4.C. Consultas multitabla y subconsultas (copia)

Sitio:	ALONSO DE AVELLANEDA
Curso:	Bases de datos DAM Vesp
Libro:	4.C. Consultas multitabla y subconsultas (copia)

Imprimido por:	Oliver Bitica
Día:	lunes, 9 de febrero de 2026, 20:47

Tabla de contenidos

1. Consultas multitablas

Recuerda que una de las propiedades de las bases de datos relacionales era que distribuíamos la información en varias tablas que a su vez estaban relacionadas por algún campo común. Así evitábamos repetir datos. Por tanto, también será frecuente que tengamos que consultar datos que se encuentren distribuidos por distintas tablas.

Si disponemos de una tabla USUARIOS cuya clave principal es Login y esta tabla a su vez está relacionada con la tabla PARTIDAS a través del campo **Cod_Creación**. Si quisiéramos obtener el nombre de los usuarios y las horas de las partidas de cada jugador necesitaríamos coger datos de ambas tablas pues las horas se guardan en la tabla PARTIDAS. Esto significa que cogeremos filas de una y de otra.

Imagina también que en lugar de tener una tabla USUARIOS, dispusiéramos de dos por tenerlas en servidores distintos. Lo lógico es que en algún momento tendríamos que unirlas.

Hasta ahora las consultas que hemos usado se referían a una sola tabla, pero también es posible hacer consultas usando varias tablas en la misma sentencia `SELECT`. Esto permitirá realizar distintas operaciones como son:

- La composición interna.
- La composición externa.

En la versión SQL de 1999 se especifica una nueva sintaxis para consultar varias tablas que Oracle incorpora, así que también la veremos. La razón de esta nueva sintaxis era separar las condiciones de asociación respecto a las condiciones de selección de registros.

La sintaxis es la siguiente:

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

1.1. Composiciones internas

¿Qué ocurre si combinamos dos o más tablas sin ninguna restricción? El resultado será un producto cartesiano.

El producto cartesiano entre dos tablas da como resultado todas las combinaciones de todas las filas de esas dos tablas.

Se indica poniendo en la cláusula `FROM` las tablas que queremos componer separadas por comas. Y puedes obtener el producto cartesiano de las tablas que quieras.

Como lo que se obtiene son todas las posibles combinaciones de filas, debes tener especial cuidado con las tablas que combinas. Si tienes dos tablas de 10 filas cada una, el resultado tendrá 10x10 filas, a medida que aumentemos el número de filas que contienen las tablas, mayor será el resultado final, con lo cual se puede considerar que nos encontraremos con una operación costosa.

Esta operación no es de las más utilizadas ya que coge una fila de una tabla y la asocia con todos y cada uno de las filas de la otra tabla, independientemente de que tengan relación o no. Lo más normal es que queramos seleccionar los registros según algún criterio.

Necesitaremos **discriminar** de alguna forma para que únicamente aparezcan filas de una tabla que estén relacionadas con la otra tabla. A esto se le llama **asociar tablas** (`JOIN`).

Para hacer una composición interna se parte de un producto cartesiano y se eliminan aquellas filas que no cumplen la condición de composición.

Lo importante en las composiciones internas es emparejar los campos que han de tener valores iguales.

Las reglas para las composiciones son:

- Pueden combinarse tantas tablas como se desee.
- El criterio de combinación puede estar formado por más de una pareja de columnas.
- En la cláusula `SELECT` pueden citarse columnas de ambas tablas, condicionen o no, la combinación.
- Si hay columnas con el mismo nombre en las distintas tablas, deben identificarse especificando la tabla de procedencia o utilizando un alias de tabla.

Las columnas que aparecen en la cláusula `WHERE` se denominan **columnas de emparejamiento** ya que son las que permiten emparejar las filas de las dos tablas. Éstas no tienen por qué estar incluidas en la lista de selección. Emparejaremos tablas que estén relacionadas entres sí y además, una de las columnas de emparejamiento será clave principal en su tabla. Cuando emparejamos campos debemos especificar de la siguiente forma: `NombreTabla1. Camporelacionado1 = NombreTabla2.Camporelacionado2`.

Puedes combinar una tabla consigo misma pero debes poner de manera obligatoria un alias a uno de los nombres de la tabla que vas a repetir.

