| Paty | 45001 | PERRO |
| Klaus | 28230 | PERRO |
| Popi | 28012 | PERRO |
| Bruno | 28008 | PERRO |
| Rita | 28008 | PERRO |
| Pato | 28320 | PERRO |
| Romeo | 28230 | PERRO |
| Pesi | 45022 | PERRO |

En este caso se ha anidado un INNER JOIN dentro de otro, el interno para sustituir a lo que antes era una tabla. Intenta comprender la estructura y el funcionamiento de la consulta.

# 5. Las demás sentencias

Documéntate sobre las siguientes sentencias SQL: INSERT, UPDATE, DELETE; uso y sintáxis.

Estás tres junto con SELECT son las básicas de administración de bases de datos y prácticamente comunes en todos los dialectos, pero existen más. Aunque están fuera del temario, ya que prácticamente no tienen utilidad en Access, deberías ampliar conocimiento sobre cómo construir y eliminar bases de datos, y como construir y mantener tablas en motores como MySQL o SQLServer, porque no todo el mundo es OFFICE.