DML – Consultas SQL

DML - Consultas SQL

El DML es el conjunto de sentencias SQL que nos permite la consulta, inserción y manipulación de los datos almacenados en nuestra base de datos.

En este resumen enumeraremos y practicaremos las principales consultas así como sus principales variaciones.

Para ejemplificar las siguientes consultas, vamos a imaginar que tenemos la tabla donaciones con los siguientes datos:

id	DNI	Nombre	Población	Donativo	Campaña
1	1111A	María	Benidorm	11	Donar a Ucrania
2	2222B	Juan	Altea	12	Crisis en Ucrania
3	3333C	Ana	Benidorm	5	Donar a España
4	4444D	Marcos	Villajoyosa	7	Crisis en Ucrania
5	5555E	Eva	Altea	21	Hambre en Áfica
6	6666F	Patricia	Benidorm	9	Donar a Ucrania
7	7777G	Patricio	Benidorm	7	Hambre en África

Consultas básicas

Una consulta básica para obtener datos de las BBDD sigue la siguiente estructura:

```
SELECT campo1
FROM tabla1;
```

De este modo estaríamos mostrando todos los valores almacenados en la columna campo1 de la tabla tabla1. Así, si deseáramos conocer el nombre de todas aquellas personas que han realizado un donativo, ejecutaríamos la sentencia:

```
SELECT nombre
FROM donaciones;
```

	nombre character varying (20)
1	María
2	Juan
3	Ana
4	Marcos
5	Eva
6	Patricia
7	Patricio

En el caso de querer consultar más de una columna, sólo debemos enumerarlas separando cada nombre de columna con una coma:

```
SELECT nombre, donativo
FROM donaciones;
```

	nombre character varying (20)	donativo numeric (6,2)
1	María	11.00
2	Juan	12.00
3	Ana	5.00
4	Marcos	7.00
5	Eva	21.00
6	Patricia	9.00
7	Patricio	7.00

Pero si deseamos mostrar todos los datos de una tabla, podemos emplear el operador * para evitar tener que enumerar todas las columnas:

```
SELECT *
FROM donaciones;
```

	id [PK] numeric (5)	dni character varying (5)	nombre character varying (20)	poblacion character varying (20)	donativo numeric (6,2)	campaña character varying (100)
1	1	1111A	María	Benidorm	11.00	Donar a Ucrania
2	2	2222B	Juan	Altea	12.00	Crisis en Ucrania
3	3	3333C	Ana	Benidorm	5.00	Donar a España
4	4	4444D	Marcos	Villajoyosa	7.00	Crisis en Ucrania
5	5	5555E	Eva	Altea	21.00	Hambre en África
6	6	6666F	Patricia	Benidorm	9.00	Donar a Ucrania
7	7	7777G	Patricio	Benidorm	7.00	Hambre en África

El orden en que se muestran los datos, en principio no viene establecido por ningún campo, si se desea ordenar en función de un campo en concreto añadiremos la cláusula `ORDER BY` indicando el campo a ordenar y el parámetro `ASC` si deseamos orden ascendente (de menor a mayor o alfabético) o `DESC` (de mayor a menor o alfabético invertido). Así si deseamos listar las donaciones ordenadas en orden de mayor donativo a menor, la sentencia sería:

```
SELECT *
FROM donaciones
ORDER BY donativo DESC;
```

	id [PK] numeric (5)	dni character varying (5)	nombre character varying (20)	poblacion character varying (20)	donativo numeric (6,2)	campaña character varying (100)
1	5	5555E	Eva	Altea	21.00	Hambre en África
2	2	2222B	Juan	Altea	12.00	Crisis en Ucrania
3	1	1111A	María	Benidorm	11.00	Donar a Ucrania
4	6	6666F	Patricia	Benidorm	9.00	Donar a Ucrania
5	4	4444D	Marcos	Villajoyosa	7.00	Crisis en Ucrania
6	7	7777G	Patricio	Benidorm	7.00	Hambre en África
7	3	3333C	Ana	Benidorm	5.00	Donar a España

Consultas con varias tablas

Hasta ahora hemos visto consultas sencillas, que implican una única tabla, sin embargo, la realidad suele ser más compleja y las consultas suelen implicar 2 o más tablas.

