UD4. Consultas con varias tablas

1. Obtener datos de múltiples tablas

Las bases de datos relacionales almacenan sus datos en varias tablas. Lo normal, en casi cualquier consulta, es requerir datos de varias tablas a la vez. Esto es posible porque los datos de las tablas están ligados por columnas que contienen claves secundarias o externas que permiten relacionar los datos de esa tabla con datos de otra tabla.

Como ejemplo, veamos esta tabla de *departamentos*:

cod_dep	Departamento
1	Ventas
2	Producción
3	Calidad
4	Dirección

Por otro lado, tenemos una tabla de *empleados*:

cod_emp	Nombre	Apellido	cod_dep	edad
1	Marisa	León	1	54
2	Arturo	Crespo	1	58
3	Ana	Díez	2	43
4	Pau	Cabanillas	2	29
5	Luisa	Rodríguez	3	34
6	Anxo	Olivenza	4	21
7	Pedro	Andérez		40

La columna *cod_dep* en la tabla de empleados es una clave externa. A través de ella sabemos que *Marisa León*, por ejemplo, es del departamento de *Ventas*.

2. Producto *cruzado* o *cartesiano* de tablas.

Usando las tablas del apartado anterior, si se quiere obtener una lista de los datos de los departamentos y los empleados, se podría hacer de esta forma:

```
SELECT nombre, apellido, departamento
FROM departamentos,empleados
ORDER BY nombre,apellido,departamento;
```

La sintaxis es correcta ya que, efectivamente, en el apartado `FROM` se pueden indicar varias tareas separadas por comas.

Sin embargo el resultado es confuso:

Nombre	Apellido	Departamento
Ana	Díez	Calidad
Ana	Díez	Dirección
Ana	Díez	Producción
Ana	Díez	Ventas
Anxo	Olivenza	Calidad
Anxo	Olivenza	Dirección
Anxo	Olivenza	Producción
Anxo	Olivenza	Ventas
Arturo	Crespo	Calidad
Arturo	Crespo	Dirección
Arturo	Crespo	Producción
Arturo	Crespo	Ventas
Luisa	Rodríguez	Calidad
Luisa	Rodríguez	Dirección
Luisa	Rodríguez	Producción
Luisa	Rodríguez	Ventas
Marisa	León	Calidad
Marisa	León	Dirección
Marisa	León	Producción
Marisa	León	Ventas

Parece que todos los trabajadores trabajan en todos los departamentos. Algo que no es cierto a tenor de los datos originales. *Ana Díez* trabaja en el departamento de Ventas, es lo que nos dice su columna *cod_dep*.

La razón de este resultado es que no hemos utilizado la relación entre las tablas para ligar los datos de ambas. Por ello aparece cada fila de la primera tabla combinada con cada fila de la segunda. Como la primera tabla (*departamentos*) tiene seis filas, y la segunda (*empleados*) cuatro, el resultado son 24 filas.

A esta forma de combinar datos de varias tablas se la conoce como **producto cruzado** o **cartesiano**. Y se utiliza para resolver algunas consultas complejas, pero normalmente no es lo que necesitamos para combinar datos de diferentes tablas.

Normalmente necesitaremos discriminar ese producto para que sólo aparezcan, como es el caso, los datos de los empleados combinados con los de sus departamentos. A eso se le llama asociar (*join*) tablas.

3. Asociaciones de tablas (Sintaxis SQL'92)

La forma de realizar correctamente la consulta anterior (asociando los empleados con sus departamentos), sería:

```
SELECT nombre, apellido, departamento
FROM departamentos,empleados
WHERE departamentos.cod_dep=empleados.cod_dep
ORDER BY nombre,apellido,departamento;
```

Nótese, que se utiliza la notación `tabla.columna` para evitar la ambigüedad. Tanto la tabla de *departamentos* como la de *empleados* tienen una columna llamada **cod_dep**, por ello hay que distinguirla anteponiendo el nombre de la tabla a la que pertenece.

