4. SQL / DQL. Consultas avanzadas

Vamos a ver cómo podemos unir las tablas para crear consultas avanzadas que unan los valores de todas ellas y que estén relacionadas por las claves principales o claves ajenas de estas.

A. Consultas multitabla o de unión

La potencia del modelo relacional aparece cuando realizamos consultas de varias tablas, es decir, que podemos unir los valores de todas las tablas fusionándose en la misma consulta por campos equivalentes que contengan los mismos valores y tipos de datos compatibles.

IMPORTANTE

Existen dos tipos de uniones: las externas y las internas. Las externas sirven para unir tablas que no están relacionadas, pero tienen los mismos campos con los mismos tipos, pudiendo coger valores de cada una de ella para conseguir una consulta con valores de ambas. Para ello utilizaremos `UNION`, si no quisiéramos valores repetidos, o `UNION ALL`, si los quisiéramos ver todos. Y la instrucción a ejecutar sería `SELECT * FROM tabla1 WHERE condicion1 UNION [ALL] SELECT * FROM tabla2 WHERE condicion2`, mostrándonos todos los valores que cumplieran cada condición en cada tabla sin estar relacionadas. Esto suele pasar cuando una base de datos no está correctamente construida o normalizada. A nosotros nos van a interesar las uniones internas que funcionan con claves primarias y claves ajenas.

Si sabemos que en la tabla SALA tenemos un campo idCine que nos indica a qué cine pertenece esa sala, y en la tabla CINE de nuestra base de datos MCGRAWCINES tenemos sus campos, podríamos pensar en unir ambos y mostrarlos todos a la vez. Por ejemplo, con la siguiente consulta.

```
SELECT sala.idCine, cine.nombreCine, cine.poblaciónCine,
                sala.idSala, sala.butacasSala FROM cine, sala;
```

¿Qué pensáis? ¿Habrá funcionado correctamente esta consulta? La respuesta es NO. Hay un problema. Trata de ejecutarla y mira qué pasa. Para solucionarlo debemos tener ambos campos claves; en este ejemplo, el campo idCine de ambas tablas. ¿Cómo lo podemos hacer?

```
SELECT cine.idCine AS "cine-cine", cine.nombreCine,
cine.poblaciónCine,
        sala.idCine AS "cine-sala", sala.idSala, sala.butacasSala
FROM cine, sala;
```

Aquí parecería que ya está haciéndose bien, pero vemos que el id del cine que toma de la tabla de cine y el que toma de la tabla de la sala no coinciden. Ese es el problema: esta última consulta combina todas las salas con todos los cines, sea o no la sala de ese cine, por lo que aún no nos sirve. Vamos a ver cómo lo podemos solucionar. La solución es poner como condición que solo se empareje cada sala con el cine correspondiente. Hacemos esto con un `WHERE cine.idCine = sala.idCine`. Siendo la consulta correcta la siguiente:

```
SELECT cine.idCine AS "Cine-cine", sala.idCine AS "cine-sala",
cine.nombreCine, cine.poblaciónCine, sala.idSala, sala.butacasSala
FROM cine, sala WHERE cine.idCine = sala.idCine;
```

Y el resultado que obtendremos será este si no hemos dado de alta más valores en las tablas originales.

Cine-cine	cine-sala	nombreCine	poblaciónCine	idSala	butacasSala
1	1	Colón	Alzira	1	100
1	1	Colón	Alzira	2	120
1	1	Colón	Alzira	3	111
2	2	Cervantes	Cullera	1	188

Fig. 6.4. Resultado de la consulta multitabla relacionada por claves.

De esta manera hemos conseguido unir ambas tablas, pero la manera correcta es utilizando las uniones internas (`LEFT JOIN, INNER JOIN y RIGHT JOIN`). Para ello empezamos ampliando la sintaxis del `SELECT` que conocemos hasta ahora añadiendo la cláusula `ON` después del `FROM`. En la cláusula `FROM` indicaremos qué tipo de unión interna queremos realizar, y en el `ON` indicaremos los campos que son equivalentes, como hemos hecho en la consulta anterior. Por tanto, la cláusula `JOIN` es equivalente a hacer un producto cartesiano que en SQL se puede obtener con la cláusula `CROSS`, pero añadiendo un filtro que es la unión por los campos equivalentes de cada tabla.

Fig. 6.5. Orden en que se han de escribir las cláusulas dentro de la orden `SELECT`.

Sintaxis completa

```
SELECT [ {ALL | DISTINCT} ]
    <nombre_campo> [ , <nombre_campo> [ , ... ] ]
FROM tabla_A  {LEFT | INNER | RIGHT} JOIN tabla_B
ON tabla_A.idNombreCampo =  tabla_B.idNombreCampo
[WHERE <condición> [ {AND | OR} <condición> [ , ...] ]
[GROUP BY <nombre_campo> [ , <nombre_campo> [ , ...] ]
[HAVING <condición> [ {AND | OR} <condición> [ , ...] ]
[ORDER BY <condición> [ {AND | OR} <condición> [ , ...] ] [DESC]
```

Tabla 6.22. Sintaxis completa de la orden `SELECT` con uniones internas de SQL.

Dentro de estos tres tipos de unión de composición (`join`) podemos llegar a conseguir obtener hasta cinco conjuntos de datos diferentes. Lo más habitual o natural es utilizar la composición interna de equivalencia `INNER JOIN` o su subtipo o especialización de este llamado, `NATURAL JOIN`, combinando solo por los campos que son iguales en ambas tablas.