Como sabemos por el punto anterior, las tablas a emplear se indican en la cláusula `FROM` , sin embargo con esto no es suficiente, ya que hay que indicar de algún modo como se relacionan las tablas entre sí; para ello haremos uso de la cláusula `WHERE` la cual también emplearemos (como veremos más tarde) para filtrar los registros a mostrar.

Para ejemplificar, a la tabla anterior vamos a añadirle una nueva donde relacionamos cada campaña con una ONG:

nombre	Campaña
Save the children	Donar a Ucrania
Oxfam Intermon	Crisis en Ucrania
Save the children	Donar a España
Oxfam Intermon	Hambre en África

Si deseamos mostrar una lista con todas las persona que han realizado una donación y conocer además de la campaña con qué ONG han participado, la consulta sería:

```
SELECT donaciones.nombre, donaciones.campaña, ong.nombre, donativo
FROM donaciones, ong
WHERE donaciones.campaña = ong.campaña;
```

	nombre character varying (20)	campaña character varying (100)	nombre character varying (20)	donativo numeric (6,2)
1	María	Donar a Ucrania	Save the children	11.00
2	Juan	Crisis en Ucrania	Oxfam Intermon	12.00
3	Ana	Donar a España	Save the children	5.00
4	Marcos	Crisis en Ucrania	Oxfam Intermon	7.00
5	Eva	Hambre en África	Oxfam Intermon	21.00
6	Patricia	Donar a Ucrania	Save the children	9.00
7	Patricio	Hambre en África	Oxfam Intermon	7.00

Si observamos la consulta anterior, hay varios campos que se repiten en ambas tablas, con lo que si hacemos referencia a ellos en la consulta, deberemos especificar a cual de los 2 campos nos referimos (no es lo mismo el campo *nombre* en la tabla *donaciones* que el campo *nombre* en la tabla *ong)*, por ello anteponemos al campo el nombre de la tabla seguido de un punto. Podemos reducir el tamaño de la consulta si en lugar del nombre de la tabla, creamos un *alias* para cada tabla y nos referimos a ella mediante dicho alias; así la consulta anterior podríamos escribirla del siguiente modo:

```
SELECT d.nombre, d.campaña, o.nombre, donativo
FROM donaciones d, ong o
WHERE d.campaña = o.campaña;
```

Observa que en ninguna de las 2 consultas, hemos especificado *donativo* de qué tabla procede, ya que el campo *donativo* sólo aparece en la tabla de *donaciones* y por tanto podemos referirnos a él de forma inequívoca.

Como hemos dicho al principio de este apartado, la cláusula `WHERE` se emplea para indicar la relación entre las tablas consultadas, pero también para establecer filtros a nivel de registro; así si deseamos modificar la consulta anterior y ver sólo las donaciones realizadas a una ONG en concreto, podríamos hacerlo mediante la siguiente consulta:

```
SELECT d.nombre, d.campaña, o.nombre, donativo
FROM donaciones d, ong o
WHERE d.campaña = o.campaña
 AND o.nombre = 'Save the children';
```

	nombre character varying (20)	campaña character varying (100)	nombre character varying (20)	donativo numeric (6,2)
1	María	Donar a Ucrania	Save the children	11.00
2	Ana	Donar a España	Save the children	5.00
3	Patricia	Donar a Ucrania	Save the children	9.00

Como hemos visto en la consulta anterior, podemos añadir distintos criterios de filtrado, para ello sólo debemos añadirlos a la consulta mediante los conectores `AND` y `OR` según las necesidades de cada consulta.

Además de los operadores de comparación y de los operadores lógicos presentes en la mayoría de lenguajes de programación:

`=` igualdad
`!=` diferencia
`>` mayor que
`<` menor que
`AND` y
`OR` o
`NOT` no
`IS NULL/NOT NULL` para comprobar si un campo es o no nulo (no se emplea el comparador de igualdad o diferencia.

