Esta consulta, una reunión de la tabla de profesores con la de departamentos, nos sirve para presentar un nuevo detalle del SQL. En la condición de la cláusula `WHERE` queremos establecer una comparación de igualdad entre el campo cod_dep de la tabla de PROFESORES y el mismo campo de la tabla de DEPARTAMENTOS. Como los dos campos tienen el mismo nombre, hemos de cualificar cada uno de ellos indicando la tabla a la que pertenece. De esta forma, eliminamos la ambigüedad que provoca el nombre común para el campo cod_dep de las dos tablas. De hecho, aunque el uso de prefijos aquí es obligatorio, éstos pueden utilizarse siempre que se desee. La sentencia anterior, es equivalente a la siguiente:

```
SELECT profesores.nom_prof, profesores.NRP,
       departamentos.nom_dep
FROM profesores, departamentos
WHERE profesores.cod_dep=departamentos.cod_dep;
```

Como puede observarse en la sintaxis expuesta al principio de este apartado, en la cláusula `FROM` puede escribirse el nombre de más de dos tablas. De esta forma se pueden realizar consultas en las que se realice el producto cartesiano de tantas tablas como se desee.

7.5.3. Subconsultas

La capacidad del SQL para la realización de consultas es muy alta. De hecho pueden construirse consultas que involucren a otras consultas en su resolución. Es lo que se denomina *uso de subconsultas*.

Existen en SQL distintos operadores que permiten operar sobre el resultado de una consulta. De esta forma, en la cláusula `WHERE` de una consulta, por ejemplo, se puede realizar una consulta adicional para recuperar cierta información de la base de datos que es necesaria para solventar la consulta principal. Veamos algunos ejemplos:

Ejemplo 7.35. *Mostrar el DNI de los alumnos de Granada.*

```
SELECT DNI
FROM alumnos
WHERE provincia='GRANADA';
```

Ejemplo 7.36. *Mostrar el DNI de los alumnos que son de la misma provincia que el alumno JUAN LOPEZ.*

Ya no podemos resolver la consulta siguiendo el anterior esquema. Primero tenemos que averiguar la provincia de JUAN LOPEZ y, conocido ese dato, podríamos plantear la consulta. Es decir, primero tendríamos que plantear la siguiente consulta:

```
SELECT provincia
FROM alumnos
WHERE nom_alum='JUAN LOPEZ';
```

Si el resultado de la consulta anterior es la provincia 'X', entonces, la segunda consulta sería:

```
SELECT DNI
FROM alumnos
WHERE provincia='X';
```

Una forma de resolver el problema anterior con una sola consulta, es la siguiente:

```
SELECT DNI
FROM alumnos
WHERE provincia = (
                  SELECT provincia
                  FROM alumnos
                  WHERE nom_alum='JUAN LOPEZ');
```

La consulta anterior es válida en SQL siempre y cuando sólo haya un alumno que se llame JUAN LOPEZ. En ese caso, el resultado de la subconsulta se calcula primero y luego se utiliza para resolver la expresión que aparece en la cláusula `WHERE` de la consulta principal.

En ejemplos como éste, cuando el resultado de la subconsulta no guarda ninguna relación con la tupla concreta que se está analizando en la consulta principal, se dice que las dos consultas no están enlazadas o ligadas. Es decir, son independientes. La mayoría de los sistemas optan por resolver primero la subconsulta y luego, sustituyen el resultado de la subconsulta en la expresión de la consulta principal donde aparece.

SQL también permite utilizar subconsultas cuyo resultado dependa de la tupla concreta de la consulta principal que se está resolviendo. Consideremos el siguiente ejemplo:

Ejemplo 7.37. *Mostrar aquellos profesores que tienen la misma categoría que su director de departamento.*

```
SELECT p1.NRP
FROM profesores p1
WHERE categoria = (
                  SELECT categoria
                  FROM profesores p2, departamentos
                  WHERE p2.NRP=departamentos.director
                    AND departamento.cod_dep=p1.cod_dep);
```

La consulta anterior nos sirve también para presentar el uso de alias para los nombres de las tablas en SQL. En algunas consultas, es necesario utilizar más de una vez la misma tabla. Para poder referirnos a cada una de las apariciones de una tabla sin ambigüedades, es necesario utilizar nombres alternativos (tal como vimos al estudiar el AR en el capítulo anterior). Esto se hace colocando un nuevo identificador a continuación del nombre de la tabla a la que queremos asociar el alias. En la consulta anterior, la instancia de la tabla PROFESORES que está siendo utilizada en la cláusula `FROM` de la consulta principal, se va a llamar `p1`, mientras que la tabla PROFESORES de la subconsulta la vamos a identificar con el alias `p2`.

