

3. Consultas multitable

3.1. Objetivos

Los objetivos de esta actividad son:

- Realizar consultas con datos de más de una tabla utilizando la composición de tablas y la unión de consultas.

Para los ejemplos, seguiremos empleado la base de datos de *empleados* utilizada en ejemplo anteriores.

3.2. Actividad

3.2.1. Consultas con datos de más de una tabla

En la primera actividad de esta unidad didáctica se han realizado consultas simples con datos de una única tabla, pero la práctica diaria de consultas sobre una base de datos implica frecuentemente hacerlos sobre más de una tabla. Este tipo de consultas se pueden realizar de varias formas:

- Utilizar composiciones de tablas en una sentencia SELECT escribiendo una única consulta que utiliza la cláusula JOIN dentro de la cláusula FROM.
- Utilizar sentencias SELECT anidadas dentro de otras sentencias SELECT. Las sentencias anidadas se llaman subconsultas y se verán más adelante.
- Uniendo el conjunto de resultados de una consulta con el conjunto de resultados de otras consultas, empleando el operador UNION.

3.2.2. Composición de tablas

Las primeras normas ANSI SQL para combinar varias tablas en una consulta permitían poner la relación de las tablas en la cláusula FROM separadas por coma, y las condiciones para hacer los enlaces entre las tablas en la cláusula WHERE. Esta sintaxis sigue estando permitida hoy en día, aunque no es la más recomendable.

Por ejemplo: seleccionar los *apellidos*, *nombre* y *ciudad* de los empleados que estén asignados a un departamento, teniendo en cuenta que la ciudad es la ciudad en la que está ubicado el departamento en el que trabaja el empleado.

```
/* Enlace entre dos tablas utilizando una clave foránea*/
```

```
select apellidos, empleado.nombre, ciudad  
from empleado, departamento  
where cod_departamento = codigo;
```

La primera operación que se realiza con una sentencia como la anterior en la que se hace referencia a más de una tabla en la cláusula FROM, es el producto cartesiano entre esas tablas, es decir, relaciona cada fila de una tabla con todas las filas de la otra tabla. Después

hay que poner la condición de enlace en la cláusula **WHERE** para que se seleccionen sólo aquellas filas que nos interesan en función de la columna correspondiente.

El número de filas que devuelve la consulta del ejemplo es 8, tal y como se observa en la imagen siguiente, a pesar de que hay 9 empleados. Esto es debido a que hay un empleado que tiene en la columna *cod_departamento* el valor **NULL**, por no estar asignado a ningún departamento, y no se puede combinar con ninguna fila de la tabla *departamento* porque la columna *código* es clave primaria y no puede tener el valor **NULL**.



	apellidos	nombre	ciudad
►	Case Rodriguez	Fernanda	Vigo
	Martinez Iglesias	Benito	Lugo
	Núñez Bernardez	Antonia	Monforte
	Fernandez Lopez	Jose Luis	Lugo
	Ruiz Macias	Dario	Villalba
	Hernandez Valin	Valentina	Ferrol
	Martinez Diaz	Carlos	Monforte
	Villar Bernal	Teolindo	Ourense

La sintaxis de las condiciones de tablas cambia a partir de la norma ANSI92, para poner las condiciones de enlace en la cláusula **FROM** junto con la relación de tablas, con la finalidad de dejar la cláusula **WHERE** para las condiciones que deben cumplir las filas a mostrar.

En esta actividad, se utilizará la sintaxis que recomienda la norma ANSI92 por ser más eficiente y estructurar mejor el código de las consultas. Por tanto, utilizaremos un tipo de cláusula llamada **JOIN** que se utilizara para realizar la composición de varias tablas. Veremos cuatro tipos de **JOIN**: **INNER JOIN**, **LEFT JOIN**, **RIGHT JOIN** y **CROSS JOIN**.

La documentación oficial de esta cláusula es la siguiente:

<https://dev.mysql.com/doc/refman/8.0/en/join.html>

También podemos encontrar un buen resumen de esta cláusula en el siguiente enlace:

https://www.w3schools.com/MySQL/mysql_join.asp

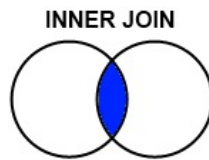
Nombres de columna cualificados

Al combinar varias tablas, hay que tener muy en cuenta que puede haber columnas que tengan el mismo nombre en dos o más tablas. Cuando sucede esto es obligatorio cualificar los nombres de las columnas, utilizando el formato *nombre_tabla.nombre_columna*, para evitar errores producidos por el uso de nombres ambiguos.