También encontramos operadores propios de SQL que nos permitirán construir consultas de manera más breve y sencilla:

`LIKE` Comparación con cadenas (se suele emplear junto a los comodines `%` y `_`)
`%` : 0, 1 o muchos caracteres
`_` : 1 único carácter (no puede ser 0, no pueden ser varios)
`BETWEEN` Especificar rangos
`IN` Listado de valores permitidos

Funciones con cadenas de caracteres y fechas

Las funciones con cadenas de caracteres y fechas, pueden emplearse tanto junto al `SELECT` como para establecer condiciones en la cláusula `WHERE`. En este apartado sólo enumeraremos algunas de ellas (las de uso más común) y cabe mencionar que pueden cambiar entre sistemas gestores de bases de datos.

`LENGTH (<campo_de texto>):` Devuelve un entero indicando la longitud del campo pasado como parámetro.

`UPPER (<campo_de texto>):` Convierte el texto a MAYÚSCULAS.
`LOWER (<campo_de texto>):` Convierte el texto a minúsculas.

`TRIM (<campo_de texto>, [<caracter>]):` Elimina el carácter indicado al principio y al final de la cadena pasada como parámetro. Si no se indica ningún parámetro, tratará de eliminar espacios en blanco. Existen dos variantes de esta función `LTRIM` y

`RTRIM`, las cuales eliminan respectivamente el carácter sólo por delante o sólo por detrás del texto indicado. Ejemplo:
`TRIM ('----Hola----', '-')` El resultado será `Hola`
`LTRIM ('----Hola----', '-')` El resultado será `Hola----`
`RTRIM ('----Hola----', '-')` El resultado será `----Hola`

`RPAD (<campo_de_texto>, <entero>, <texto>)` o su otra variante `LPAD (<campo_de texto>, <entero>, <texto>):` Podriamos decir que se tratan de la funciones opuestas a RTRIM y LTRIM respectivamente, es decir, dado un campo de texto `<campo_de_texto>`, lo rellenan (por la izquierda o derecha, según la función) con el texto `<texto>` hasta tener una longitud de `<entero>` caracteres; si el texto ya tiene una longitud igual o superior a la indicada, no hace nada.

`substring(<cadena_de_texto> [from int] [for int])`: devuelve una subcadena de la cadena pasada como parámetro, que vas desde la posición `[from int]` (o inicial si no se indica este parámetro), hasta la posición `[for int]` (o final si no se indica dicho parámetro).

Operaciones de agregación (con números)

En ocasiones deseamos obtener el resultado de ciertas operaciones sobre un campo numérico. Para ello SQL incorpora ciertas funciones de uso común, que nos facilitará las consultas de este tipo.

Así si deseamos sumar todos los valores de un campo, tenemos la función `SUM`, para calcular la media `AVG`, para obtener el máximo y el mínimo contamos con `MAX` y `MIN` respectivamente, y si simplemente deseamos contar registros tenemos `COUNT`.

Siguiendo con el ejemplo de las donaciones, si deseamos saber cuantas donaciones se han realizado, la consulta sería:

```
SELECT COUNT(*)
FROM donaciones;
```

	count bigint
1	7

En cambio si deseamos saber el dinero recaudado, la consulta quedaría así:

```
SELECT SUM(donativo)
FROM donaciones;
```

	sum numeric
1	72.00

Si deseamos calcular el dinero recaudado en una campaña, la consulta sería:

```
SELECT SUM(donativo)
FROM donaciones
WHERE campaña = 'Hambre en África';
```

	sum numeric
1	28.00

Y para consultar el importe máximo donado a una ONG en concreto:

```
SELECT MAX(donativo)
FROM donaciones d, ong o
WHERE d.compaña = o.campaña
      AND o.nombre = 'Oxfam Intermon';
```

	max numeric
1	21.00

En muchas ocasiones, a funciones de agregación van acompañadas de la clausula `GROUP BY` que permitirá hacer agrupaciones para mostrar resultados por grupos.

Para entender mejor el funcionamiento de `GROUP BY` , imaginemos que queremos mostrar un listado donde calculemos el importe recaudado **no para una campaña en concreto**, sino **para cada una de las campañas existentes en la base de datos**. Esto lo podríamos conseguir mediante el uso de `GROUP BY.`

```
SELECT campaña, SUM(donativo)
FROM donaciones
GROUP BY campaña;
```

	campaña character varying (100)	sum numeric
1	Donar a Ucrania	20.00
2	Hambre en África	28.00
3	Donar a España	5.00
4	Crisis en Ucrania	19.00

Como podemos ver en el ejemplo anterior, agrupa las donaciones por campaña y realiza la suma para cada una de ellas.