Veamos un ejemplo, si queremos obtener el historial laboral de los empleados incluyendo nombres y apellidos de los empleados, la fecha en que entraron a trabajar y la fecha de fin de trabajo si ya no continúan en la empresa, tendremos:

```
SELECT Nombre, Apellido1, Apellido2, Fecha_inicio, Fecha_fin
FROM EMPLEADOS, HISTORIAL_LABORAL
WHERE HISTORIAL_LABORAL.Empleado_DNI= EMPLEADOS.DNI;
```

Vamos a obtener el historial con los nombres de departamento, nombre y apellidos del empleado de todos los departamentos:

```
SELECT Nombre_Dpto, Nombre, Apellido1, Apellido2
FROM DEPARTAMENTOS, EMPLEADOS, HISTORIAL_LABORAL
WHERE EMPLEADOS.DNI= HISTORIAL_LABORAL. EMPLEADO_DNI
AND HISTORIAL_LABORAL.DPTO_COD = DEPARTAMENTOS. DPTO_COD;
```

1.2. Ejercicio 7

Sobre la base de datos implementada en el ejercicio 1 resolver las siguientes consultas SQL:

1. Nombre de los empleados junto a su salario mostrando los diferentes salarios que ha tenido en la empresa indicando la fecha de inicio de cada uno de ellos.

2. Histórico laboral de un empleados cuyo DNI sea '12345'. En dicho listado interesa conocer el nombre del puesto y el rango salarial.

Enlace a la solución

1.3. Composiciones 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_empleado, Nombre, Apellidos, salario y Cod_dpto) por otro lado los departamentos (Codigo_dep, Nombre) 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.

¿Cómo es el formato? Muy sencillo, añadiremos un signo más entre paréntesis (+) en la igualdad entre campos que ponemos en la cláusula WHERE. El carácter (+) irá detrás del nombre de la tabla en la que deseamos aceptar valores nulos.

En nuestro ejemplo, la igualdad que tenemos en la cláusula WHERE es Cod_dpto (+)= Codigo_dep ya que es en la tabla empleados donde aparecerán valores nulos.

Ejemplo: Se pretende obtener un listado con los nombres de los distintos departamentos y sus jefes incluyendo los datos personales pero queremos que aparezcan incluso los departamentos en los que no hay jefe.

Lo primero que tenemos que hacer es insertar un departamento sin jefe en la tabla DEPARTAMENTOS.

```
Run SQL Command Line

SQL> desc departamentos;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DPTO_COD                                  NOT NULL NUMBER(5)
 NOMBRE_DPTO                               NOT NULL VARCHAR2(30)
 JEFE                                               NUMBER(8)
 PRESUPUESTO                               NOT NULL NUMBER(6)
 PRES_ACTUAL                                        NUMBER(6)

SQL> INSERT INTO DEPARTAMENTOS VALUES('99999','DATAMINING', NULL, 20000,14000);

1 row created.
```

Ahora comenzamos con la consulta, los datos de los departamentos están en la tabla DEPARTAMENTOS y los de empleados en la tabla EMPLEADOS.

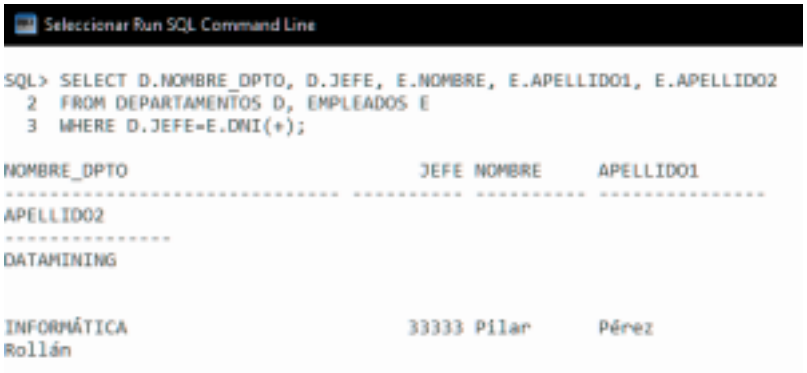
```
Run SQL Command Line
SQL> DESC DEPARTAMENTOS;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DPTO_COD                                  NOT NULL NUMBER(5)
 NOMBRE_DPTO                               NOT NULL VARCHAR2(30)
 JEFE                                               NUMBER(8)
 PRESUPUESTO                               NOT NULL NUMBER(6)
 PRES_ACTUAL                                        NUMBER(6)

SQL> DESC EMPLEADOS;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DNI                                       NOT NULL NUMBER(8)
 NOMBRE                                    NOT NULL VARCHAR2(10)
 APELLIDO1                                 NOT NULL VARCHAR2(15)
 APELLIDO2                                          VARCHAR2(15)
 SALARIO                                            NUMBER(10,2)
 DIRECC1                                            VARCHAR2(25)
 DIRECC2                                            VARCHAR2(20)
 CIUDAD                                             VARCHAR2(20)
 MUNICIPIO                                          VARCHAR2(20)
 COD_POSTAL                                         VARCHAR2(5)
 SEXO                                               CHAR(1)
 FECHA_NAC                                          DATE

SQL>
```