Para evitar repetir continuamente el nombre de la tabla, se puede utilizar un alias de tabla:

```
SELECT nombre, apellido, departamento
FROM departamentos d,empleados e
WHERE d.cod_dep=e.cod_dep
ORDER BY nombre,apellido,departamento;
```

En cualquier caso el resultado muestra realmente los empleados y los departamentos a los que pertenecen.

Nombre	Apellido	Departamento
Ana	Díez	Producción
Anxo	Olivenza	Dirección
Arturo	Crespo	Ventas
Luisa	Rodríguez	Calidad
Marisa	León	Ventas
Pau	Cabanillas	Producción

Al apartado `WHERE` se le pueden añadir condiciones encadenándolas con el operador `AND`. Ejemplo:

```
SELECT nombre, apellido, departamento
FROM departamentos d,empleados e
WHERE d.cod_dep=e.cod_dep AND nombre='Ana'
ORDER BY nombre,apellido,departamento;
```

3.1 Asociaciones de más de dos tablas

Por supuesto es posible asociar datos de más de dos tablas. Por ejemplo supongamos que tenemos estas tablas, mostradas desde su diseño relacional:

Queremos mostrar el nombre y apellidos de los alumnos junto con el número de curso a los que están asociados y las fecha de inicio y fin de los mismos.

Aunque los datos que necesitamos están en dos tablas (*alumnos* y *cursos*), no hay relación entre ambas, la relación entre esas tablas es la indicada por la tabla *asistir*, por lo que necesitamos indicar esa tabla. La consulta resultante sería:

```
SELECT nombre,apellido1,apellido2,
  c.n_curso, fecha_inicio,fecha_fin
FROM alumnos a, asistir s, cursos c
WHERE a.dni=s.dni AND c.n_curso=s.n_curso;
```

Por lo tanto, podemos ligar más de dos tablas sin ningún problema.

Al hacer consultas sobre varias tablas hay que tener en cuenta estos detalles:

- Deberemos añadir las tablas que contienen los datos que necesitemos
- Además deberemos añadir las tablas necesarias para asociar correctamente a las tablas anteriores
- Por último, de todas las anteriores, deberemos usar las **mínimas** tablas. Cuantas menos mejor. Especialmente en los inicios de un desarrollador en SQL, conviene tener en cuenta este último punto. Añadir tablas de más puede cambiar el resultado.

Por ejemplo, si en el ejemplo anterior usamos y asociamos una tabla con información sobre los *profesores* de los cursos, solo aparecerían datos de cursos que tienen asignado al menos un profesor, y eso puede cambiar notablemente el resultado.

4. Sintaxis SQL'99

En la versión SQL de 1999 se ideó una nueva sintaxis para consultar varias tablas. La razón fue separar las condiciones de asociación respecto de las condiciones de selección de registros, lo cual otorga una mayor claridad a las instrucciones SQL.

Postgresql es totalmente compatible con esta normativa.

La sintaxis completa de las consultas en formato SQL 99 es:

```
SELECT tabla1.columna1, tabl1.columna2,...
tabla2.columna1, tabla2.columna2,... FROM tabla1
[CROSS JOIN tabla2]|
[NATURAL JOIN tabla2]|
[[LEFT|RIGHT|FULL OUTER] JOIN tabla2 USING(columna)]|
[[LEFT|RIGHT|FULL OUTER] JOIN tabla2
    ON (tabla1.columa=tabla2.columna)]
```

Se explica en detalle las capacidades de esta forma de indicar consultas.

4.1 CROSS JOIN

Utilizando la opción CROSS JOIN se realiza un producto cruzado entre las tablas indicadas (ver el apartado 2). Los productos cruzados se utilizan para consultas más avanzadas (se explican en temas posteriores).