Unida a la cláusula `GROUP BY` solemos encontrar le sentencia `HAVING`. Dicha sentencia permite el filtrado de datos agrupados; podríamos decir que es algo así como un `WHERE` pero sobre las operaciones de agregación. Así, si sobre el ejemplo anterior, deseáramos conocer aquellas campañas que han recaudado 20 o más euros, añadiríamos la cláusula `HAVING` indicando la correspondiente condición:

```
SELECT campaña, SUM(donativo)
FROM donaciones
GROUP BY campaña
HAVING SUM(donativo) >= 20;
```

	campaña character varying (100)	sum numeric
1	Donar a Ucrania	20.00
2	Hambre en África	28.00

Para comprender la diferencia entre WHERE y HAVING, debemos pensar que el WHERE sirve para aplicar filtros a nivel de registros (filas de cada tabla) en cambio el HAVING se emplea para filtrar sobre operaciones de agregación. Ambas sentencias pueden aparecer en una misma sentencia; por ejemplo, imaginemos que deseamos visualizar, agrupadas por campaña, todas las donaciones realizadas desde Benidorm y que hayan sumado más de 5 €:

```
SELECT campaña, SUM(donativo)
```

```
FROM donaciones
WHERE poblacion = 'Benidorm'
GROUP BY campaña
HAVING SUM(donativo) >5;
```

	campaña character varying (100)	sum numeric
1	Donar a Ucrania	20.00
2	Hambre en África	7.00

También cabe mencionar, que unido a las operaciones de agregación, suelen emplearse alias para renombrar estas columnas de resultados, facilitando el acceso a dichos datos desde aplicaciones o servicios externos:

```
SELECT campaña, SUM(donativo) total_donativos
FROM donaciones
WHERE poblacion = 'Benidorm'
GROUP BY campaña
HAVING SUM(donativo) >5;
```

	campaña character varying (100)	total_donativos numeric
1	Donar a Ucrania	20.00
2	Hambre en África	7.00

Subconsultas

En algunas situaciones, necesitaremos anidar una (o varias SELECT) como condición dentro de otra SELECT; de este modo podremos abordar consultas más complejas. Estas subconsultas suelen anidarse mediante comparaciones y las expresiones ANY, ALL, SOME , IN, NOT IN , EXISTS o NOT EXISTS.

Así, si deseáramos conocer quién es la persona que ha realizado el mayor donativo, podríamos hacerlo mediante una consulta del tipo:

```
SELECT nombre
FROM donaciones
WHERE donativo = (SELECT MAX(donativo)
                  FROM donaciones);
```

	nombre character varying (20)
1	Eva

Si deseamos conocer qué personas han realizado donaciones por encima de la media, la consulta a emplear sería:

```
SELECT nombre, donativo
FROM donaciones
WHERE donativo > (SELECT AVG(donativo)
                  FROM donaciones);
```

	nombre character varying (20)	donativo numeric (6,2)
1	María	11.00
2	Juan	12.00
3	Eva	21.00

En los 2 casos anteriores, la subconsulta sirve para calcular un valor y anidarlo a la consulta principal mediante una operación de comparación; pero habrá situaciones en las que la subconsulta nos devuelva un listado de valores, en tal caso no podremos aplicar directamente la operación de comparación, sino que deberemos añadir una de las expresiones vistas anteriormente (*ANY, ALL, SOME , IN, NOT IN , EXISTS o NOT EXISTS*) según las necesidades del momento.

Imaginemos que deseamos conocer el nombre y la donación realizada por aquellas personas cuyo donativo es superior a la media de donaciones de todas las campañas; en este caso emplearíamos la subconsulta para calcular la media de donativos realizados en cada campaña y la consulta principal compararíamos el valor de cada donativo con el listado de valores obtenidos anteriormente:

```
SELECT nombre, donativo
FROM donaciones
WHERE donativo > ALL(SELECT AVG(donativo)
                     FROM donaciones
                     GROUP BY campaña);
```

	nombre character varying (20)	donativo numeric (6,2)
1	Eva	21.00

El predicado `ALL` se utiliza para recuperar únicamente aquellos registros de la consulta principal que cumplen la condición establecida con todos los registros recuperados en la subconsulta. En cambio, el predicado `ANY`, permite recuperar los registros que cumplen la condición de comparación con respecto a alguno de los registros recuperados en la subconsulta (**no respecto a todos**).