Si realizamos la consulta enlazando las dos tablas por el campo JEFE de DEPARTAMENTOS que contiene el DNI del jefe de departamento y el campo DNI de la tabla EMPLEADOS obtenemos:

Donde solo se muestran las filas que cumplen la condición del WHERE y por lo tanto los departamentos sin jefe no se muestran, sin embargo en esta otra consulta:



Si aparece el departamento de DATAMINING aunque no tenga jefe asociado.

1.4. Composiciones en la versión SQL99

Como has visto, SQL incluye en esta versión mejoras de la sintaxis a la hora de crear composiciones en consultas. Recuerda que la sintaxis es la siguiente:

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

- `CROSS JOIN`: creará un producto cartesiano de las filas de ambas tablas por lo que podemos olvidarnos de la cláusula `WHERE`.

https://www.w3resource.com/sql/joins/cross-join.php

- `NATURAL JOIN`: detecta automáticamente las claves de unión, basándose en el nombre de la columna que coincide en ambas tablas. Por supuesto, se requerirá que las columnas de unión tengan el mismo nombre en cada tabla. Además, esta característica funcionará incluso si no están definidas las claves primarias o ajenas. Si dos tablas están enlazadas por columnas con el mismo nombre (como en los ejemplos anteriores), los INNER JOIN se aplican como **NATURAL JOIN**.

https://www.w3resource.com/sql/joins/natural-join.php

- `JOIN USING`: las tablas pueden tener más de un campo para relacionar y no siempre queremos que se relacionen por todos los campos. Esta cláusula permite establecer relaciones indicando qué campo o campos comunes se quieren utilizar para ello.
- `JOIN ON`: se utiliza para unir tablas en la que los nombres de columna no coinciden en ambas tablas o se necesita establecer asociaciones más complicadas.
- `OUTER JOIN`: se puede eliminar el uso del signo (+) para composiciones externas utilizando un `OUTER JOIN`, de este modo resultará más fácil de entender.
- `LEFT OUTER JOIN`: es una composición externa izquierda, todas las filas de la tabla de la izquierda se devuelven, aunque no haya ninguna columna correspondiente en las tablas combinadas.
- `RIGHT OUTER JOIN`: es una composición externa derecha, todas las filas de la tabla de la derecha se devuelven, aunque no haya ninguna columna correspondiente en las tablas combinadas.
- `FULL OUTER JOIN`: es una composición externa en la que se devolverán todas las filas de los campos no relacionados de ambas tablas.