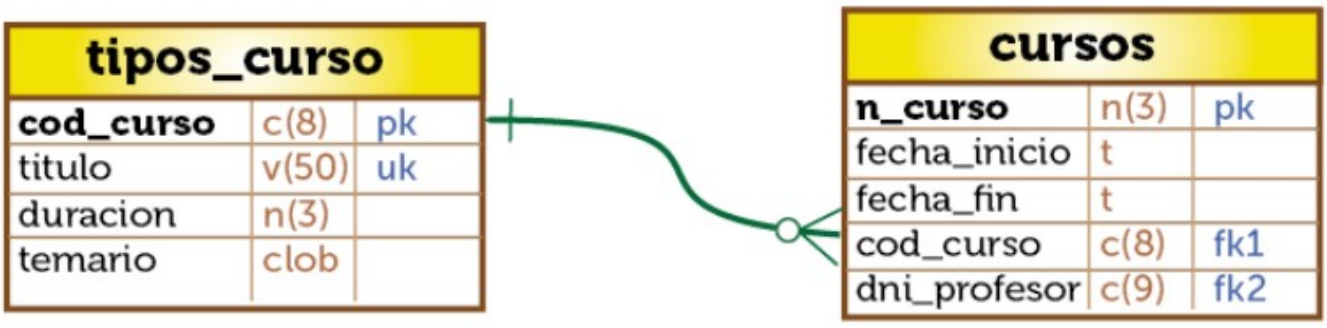
Usando el mismo ejemplo visto anteriormente, en SQL 99 quedaría así el producto cruzado:

```
SELECT nombre, apellido, departamento
FROM departamentos
CROSS JOIN empleados
ORDER BY nombre,apellido,departamento;
```

4.2 NATURAL JOIN

Establece una relación de igualdad entre las tablas a través de los campos que tengan el mismo nombre en ambas tablas. No es raro que las únicas columnas con el mismo nombre entre dos tablas son las que permiten relacionar sus datos.

Para entender mejor la idea observemos este ejemplo:



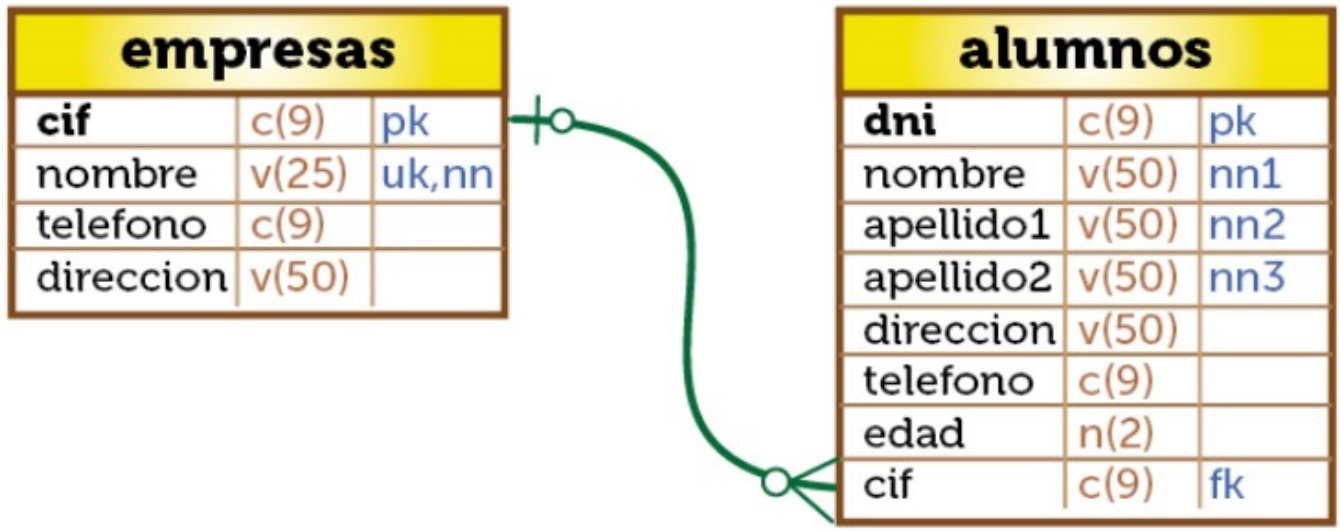
Supongamos que deseamos obtener el *título* de cada tipo de curso junto con los números de curso asociados a ese tipo. Necesitamos usar ambas tablas y observamos que la relación entre ambas tablas la lleva la columna *cod_curso* (presente en ambas tablas) y que, además, es la única columna que tiene el mismo nombre entre ambas tablas.