Si reemplazamos `ALL` por `ANY` en el ejemplo anterior, la consulta no sólo mostrará el nombre de aquellas personas cuyo donativo supere la media de donaciones en todas las campañas, sino que mostrará aquellos cuyo donativo sea superior a la media de donaciones de cualquier campaña.

```
SELECT nombre, donativo
FROM donaciones
WHERE donativo > ANY(SELECT AVG(donativo)
                     FROM donaciones
                     GROUP BY campaña);
```

	nombre character varying (20)	donativo numeric (6,2)
1	María	11.00
2	Juan	12.00
3	Marcos	7.00
4	Patricia	9.00
5	Eva	21.00
6	Patricio	7.00

El predicado SOME actúa de igual modo que el ANY. En cambio la cláusula `EXISTS` nos permite generar consultas con condiciones complejas del tipo *lista el nombre y donativo de todas aquellas personas cuyo donativo sea el doble que el donativo realizado por otra persona para la misma campaña*:

```
SELECT nombre, donativo
FROM donaciones d1
WHERE EXISTS(SELECT d2.nombre, d2.donativo
             FROM donaciones d2
             WHERE d1.campaña =d2.campaña
                   AND d1.donativo >= 2*d2.donativo);
```

	nombre character varying (20)	donativo numeric (6,2)
1	Eva	21.00

Consultas de conjuntos

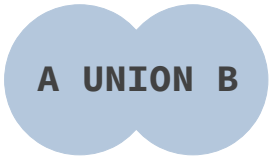
En algunas situaciones, lo que necesitamos es realizar la suma, resta o intersección de 2 consultas *independientes*; esto lo podremos hacer mediante las consultas de conjuntos.

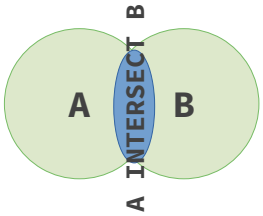
Este tipo de consultas se construyen mediante los predicados `UNION`, `UNION ALL`, `INSTERSECT` o `EXCEPT` (o `MINUS`). **Cabe mencionar que para unir las 2 consultas de conjuntos, ambas deben devolver el mismo número y tipo de campos, aunque pueden proceder de diferentes tablas.**

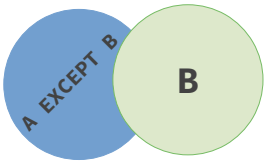
`UNION`	Devuelve como resultado la unión (o suma) de los resultados obtenidos por ambas consultas, sin duplicados.
`UNION ALL`	Su funcionamiento es el mismo que el anterior, pero devolviendo duplicados
`INTERSECT`	Devuelve la intersección de los resultados obtenidos por las dos consultas, o dicho de otra manera, aquellos resultados que aparecen en ambas consultas.
`EXCEPT` (o `MINUS`)	Devuelven los registros de la primera consulta que no aparecen en la segunda, es decir a los resultados de la primera le resta los coincidentes en la segunda consulta.

Para visualizarlo de manera gráfica, imaginemos que tenemos 2 consultas A y B cuyos resultados representamos con las siguientes circunferencias:

El resultado de las distintas operaciones, devolvería los registros marcados mediante el color azul:







Para ver un ejemplo de uso, imaginemos que queremos listar todas aquellas campañas que no hayan recibido ningún donativo de Benidorm; lo primero que se nos ocurriría es una consulta del tipo

```
SELECT *
FROM donaciones
WHERE poblacion != 'Benidorm';
```

	campaña character varying (100)
1	Crisis en Ucrania
2	Crisis en Ucrania
3	Hambre en África