También es recomendable utilizar los nombres de columna cualificados para mejorar el rendimiento de la consulta, porque proporcionan información al servidor y así no tiene que buscar la tabla a la que pertenece cada columna. Para simplificar el uso de nombres cualificados y que estos sean más cortos, se recomienda utilizar alias para los nombres de tabla. Por ejemplo:

```
select emp.apellidos, emp.nombre, dep.ciudad
from empleado as emp, departamento as dep
where emp.cod_departamento = dep.codigo;
```

3.2.2.1. INNER JOIN



Este tipo de composición es la más utilizada y permite relacionar dos tablas que tienen alguna columna con datos comunes que sirve de enlace entre ellas. El caso más normal es el que establecen las claves foráneas de una tabla, que toman valores que tienen que coincidir con los valores de la clave primaria de otra tabla. Esta cláusula da como resultado el conjunto formado por las parejas de filas de las dos tablas que intervienen en la composición, que cumplan la condición de que el valor de la clave foránea de una sea igual al valor de la clave primaria de la otra.

La sintaxis del INNER JOIN es:

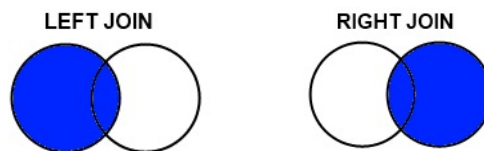
```
SELECT columnas
FROM tabla_1 INNER JOIN tabla_2
ON (condición de enlace)
```

Por ejemplo: selecciona los *apellidos*, *nombre* y *ciudad* de los empleados que estén asignados a un departamento, teniendo en cuenta que *ciudad* es la ciudad en la que está ubicado el departamento en el que trabaja el empleado.

```
select emp.apellidos, emp.nombre, dep.ciudad
from empleado as emp inner join departamento as dep
on (emp.cod_departamento = dep.codigo);
```

3.2.2.2. LEFT JOIN y RIGHT JOIN

Las cláusulas LEFT JOIN y RIGHT JOIN permiten devolver todas las filas de una de las tablas y las filas relacionadas de la otra tabla.



La sintaxis es:

```
SELECT columnas
FROM tabla_1 {LEFT | RIGHT} JOIN tabla_2
ON (condición de enlace)
```

- La opción LEFT muestra todas las filas de la tabla de la izquierda, aunque no estén relacionadas con filas del resto de las tablas.
- La opción RIGHT, muestra todas las filas de la tabla de la derecha, aunque no estén relacionadas con filas del resto de las tablas.

Por ejemplo: mostrar los *apellidos* y *nombre* de los empleados y nombre de la ciudad en la que está el departamento en el que trabaja, aunque el empleado no tenga departamento asignado. El resultado tiene que estar ordenado por el nombre del empleado.

```

select emp.apellidos, emp.nombre, dep.ciudad
from empleado as emp left join departamento as dep
on (emp.cod_departamento = dep.codigo)
order by emp.nombre;

```

En la sentencia anterior, la tabla principal es la tabla *empleado* por estar situada a la izquierda (*left*), y la tabla *departamento* se considera la tabla secundaria de la consulta. En la ejecución, se mostrará una fila para cada empleado, y para quienes no cumplan la condición de enlace por no estar asignado a ningún departamento, se muestra el valor NULL en la columna *ciudad*. Por tanto, el resultado es el siguiente:

Result Grid			
Filter Rows:			
	apellidos	nombre	ciudad
▶	Iglesias Dominguez	Adolfo	NULL
	Núñez Bernardez	Antonia	Monforte
	Martinez Iglesias	Benito	Lugo
	Martinez Diaz	Carlos	Monforte
	Ruiz Macias	Dario	Villalba
	Case Rodriguez	Fernanda	Vigo
	Fernandez Lopez	Jose Luis	Lugo
	Villar Bernal	Teolindo	Ourense
	Hernandez Valin	Valentina	Ferrol

Por ejemplo: si queremos mostrar todos los empleados de todos los departamentos, podemos utilizar un RIGHT JOIN:

```

select emp.apellidos, emp.nombre, dep.codigo
from empleado as emp right join departamento as dep
on (emp.cod_departamento = dep.codigo)
order by dep.codigo;

```

En este caso, la tabla principal es la de *departamento* por estar a la derecha. Entonces, se mostrarán todos los departamentos relacionados con los empleados de cada uno de ellos. Para los departamentos que no tengan empleados, se mostrará un NULL. El resultado sería:

Result Grid			
Filter Rows:			
	apellidos	nombre	codigo
▶	Martinez Iglesias	Benito	1
	Fernandez Lopez	Jose Luis	1
	Núñez Bernardez	Antonia	2
	Martinez Diaz	Carlos	2
	Hernandez Valin	Valentina	3
	Case Rodriguez	Fernanda	4
	Villar Bernal	Teolindo	5
	NULL	NULL	6
	NULL	NULL	7
	NULL	NULL	8
	NULL	NULL	9
	Ruiz Macias	Dario	10

3.2.2.3. CROSS JOIN

La cláusula CROSS JOIN devuelve el producto cartesiano de las filas de ambas tablas. Es decir, relaciona todas las filas de la primera tabla con todas las filas de la segunda tabla. Por tanto, hay que tener en cuenta que el tamaño de los resultados devueltos puede ser muy grande. La sintaxis es:

```
SELECT columnas
FROM tabla_1 CROSS JOIN tabla_2
```

Normas para la realización de composiciones

No existen normas estándar para las composiciones, pero a continuación se enumeran una serie de conclusiones que ayudarán a utilizarlas mejor:

- Las columnas que se utilizan para hacer la composición deben ser del mismo tipo.
- Cuando se hace una composición, es como si se hubiese creado una tabla que tiene las columnas de todas las tablas que se combinan, y las filas que verifican las condiciones de composición. Por esta razón, en la sentencia SELECT puede utilizarse cualquier columna de las tablas que se relacionan en la cláusula FROM.
- Se pueden combinar tantas tablas como deseemos, pero poner tablas innecesarias reduce el rendimiento de la consulta.
- Una misma tabla puede aparecer más de una vez en la composición, pero utilizando alias diferentes.
- Se recomienda utilizar nombres de columna cualificados.
- Si una tabla tiene una clave primaria compuesta, en las condiciones de composición hay que hacer referencia a la clave entera.

3.2.3.Unión de consultas (UNION)

La cláusula UNION se utiliza para combinar el resultado de varias consultas en un único conjunto de resultados. La sintaxis es:

```
Sentencia SELECT
UNION [ALL | DISTINCT]
Sentencia SELECT
[UNION [ALL | DISTINCT]
Sentencia SELECT] ...
```

- Las opciones ALL y DISTINCT permiten indicar si hay que mostrar las filas duplicadas o no. El valor por defecto es DISTINCT. Para que se muestren las filas duplicadas hay que poner la opción ALL.
- Las sentencias SELECT que se utilizan en la UNION tienen que tener el mismo número de columnas en la lista de selección. Las columnas seleccionadas tienen que ser datos de tipos similares en todas las sentencias SELECT. Por ejemplo, si la primera consulta selecciona tres columnas, el resto de las sentencias SELECT deben seleccionar también tres columnas; si la primera columna de la primera consulta es de tipo char(50), la

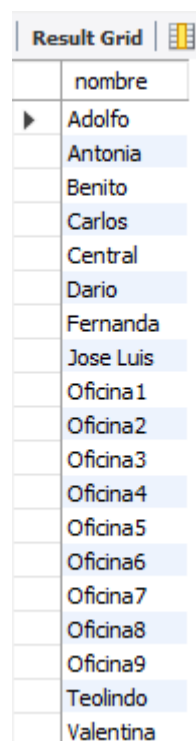
primera columna del resto de sentencias SELECT tiene que ser de tipo char(50). Se admiten excepciones con respecto al tipo de dato si MySQL puede realizar una conversión de un tipo a otro (por ejemplo, si la primera consulta utiliza un VARCHAR y la segunda utiliza un INT, MySQL podría convertir los números a formato texto e interpretarlos como VARCHAR).

- Las columnas de cada sentencia SELECT deben estar en el mismo orden.
- Los nombres de las columnas de la tabla resultante de la UNION son los que corresponden a los nombres de las columnas de la primera sentencia SELECT.
- Cuando se quieren utilizar las cláusulas ORDER BY o LIMIT, hay que ponerlas después de la última sentencia SELECT.

Un ejemplo de UNION con la base de datos de *empleados* podría ser: seleccionar todos los nombres de empleados y departamentos y mostrarlos por orden alfabético.

```
select nombre from empleado
union
select nombre from departamento
order by nombre;
```

El resultado será el siguiente:



nombre
Adolfo
Antonia
Benito
Carlos
Central
Dario
Fernanda
Jose Luis
Oficina1
Oficina2
Oficina3
Oficina4
Oficina5
Oficina6
Oficina7
Oficina8
Oficina9
Teolindo
Valentina

Vemos como aparecen mezclados los nombres de los empleados y los nombres de los departamentos. Esto es posible porque en ambas tablas existe la columna *nombre* y tiene el mismo tipo: varchar(45).