En ese caso podemos usar un NATURAL JOIN sin problemas:

```
SELECT cod_curso, titulo, n_curso
FROM tipos_curso t
NATURAL JOIN cursos c;
```

Notemos que la columna cod_curso no se ha cualificado. Aunque es la misma columna en ambas tablas, no se considera (como sí ocurriría si hubiéramos usado la forma SQL'92) que haya ambigüedad, ya que `NATURAL JOIN` considera que no hay dos columnas *cod_curso* sino solo una.

Sin embargo observemos este otro diagrama:



Si quisiéramos obtener el nombre de las empresas y el nombre de los alumnos, viendo que ambas tablas se relacionan a través de la columna llamada **cif**, en ellas podíamos tener la tentación de usar este código:

```
SELECT e.nombre, a.nombre
FROM empresas e
NATURAL JOIN alumnos a;
```

La instrucción devuelve un error por cualificar la columna *nombre*. La razón es que, en este caso, no es solo el *cif* la columna común sino también el *nombre*, el *teléfono* y la *dirección*.

Postgresql sólo mostraría, con un `NATURAL JOIN`, datos de ambas tablas si esas cuatro columnas tuvieran los mismos valores. No tiene ningún sentido hacer un `NATURAL JOIN` sobre esas tablas.

> Por ello hay que recordar que `NATURAL JOIN` sólo se puede aplicar a tablas donde las únicas columnas que tienen el mismo nombre son las que sirven para relacionar las tablas.

4.3 JOIN USING

`JOIN USING` permite asociar tablas indicando las columnas que las relacionan, si estas tienen el mismo nombre en ambas tablas. A diferencia de `NATURAL JOIN` no tienen porque ser las únicas con el mismo nombre.

El problema indicado al final del apartado anterior se puede resolver mediante JOIN USING de esta forma.

```
SELECT e.nombre, a.nombre
FROM empresas e
JOIN alumnos a USING(cif);
```

Al indicar que es el cif la columna común, ahora sí hay ambigüedad en las columnas nombre, ya que ahora se necesita discriminar si estamos hablando del nombre de las empresas o del de los alumnos. La consulta funcionará correctamente.

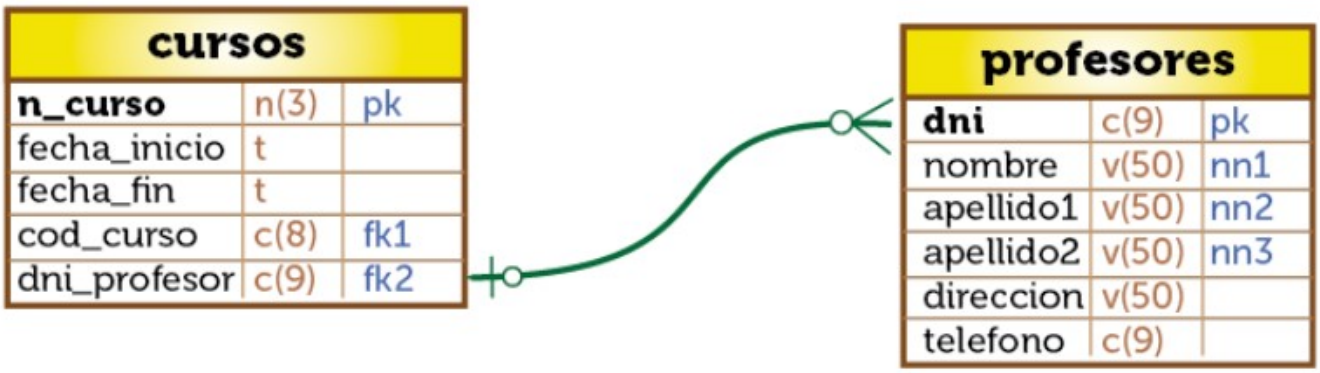
A veces no es solo una columna la que sirve para relacionar, lo cual implica indicar todas las columnas que relacionan. Ejemplo:

```
SELECT tipo,modelo,n_almacen
FROM piezas p
JOIN suministros s USING(tipo,modelo);
```