Sin embargo, con una rápida revisión de los datos almacenados en las tablas, observaremos que la campaña Hambre en África recibe un donativo de Patricio desde Benidorm. Por lo tanto esta consulta no es la correcta, en realidad está mostrando un listado de las campañas procedente de los registros donde población es distinto de Benidorm, o dicho de otro modo, muestra las campañas donde ha participado gente

que no es de Benidorm, pero no excluye las campañas donde sí han participado benidormenses. En este caso, para obtener el resultado deseado, podríamos hacerlo mediante una resta de conjuntos, en primer lugar listamos todas las campañas y a continuación le restamos las campañas con donativos de Benidorm:

```
SELECT campaña
FROM donaciones

EXCEPT

SELECT campaña
FROM donaciones
WHERE poblacion = 'Benidorm';
```

	campaña character varying (100)
1	Crisis en Ucrania

Consultas con combinación interna (JOIN)

Como hemos visto, es bastante frecuente tener que recuperar combinaciones de datos procedentes de tablas relacionadas. Hasta ahora hemos visto cómo realizar esto indicando la relación de las tablas en el predicado WHERE; sin embargo, hay situaciones en las que se desea mostrar no sólo los datos en los que se cumple la condición de relación establecido en la cláusula WHERE, sino también aquellos datos que no se relacionan con otros registros (suele tratarse de registros cuyo campo clave ajena se encuentra a NULL); para estos casos se creó el predicado JOIN, el cual puede facilitarnos mucho dicha tarea.

Hay diferentes tipos de predicado JOIN, pero en este tema nos centraremos en INNER JOIN, LEFT JOIN, RIGHT JOIN y FULL OUTER JOIN.

Para entender mejor el funcionamiento de estas sentencias, vamos a añadir un nuevo registro a nuestras 2 tablas; en la tabla de donaciones añadiremos el registro:

id	DNI	Nombre	Población	Donativo	Campaña
9	9999Z	Aurelio	Alfaz del pi		

Y en la tabla de ONGs insertamos la campaña:

nombre	Campaña
Save the children	Los últimos 100

Como vemos, tenemos un donante que aún no ha donado nada a ninguna campaña, y tenemos una nueva campaña que no tiene asociado ningún donante.

Volviendo a la teoría, tenemos que el predicado [INNER] JOIN (recuerda que la cláusula INNER puede ser omitida) funciona de manera similar a establecer la relación de registros mediante una condición en la cláusula WHERE, y por tanto sólo mostrará aquellos registros en los que sí existe la relación:

```
SELECT d.nombre, d.donativo, o.nombre, o.campaña
FROM donaciones d INNER JOIN ong o ON o.campaña = d.campaña;
```

En este caso sólo obtendremos el nombre de donantes que sí han realizado donaciones y las campañas a las que han donado:

	nombre character varying (20)	donativo numeric (6,2)	nombre character varying (20)	campaña character varying (100)
1	María	11.00	Save the children	Donar a Ucrania
2	Juan	12.00	Oxfam Intermon	Crisis en Ucrania
3	Ana	5.00	Save the children	Donar a España
4	Marcos	7.00	Oxfam Intermon	Crisis en Ucrania
5	Patricia	9.00	Save the children	Donar a Ucrania
6	Eva	21.00	Oxfam Intermon	Hambre en África
7	Patricio	7.00	Oxfam Intermon	Hambre en África

Si cambiamos la cláusula `INNER` por `LEFT`, da *prioridad* a los registros de donaciones, es decir mostrará los donantes y las ONG donde han donado, pero también listarán aquellos donantes que aún no han hecho el donativo (en el resultado, observa el registro 8):

```
SELECT d.nombre, d.donativo, o.nombre, o.campaña
FROM donaciones d LEFT JOIN ong o ON o.campaña = d.campaña;
```

	nombre character varying (20)	donativo numeric (6,2)	nombre character varying (20)	campaña character varying (100)
1	María	11.00	Save the children	Donar a Ucrania
2	Juan	12.00	Oxfam Intermon	Crisis en Ucrania
3	Ana	5.00	Save the children	Donar a España
4	Marcos	7.00	Oxfam Intermon	Crisis en Ucrania
5	Patricia	9.00	Save the children	Donar a Ucrania
6	Eva	21.00	Oxfam Intermon	Hambre en África
7	Patricio	7.00	Oxfam Intermon	Hambre en África
8	Aurelio	[null]	[null]	[null]