Pero a veces podemos querer saber qué pasa con el resto de los datos que tenemos. Si utilizamos la `LEFT` o la `RIGHT` también admite poner la cláusula [`OUTER`] a continuación, ya que estos se

denominan composiciones externas junto a la opción completa de `FULL [OUTER] JOIN`, que permite obtener otros dos conjuntos diferentes.

Tanto utilizando la opción a izquierdas o a derechas, obtendremos aquellos campos que no tengan su entrada equivalente en la otra tabla, que se rellenará con valores `NULL` en los campos que no tenemos valores. Y justamente filtrando por los valores nulos podemos llegar a saber solo los valores de las tablas A o B que tienen valores sin ninguna entrada equivalente en la otra tabla. Esto nos puede valer para llegar a saber, por ejemplo, qué CINES tenemos sin SALAS dadas de alta o qué SALAS existen sin CINE. Aunque esta segunda es más complicada que exista si tenemos nuestra base de datos bien creada.

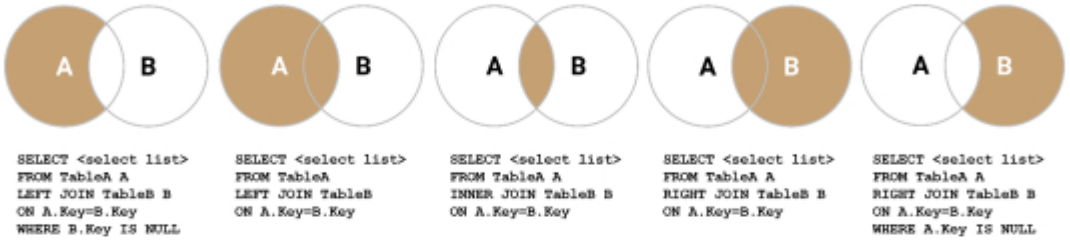

Fig. 6.6. Diferentes uniones internas que podemos utilizar con SQL JOIN.

La consulta anterior con uniones internas quedaría de la siguiente manera:

```
SELECT cine.idCine AS "cine-Cine", sala.idCine AS "cine-Sala",
cine.nombreCine, cine.poblaciónCine, sala.idSala, sala.butacasSala
FROM cine INNER JOIN sala ON cine.idCine = sala.idCine;
```

De este modo, dejamos la cláusula `WHERE` nuevamente para aplicar filtros.

IMPORTANTE

Un subtipo de `INNER JOIN` es `NATURAL JOIN`; cuando los campos claves se llaman igual, como es este caso, podríamos aplicarlo y quedaría así:

```
SELECT cine.idCine AS "cine-Cine", sala.idCine AS "cine-Sala",
cine.nombreCine, cine.poblaciónCine, sala.idSala, sala.butacasSala
FROM cine NATURAL JOIN sala;
```

O en este caso simplemente:

```
SELECT * FROM cine NATURAL JOIN sala;
```

Obtenemos el mismo resultado.

B. Consultas multitabla o de unión filtradas

Ahora que podemos construir consultas multitabla, queremos poder filtrar por el criterio que nos interese para no ver todos los registros: para poder ver todos los cines SOLO de Cullera que tengan butacas dadas de alta en la tabla SALA. Cuando un nombre de campo coincida en más de una tabla deberemos anteponer el nombre de la tabla al campo separado por un punto para saber qué campo queremos.

```
SELECT sala.idCine, nombreCine, poblaciónCine, sala.idSala,
butacasSala FROM cine INNER JOIN sala ON cine.idCine = sala.idCine
WHERE cine.poblaciónCine = "Cullera";
```

idCine	nombreCine	poblaciónCine	idCine	idSala	butacasSala
2	Cervantes	Cullera	2	1	188

Fig. 6.7. Resultado de la consulta de Cines con Salas en Cullera desde phpMyAdmin.

Si ahora quisiéramos ver todos los cines de Cullera que tenemos dados de alta en nuestra tabla de cines, cambiaríamos la unión interna INNER por la de LEFT para verlos todos.

```
SELECT cine.idCine, nombreCine, poblaciónCine, sala.idCine,
sala.idSala, butacasSala
FROM cine LEFT JOIN sala ON cine.idCine = sala.idCine WHERE
cine.poblaciónCine = "Cullera"
```

idCine	nombreCine	poblaciónCine	idCine	idSala	butacasSala
2	Cervantes	Cullera	2	1	188
22	Cines Victoria	Cullera	NULL	NULL	NULL

Fig. 6.8. Resultado de la consulta de todos los CINES de Cullera desde phpMyAdmin.

Si ahora quisiéramos saber cuántos cines de Cullera tenemos dados de alta en nuestra tabla de cines que aún no se han dado de alta sus salas con sus butacas, en vez de dos registros, la consulta anterior podría haber devuelto dos mil registros, por lo que esta consulta solo muestra los que aún no tienen salas.

```
SELECT cine.idCine, nombreCine, poblaciónCine, sala.idCine,
sala.idSala, butacasSala FROM cine LEFT JOIN sala ON cine.idCine =
sala.idCine WHERE cine.poblaciónCine = "Cullera" AND sala.idCine IS
NULL
```

idCine	nombreCine	poblaciónCine	idCine	idSala	butacasSala
22	Cines Victoria	Cullera	NULL	NULL	NULL

Fig. 6.9. Resultado de la consulta de todos los CINES sin SALSAS de Cullera desde phpMyAdmin.