4.4 JOIN ON

A veces las tablas se relacionan en formas que no encajan con ninguno de los JOIN anteriores. Esta situación se puede entender observando el diagrama la siguiente imagen.

Supongamos que queremos obtener el número de cada curso y el nombre del profesor o profesora que imparte ese curso. Necesitamos usar las tablas *cursos* y *profesores*. Eso implica relacionar ambas y, en este caso, la columna que asocia ambas se llama *dni* en la tabla de *profesores* y *dni_profesor* en la taba de *cursos*.



Esta situación no encaja ni en un `NATURAL JOIN` ni en un `JOIN USING`. Para esto sirve `JOIN ON`.

`JOIN ON` permite indicar la condición (al estilo de SQL 92) que relaciona las tablas. En este caso sería:

```
SELECT n_curso,nombre
FROM cursos c
JOIN profesores p ON(dni=dni_profesor);
```

Hay que señalar que cualquier `JOIN` se puede hacer con `JOIN ON` (sin embargo esto no ocurre al revés). Por ejemplo si tenemos:

```
SELECT e.nombre, a.nombre
FROM empresas e
JOIN alumnos a USING(cif);
```

Podemos hacer un `SELECT` equivalente de esta forma:

```
SELECT e.nombre, a.nombre
FROM empresas e
JOIN alumnos a ON(e.cif=a.cif);
```

> **`JOIN ON` sí requiere cualificar las columnas que se relacionan aunque tengan el mismo nombre.**

4.5 Uso de condiciones en la sintaxis SQL'99

La ventaja principal de la sintaxis 1999 de SQL es la separación entre las indicaciones sobre las relaciones entre las tablas (normalmente referidas a claves principales y claves foráneas relacionadas) y las condiciones que permiten seleccionar las filas que cumplan una determinada condición. Por ejemplo en:

```
SELECT p.nombre nombre, p.apellido1, p.apellido2, e.nombre empresa
FROM personas p
JOIN empresas e USING(cif)
WHERE e.provincia='Sevilla';
```

La cláusula `JOIN` es la que se encarga de ligar las tablas *empresas* y *alumnos* a través de la columna *cif* presente en ambas tablas. De ese modo conseguimos que se nos indique el nombre de la empresa en la que trabaja cada persona. La cláusula `WHERE` restringe esta información para que sólo aparezcan las personas que trabajan en empresas de Sevilla.

Esta consulta también se podría realizar de esta forma:

```
SELECT p.nombre nombre, p.apellido1, p.apellido2, e.nombre empresa
FROM personas p
JOIN empresas e USING(cif)
AND e.provincia='Sevilla';
```

De modo que, incluso, se puede obviar la cláusula `WHERE`.

5. Asociaciones externas

¿Has pensado que puede que te interese seleccionar algunas filas de una tabla aunque éstas no tengan correspondencia con las filas de la otra tabla? Esto puede ser necesario.