En cambio, si indicamos `RIGHT JOIN` da prioridad a mostrar los datos de la segunda tabla, en este caso las donaciones; dicho de otro modo, listará todas las donaciones realizadas y las campañas donde se ha donado, pero también visualizará aquellas campañas que no han recibido ningún donativo todavía (observa el registro 8):

```
SELECT d.nombre, d.donativo, o.nombre, o.campaña
FROM donaciones d RIGHT JOIN ong o ON o.campaña = d.campaña;
```

	nombre character varying (20)	donativo numeric (6,2)	nombre character varying (20)	campaña character varying (100)
1	María	11.00	Save the children	Donar a Ucrania
2	Juan	12.00	Oxfam Intermon	Crisis en Ucrania
3	Ana	5.00	Save the children	Donar a España
4	Marcos	7.00	Oxfam Intermon	Crisis en Ucrania
5	Patricia	9.00	Save the children	Donar a Ucrania
6	Eva	21.00	Oxfam Intermon	Hambre en África
7	Patricio	7.00	Oxfam Intermon	Hambre en África
8	[null]	[null]	Save the children	Los últimos 100

Finalmente, encontramos el predicado `FULL [OUTER] JOIN` el cual muestra todos los registros tanto los relacionados en ambas tablas como los donantes que no han donado, o las campañas sin donativos:

```
SELECT d.nombre, d.donativo, o.nombre, o.campaña
FROM donaciones d FULL OUTER JOIN ong o ON o.campaña = d.campaña;
```

	nombre character varying (20)	donativo numeric (6,2)	nombre character varying (20)	campaña character varying (100)
1	María	11.00	Save the children	Donar a Ucrania
2	Juan	12.00	Oxfam Intermon	Crisis en Ucrania
3	Ana	5.00	Save the children	Donar a España
4	Marcos	7.00	Oxfam Intermon	Crisis en Ucrania
5	Patricia	9.00	Save the children	Donar a Ucrania
6	Eva	21.00	Oxfam Intermon	Hambre en África
7	Patricio	7.00	Oxfam Intermon	Hambre en África
8	Aurelio	[null]	[null]	[null]
9	[null]	[null]	Save the children	Los últimos 100

Limitar los resultados en las consultas

Para finalizar con este repaso a las consultas SQL, nos queda mencionar el predicado LIMIT, el cual limita el número de registros devuelto por la consulta. Así si sólo deseamos ver los 3 mayores donativos realizados, la sentencia sería:

```
SELECT *
FROM donaciones
ORDER BY donativo DESC
LIMIT 3;
```

	id [PK] numeric (5)	dni character varying (5)	nombre character varying (20)	poblacion character varying (20)	donativo numeric (6,2)	campaña character varying (100)
1	5	5555E	Eva	Altea	21.00	Hambre en África
2	2	2222B	Juan	Altea	12.00	Crisis en Ucrania
3	1	1111A	María	Benidorm	11.00	Donar a Ucrania

Pero cuidado porque esta instrucción simplemente limita los resultados mostrados sin tener en cuenta los valores de ninguna condición, por tanto si tenemos un cuarto donativo con el mismo importe que el tercero, no se visualizaría en esta lista.

Este ejemplo se visualiza mejor con una consulta del tipo *Queremos saber quien ha realizado el mayor donativo y de cuánto es dicha cantidad.* Pero primero debemos

introducir un nuevo registro en la tabla de donaciones con los siguientes valores:

id	DNI	Nombre	Población	Donativo	Campaña
10	1010W	Segundina	Orxeta	21	Donar a España

Volviendo a la consulta planteada en el párrafo anterior, podemos pensar que con la siguiente sentencia lo tendríamos resuelto:

```
SELECT nombre, donativo, campaña
FROM donaciones
ORDER BY donativo DESC
LIMIT 1;
```

	nombre character varying (20)	donativo numeric (6,2)	campaña character varying (100)
1	Eva	21.00	Hambre en África

Sin embargo, con el nuevo registro añadido anteriormente, Eva no es la única donante que ha donado 21€ por lo tanto esta consulta no no es del todo válida. En este caso, deberíamos hacer uso de una subconsulta:

```
SELECT nombre, donativo, campaña
FROM donaciones
WHERE donativo = (SELECT MAX(donativo)
                  FROM donaciones);
```

	nombre character varying (20)	donativo numeric (6,2)	campaña character varying (100)
1	Eva	21.00	Hambre en África
2	Segundina	21.00	Donar a España