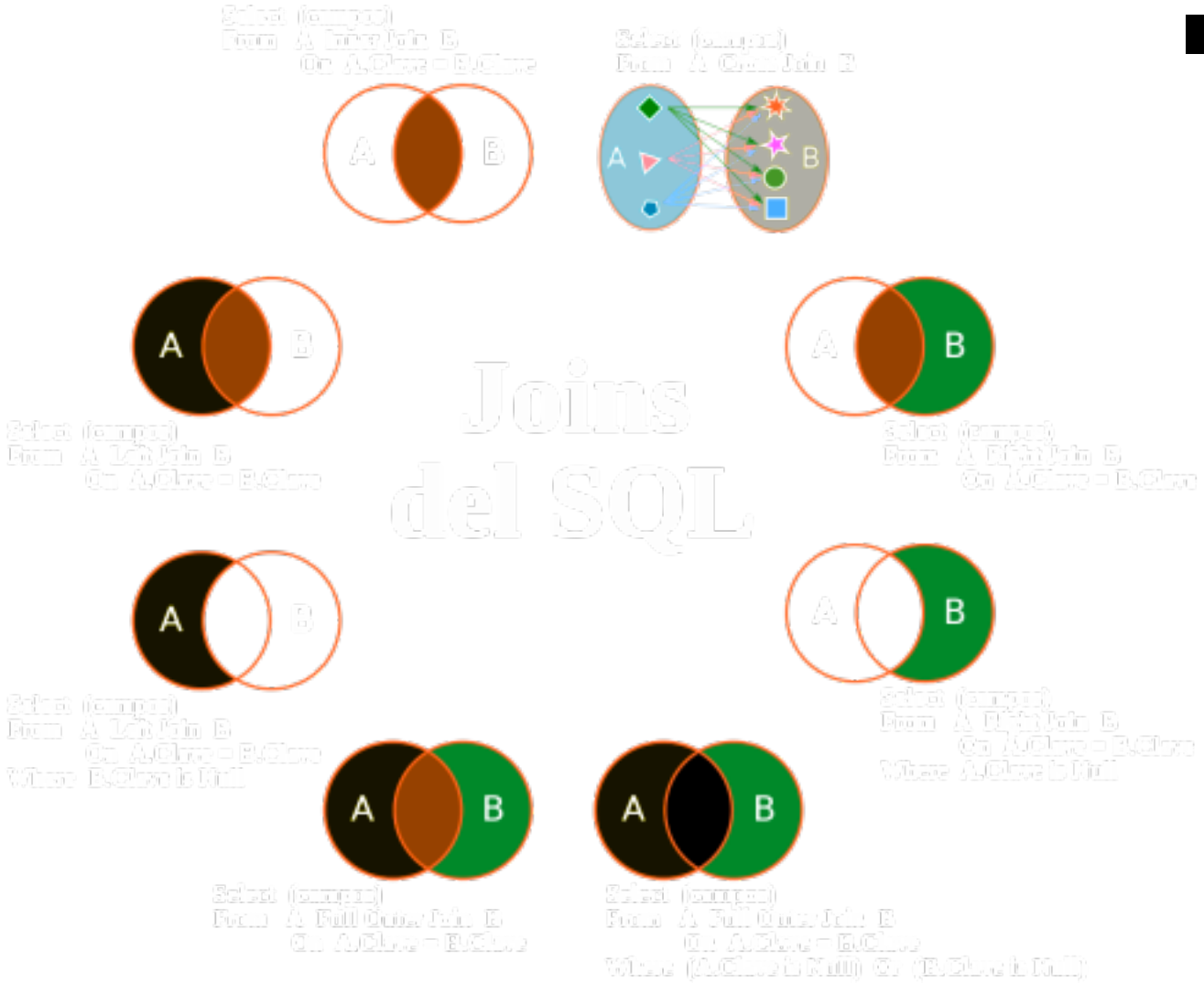
Podríamos transformar algunas de las consultas con las que hemos estado trabajando:

Queríamos obtener el historial laboral de los empleados incluyendo nombres y apellidos de los empleados, la fecha en que entraron a trabajar y la fecha de fin de trabajo si ya no continúan en la empresa. Es una consulta de composición interna, luego utilizaremos `JOIN ON`:

```
SELECT E.Nombre, E.Apellido1, E.Apellido2, H.Fecha_inicio, H.Fecha_fin
FROM EMPLEADOS E JOIN HISTORIAL_LABORAL H ON (H.Empleado_DNI= E.DNI);
```

Queríamos también, obtener un listado con los nombres de los distintos departamentos y sus jefes con sus datos personales. Ten en cuenta que deben aparecer todos los departamentos aunque no tengan asignado ningún jefe. Aquí estamos ante una composición externa, luego podemos utilizar `OUTER JOIN`:

```
SELECT D.NOMBRE_DPTO, D.JEFE, E.NOMBRE, E.APELLIDO1, E.APELLIDO2
FROM DEPARTAMENTOS D LEFT OUTER JOIN EMPLEADOS E ON ( D.JEFE = E.DNI);
```



Referencias:

https://diego.com.es/principales-tipos-de-joins-en-sql

https://www.w3resource.com/sql/joins/sql-joins.php

1.5. Unión, intersección y diferencia de consultas

Seguro que cuando empieces a trabajar con bases de datos llegará un momento en que dispongas de varias tablas con los mismos datos guardados para distintos registros y quieras unirla en una única tabla. ¿Esto se puede hacer? Es una operación muy común junto a otras. Al fin y al cabo, una consulta da como resultado un conjunto de filas y con conjuntos podemos hacer entre otras, tres tipos de operaciones comunes como son: unión, intersección y diferencia.

`UNION`: combina las filas de un primer `SELECT` con las filas de otro `SELECT`, desapareciendo las filas duplicadas.

`INTERSECT`: examina las filas de dos `SELECT` y devolverá aquellas que aparezcan en ambos conjuntos. Las filas duplicadas se eliminarán.