Imagina que tenemos en una base de datos guardadas en dos tablas la información de los empleados de la empresa (*Cod_emp, Nombre, Apellido, Cod_dep* y *Edad*) por otro lado los departamentos (*Codigo_dep,*

Departamento) de esa empresa. Recientemente se ha remodelado la empresa y se han creado un par de departamentos más pero no se les ha asignado los empleados. Si tuviéramos que obtener un informe con los datos de los empleados por departamento, seguro que deben aparecer esos departamentos aunque no tengan empleados. Para poder hacer esta combinación usaremos las composiciones externas.

Por ejemplo, dadas estas tablas:

cod_dep	Departamento
1	Ventas
2	Producción
3	Calidad
4	Dirección

cod_emp	Nombre	Apellido	cod_dep	edad
1	Marisa	León	1	54
2	Arturo	Crespo	1	58
3	Ana	Díez	2	43
4	Pau	Cabanillas	2	29
5	Luisa	Rodríguez	3	34
6	Anxo	Olivenza	4	21
7	Pedro	Andérez		40

La siguiente sentencia:

```
SELECT nombre, apellido, departamento
FROM departamentos d
JOIN empleados e USING(cod_dep)
ORDER BY nombre,apellido,departamento;
```

Obtiene el siguiente resultado:

Nombre	Apellido	Departamento
Ana	Díez	Producción
Anxo	Olivenza	Dirección
Arturo	Crespo	Ventas
Luisa	Rodríguez	Calidad
Marisa	León	Ventas
Pau	Cabanillas	Producción

Se observa que no aparece **Pedro Andérez** porque no tiene asignado ningún departamento.

Esto ocurre porque la cláusula **JOIN** utiliza por defecto el valor **INNER**, es decir usan solo los valores internos a la relación. Es decir, como ya se ha dicho, solo aparecen las filas relacionadas en ambas tablas.

> Para poder resolver este problema, necesitamos un `JOIN` de tipo `OUTER`.

5.1 Claúsula OUTER JOIN

Es posible forzar que aparezcan los valores que están fuera de la relación (externos, `OUTER`).

Su sintaxis es:

```
...
{LEFT | RIGHT | FULL} OUTER JOIN tabla
{ON(condición) | USING (expresion)}
...
```

Se puede observar que solo se puede utilizar la cláusula `OUTER` en `JOIN` de tipo `ON` o `USING`.

`LEFT` se indica si queremos que aparezcan **todos los datos de la tabla que queda a la izquierda de la palabra JOIN**

> Hay que recordar que para Postgresql la instrucción es como si tuviera una sola línea, desde la palabra `SELECT` hasta el punto y coma.

De la misma forma, si queremos que sea **la tabla de la derecha la que muestre todos los datos**, se indica `RIGHT`.

Finalmente, **si queremos que sean ambas las que muestren todos los datos** se usa `FULL`.

Así esta consulta:

```
SELECT nombre, apellido, departamento
FROM departamentos d
RIGHT OUTER JOIN empleados e USING(cod_emp)
ORDER BY nombre,apellido,departamento;
```

Muestra:

Nombre	Apellido	Departamento
Ana	Díez	Producción
Anxo	Olivenza	Dirección
Arturo	Crespo	Ventas
Luisa	Rodríguez	Calidad
Marisa	León	Ventas
Pau	Cabanillas	Producción
Pedro	Andérez	

5.2 Consultas no coincidentes

Un caso típico de uso de las relaciones externas es el uso de las llamadas consultas de no coincidentes. Estas consultas permiten obtener datos de una tabla que no se relacionan con otras.

Usando las tablas *empleados* y *departamentos* anteriores. Supongamos que necesitamos saber qué empleados no están asignados a un departamento. La consulta que lo consigue es:

```
SELECT nombre, apellido
FROM departamentos d
RIGHT OUTER JOIN empleados e USING(cod_emp)
WHERE departamento IS NULL
ORDER BY nombre,apellido,departamento;
```

Aparecería solo **Pedro Andérez**, único empleado sin departamento. La consulta funciona ya que forzamos a que aparezcan todos los empleados y luego indicamos (en el `WHERE`) que elimine a los empleados que tengan un departamento.