Por tanto, la interpretación de la consulta anterior es la siguiente. Para cada tupla de la tabla PROFESORES (`p1`) consultamos la categoría del director de su departamento. Para ello, necesitamos volver a utilizar la tabla profesores (`p2`), hacer una reunión con DEPARTAMENTOS (`p2.NRP=departamentos.director`) y fijarnos sólo en el departamento del profesor que está siendo analizado en la consulta principal (`departamento.cod_dep=p1.cod_dep`).

Este tipo de consultas se dice que están encadenadas o que son dependientes. El sistema no puede ejecutar la subconsulta al principio para luego sustituir y resolver la consulta principal, ya que el resultado de la subconsulta depende de la tupla concreta que está siendo analizada en la consulta principal. El uso de este tipo de subconsulta aumenta bastante la complejidad de la consulta principal. Si la instancia de profesores tiene 100 tuplas, en el anterior ejemplo habría que calcular la subconsulta 100 veces.

Operadores para las subconsultas

El operador `IN` que ya hemos utilizado con anterioridad también puede utilizarse en combinación con una subconsulta. Este operador booleano controla si un valor se

encuentra entre los elementos de un conjunto. Ese conjunto puede venir especificado en forma de lista explícita de valores o bien puede construirse mediante el uso de una subconsulta. Veamos un par de ejemplos.

Ejemplo 7.38. *Mostrar los alumnos que están cursando alguna asignatura optativa.*

```
SELECT DNI
FROM matriculas
WHERE cod_asig IN
      (SELECT cod_asig
       FROM asignaturas
       WHERE caracter='op');
```

En este caso, para que el alumno representado por una tupla de la tabla MATRICULAS forme parte del resultado, la asignatura de la que esté matriculado deberá formar parte del conjunto de asignaturas optativas que calcula la subconsulta.

El operador `IN` tiene su complementario en el operador `NOT IN`.

En general, el operador `IN` comprueba si una tupla pertenece a un conjunto de tuplas. En este caso, los valores de la tupla se ponen entre paréntesis y separados por comas.

Ejemplo 7.39. *Mostrar aquellas asignaturas para las que existe otra distinta con el mismo carácter y los mismos créditos.*

```
SELECT cod_asig
FROM asignaturas a1
WHERE (a1.caracter,a1.creditos) IN
      (SELECT a2.caracter, a2.creditos
       FROM asignaturas a2
       WHERE a2.cod_asig<>a1.cod_asig);
```

Otro operador booleano de notable utilidad cuando se utilizan subconsultas es el operador `[NOT] EXISTS`. Este operador admite como operando una subconsulta y nos indica si ésta devuelve al menos una tupla, sin importar el contenido ni el número de campos de esa tupla. Veamos un ejemplo de su uso:

Ejemplo 7.40. *Mostrar los nombres de las asignaturas que tienen algún alumno matriculado.*

```
SELECT nom_asig
FROM asignaturas
WHERE EXISTS
    (SELECT *
     FROM matriculas
     WHERE matriculas.cod_asig=asignaturas.cod_asig);
```

Es importante resaltar aquí que una consulta puede construirse de muchas formas alternativas en SQL. Por ejemplo, la anterior consulta podría haberse escrito de una forma más sencilla:

```
SELECT DISTINCT nom_asig
FROM matriculas, asignaturas
WHERE matriculas.cod_asig=asignaturas.cod_asig;
```

El resultado es el mismo, pero el tiempo empleado por el sistema para resolver ambas consultas puede que no lo sea. En realidad, la mayoría de los SGBDs comerciales incorporan un módulo optimizador de consultas que elabora planes para la solución de las consultas que el usuario plantea. Teniendo en cuenta equivalencias como la anterior, el optimizador trata de minimizar el tiempo empleado en la resolución de las consultas.

Subconsultas en la cláusula FROM

En algunos casos, puede interesarnos trabajar sobre la tabla *virtual* que nos devuelve una consulta para resolver otra. SQL permite poner subconsultas en la cláusula `FROM` de una sentencia `SELECT`. El uso conjunto de la subconsulta y de un alias para la misma permite resolver consultas complejas que de otro modo tendrían una difícil solución.

Ejemplo 7.41. *Mostrar los datos (nombre y número de créditos) de las asignaturas que se imparten en un aula con capacidad mayor que 100.*

```
SELECT nom_asig, creditos
FROM (SELECT DISTINCT cod_asig
      FROM clase,aula
      WHERE clase.cod_aula=aula.cod_aula AND capacidad>100) a1,
      asignaturas
WHERE a1.cod_asig=asignaturas;
```