`MINUS`: devuelve aquellas filas que están en el primer `SELECT` pero no en el segundo. Las filas duplicadas del primer `SELECT` se reducirán a una antes de comenzar la comparación.

Para estas tres operaciones es muy **importante** que utilices en los dos `SELECT` el **mismo número** y `tipo` de **columnas** y en el **mismo orden**.

Estas operaciones se pueden combinar anidadas, pero es conveniente utilizar paréntesis para indicar que operación quieres que se haga primero.

Veamos un ejemplo de cada una de ellas.

UNIÓN: Obtener los nombres y ciudades de todos los proveedores y clientes de Alemania.

```
SELECT NombreCia, Ciudad
FROM PROVEEDORES
WHERE Pais = 'Alemania'
UNION
SELECT NombreCia, Ciudad
FROM CLIENTES
WHERE Pais = 'Alemania';
```

INTERSECCIÓN: Una academia de idiomas da clases de inglés, frances y portugues; almacena los datos de los alumnos en tres tablas distintas una llamada "ingles", en una tabla denominada "frances" y los que aprenden portugues en la tabla "portugues". La academia necesita el nombre y domicilio de todos los alumnos que cursan los tres idiomas para enviarles iformación sobre los exámenes.

```
SELECT nombre, domicilio
FROM ingles
INTERSECT
SELECT nombre, domicilio
FROM frances
INTERSECT
SELECT nombre, domicilio
FROM portugues;
```

DIFERENCIA: Ahora la academia necesita el nombre y domicilio solo de todos los alumnos que cursan inglés (no quiere a los que ya cursan portugués pues va a enviar publicidad referente al curso de portugués).

```
SELECT nombre, domicilio
FROM INGLES
MINUS
SELECT nombre,domicilio
FROM PORTUGUES;
```

1.6. Subconsultas

A veces tendrás que utilizar en una consulta los resultados de otra que llamaremos **subconsulta**. La sintaxis es:

```
SELECT listaExpr
FROM tabla
WHERE expresión OPERADOR ( SELECT listaExpr
                           FROM tabla);
```

La subconsulta puede ir dentro de las cláusulas `WHERE`, `HAVING` o `FROM`.

El **OPERADOR** puede ser `>`, `<`, `>=`, `<=`, `!=`, `=` o `IN`. Las subconsultas que se utilizan con estos operadores `devuelven un único valor`, si la subconsulta devolviera más de un valor devolvería un error.

Como puedes ver en la sintaxis, las subconsultas `deben ir entre paréntesis y a la derecha` del operador.

Pongamos un ejemplo:

```
SELECT Nombre_empleado, sueldo
FROM EMPLEADOS
WHERE SUELDO < (SELECT SUELDO
                FROM EMPLEADOS
                WHERE Nombre_emple = 'Ana');
```

Obtendríamos el nombre de los empleados y el sueldo de aquellos que cobran menos que Ana.

Los tipos de datos que devuelve la subconsulta y la columna con la que se compara ha de ser el mismo.

¿Qué hacemos si queremos comparar un valor con varios, es decir, si queremos que la subconsulta devuelva más de un valor y comparar el campo que tenemos con dichos valores? Imagina que queremos ver si el sueldo de un empleado que es administrativo es mayor o igual que el sueldo medio de otros puestos en la empresa. Para saberlo deberíamos calcular el sueldo medio de las demás ocupaciones que tiene la empresa y éstos compararlos con la de nuestro empleado. Como ves, el resultado de la subconsulta es más de una fila. ¿Qué hacemos?

Cuando el resultado de la subconsulta es más de una fila, SQL utiliza **instrucciones especiales** entre el operador y la consulta. Estas instrucciones son:

- `ANY`. Compara con cualquier fila de la consulta. La instrucción es válida si hay un registro en la subconsulta que permite que la comparación sea cierta.
- `ALL`. Compara con todas las filas de la consulta. La instrucción resultará cierta si es cierta toda la comparación con los registros de la subconsulta.
- `IN`. No utiliza comparador, lo que hace es comprobar si el valor se encuentra en el resultado de la subconsulta.
- `NOT IN`. Comprueba si un valor no se encuentra en una subconsulta.

En la siguiente consulta obtenemos el empleado que menos cobra:

```
SELECT nombre, sueldo
FROM EMPLEADOS
WHERE sueldo <= ALL (SELECT sueldo
                     FROM EMPLEADOS);
```