6. Producto cartesiano para resolver *consultas complejas*

Visto toda esta unidad, los productos cruzados parece que no ofrecen ventajas. Sin embargo, sí es así.

Hay una restricción importante en el manejo de SQL y es el hecho de no poder comparar datos de diferentes filas.

Por ejemplo, usando la tabla de empleados vista anteriormente, supongamos que queremos saber el nombre, apellido y edad de los trabajadores y trabajadoras que tienen más años que **Ana Díez**.

La solución pasa por hacer un producto cruzado de todos los empleados con una tabla que solo muestre a **Ana Diez**.

Así este código:

```
SELECT e.nombre, e.apellido,e.edad,
  a.nombre, a.apellido,a.edad
FROM empleados e
CROSS JOIN empleados a
WHERE a.nombre='Ana' AND a.apellido='Díez';
```

Nombre	Apellido	Edad	Nombre	Apellido	Edad
Marisa	León	54	Ana	Díez	43
Arturo	Crespo	58	Ana	Díez	43
Ana	Díez	43	Ana	Díez	43
Pau	Cabanillas	39	Ana	Díez	43
Luisa	Rodríguez	34	Ana	Díez	43
Anxo	Olivenza	21	Ana	Díez	43
Pedro	Andérez	40	Ana	Díez	43

Este resultado ya está cerca. Rematamos cambiando el código así:

```
SELECT e.nombre, e.apellido,e.edad
FROM empleados e
CROSS JOIN empleados a
WHERE a.nombre='Ana' AND a.apellido='Díez';
   AND e.edad>a.edad;
```

Se obtiene:

Nombre	Apellido	Edad
Marisa	León	54
Arturo	Crespo	58

Este es solo un ejemplo de la potencia de consultas que permite esta técnica. Aunque bien es cierto, que, en muchos casos, se pueden realizar con otras técnicas más sencillas, como veremos en unidades posteriores.

7. Relaciones sin igualdad

A las relaciones descritas en todos los apartados anteriores, salvo los dedicados al producto cruzado, se las llama relaciones en igualdad (*equijoins*), ya que las tablas se relacionan a través de campos que contienen valores iguales en dos tablas. Sin duda, es la situación habitual.

Sin embargo, no siempre las tablas tienen ese tipo de relación. Supongamos que tenemos estas tablas de **Empleados** y de **Categorías**:

Nombre	Sueldo
Antonio	28000
Marta	41000
Sonia	20000

Categoría	Sueldo Mínimo	Sueldo máximo
Dirección	50000	72000
Coordinación	35000	49999
Puesto cualificado	25000	34999
Puesto sin cualificar	18000	24999

En el ejemplo anterior podríamos averiguar la categoría a la que pertenece cada empleado, en relación a donde encaja su salario en la tabla de categorías.

La cuestión es que tenemos que usar otro tipo de relación que no implica el uso de claves secundarias, las cuales se relacionan por su igualdad con claves primarias.

Estamos utilizando otro tipo de relación, pero perfectamente válida. Para saber la categoría de los empleados, tenemos que comprobar que su salario está entre el sueldo mínimo y máximo.

El código que lo permite es (SQL 92):

```
SELECT nombre, sueldo, categoria
FROM empleados e, categorias c
WHERE sueldo BETWEEN sueldo_minimo AND b.sueldo_maximo;
```

También de una forma más semántica (SQL 99):

```
SELECT nombre, sueldo, categoria
FROM empleados e
JOIN categorias c
   ON(sueldo BETWEEN sueldo_minimo AND b.sueldo_maximo);
```

En todo caso obtenemos:

Nombre	Sueldo	Categoría
Antonio	28000	Trabajador cualificado
Marta	41000	Encargado
Sonia	20000	Puesto sin cualificar