BASES DE DATOS CON ORACLE (+ SQL DEVELOPER) - II

Nicolás TrescolíBlasco

IES Henri Matisse

ÍNDICE

Recordatorio Oracle :

1. Entrar como system, crear un tablespace para las bases de datos y seleccionarlo por

defecto

CREATE TABLESPACE nba DATAFILE 'nba.dbf' SIZE 10M;

ALTER DATABASE DEFAULT TABLESPACE nba;

2. Crear un usuario

CREATE USER nba IDENTIFIED BY nbapwd;

3. Otorgar permisos al usuario para que pueda crear tablas y hacer consultas,

GRANT CREATE SESION, CREATE TABLE, SELECT ANY TABLE, UNLIMITED TABLESPACE TO nba;

4. Salir y volver a entrar con el usuario nba.

5. Ejecutar script de creación de tablas.

@nba.sql

indices → solo guarda las relaciones de claves primarias para agilizar consultas

CREATE TABLESPACE prueba DATAFILE 'prueba.dbf' SIZE 10M;

CREATE TEMPORARY TABLESPACE prueba_temp TEMPFILE 'prueba_temp.dbf' SIZE 200 M;

CREATE TABLESPACE prueba_index DATAFILE 'prueba_index.dbf' SIZE 200M;

SELECT * FROM DBA_DATA_FILES;

CONSULTAS II

Nombre de Columna Personalizado (AS)

A la hora de realizar consultas de cualquier tipo, los nombres de las columnas mostradas a menudo son palabras del vocabulario del programador, no muy intuitivas para el lector promedio. Es por eso que cuando se requiere presentar un documento que muestra una tabla de consulta, es una práctica común cambiar el nombre de las columnas en el momento de realizar la consulta. Para ello existe la instrucción **AS**.

SELECT *campo1* **AS** *nombre1, campo2* **AS** *nombre2 ….* **FROM** *nombre_tabla …. ;*

Dada la tabla actores:

ACTORES

dni	nombre	fecha_nac	teléfono	premios	ciudad
20969932j	Carmen	04/06/1994	767472456	4	Madrid
21349733p	Natalia	23/09/1995	555555555	6	Valencia
29879837k	Marcos	15/02/2003	876962983	1	Valencia
98769823y	Pedro	06/06/1998	675456252	8	Barcelona

Ejemplo:

SELECT *nombre, premios* **AS** *galardones, ciudad* **FROM** *actores ;*

Resultado:

ACTORES

nombre	galardones	ciudad
Carmen	4	Madrid
Natalia	6	Valencia
Marcos	1	Valencia
Pedro	8	Barcelona

(El nombre de la columna “premios” ha cambiado a “galardones” para esta consulta).

Filtros con test de patrón (% y _)

Los filtros con test patrón seleccionan los registros en que determinados campos contienen una sucesión de caracteres concreta. La sintaxis queda así:

SELECT *campos* **FROM** *nombre_tabla* **WHERE** *nombre_campo* **LIKE** *'ejemplo';*

Se utilizan los caracteres comodines **%** y **_** para buscar una cadena de caracteres de la cual desconocemos la palabra completa.

- El carácter **%** busca coincidencias de cualquier número de caracteres, incluso cero.
- El carácter **_** busca coincidencias de exactamente un carácter.

La mejor forma de ilustrarlo es con ejemplos:

<table>
<tr><th>Ejemplo</th><th>Sintaxis</th><th>Resultado</th></tr>
<tr><td>Mostrar los registros cuyo valor para el campo 'ciudad' empiece por 'Bar'.</td><td>SELECT nombre, ciudad FROM actores WHERE ciudad LIKE 'Bar%';</td><td>ACTORES<table><tr><th>nombre</th><th>ciudad</th></tr><tr><td>Pedro</td><td>Barcelona</td></tr></table></td></tr>
<tr><td>Mostrar los registros cuyo valor para el campo 'ciudad' acaba en 'ona'.</td><td>SELECT nombre, ciudad FROM actores WHERE ciudad LIKE '%ona';</td><td>ACTORES<table><tr><th>nombre</th><th>ciudad</th></tr><tr><td>Pedro</td><td>Barcelona</td></tr></table></td></tr>
<tr><td>Mostrar los registros cuyo valor para el campo 'ciudad' contenga 'rcelo'.</td><td>SELECT nombre, ciudad FROM actores WHERE ciudad LIKE '%rcelo%';</td><td>ACTORES<table><tr><th>nombre</th><th>ciudad</th></tr><tr><td>Pedro</td><td>Barcelona</td></tr></table></td></tr>
<tr><td>Mostrar los registros cuyo valor para el campo 'ciudad' empiece por 'Ma', acabe en 'd' y tenga 5 caracteres.</td><td>SELECT nombre, ciudad FROM actores WHERE ciudad LIKE 'Ma___d';</td><td>ACTORES<table><tr><th>nombre</th><th>ciudad</th></tr><tr><td>Carmen</td><td>Madrid</td></tr></table></td></tr>
<tr><td>Mostrar los registros cuyo valor para el campo 'ciudad' tenga como penúltimo carácter 'i'</td><td>SELECT nombre, ciudad FROM actores WHERE ciudad LIKE '%i_';</td><td>ACTORES<table><tr><th>nombre</th><th>ciudad</th></tr><tr><td>Carmen</td><td>Madrid</td></tr><tr><td>Natalia</td><td>Valencia</td></tr><tr><td>Marcos</td><td>Valencia</td></tr></table></td></tr>
</table>

AGRUPAR (GROUP BY)

Utilizamos GROUP BY cuando queremos mostrar los resultados de una consulta agrupados según algún criterio.

SELECT *campo1* **FROM** *nombre_tabla* **WHERE** *condiciones* **GROUP BY** *campo1;*

Ejemplo.:

SELECT *ciudad* **FROM** *actores* **WHERE** *premios<7* **GROUP BY** *ciudad;*

<u>Resultado:</u>

ACTORES

ciudad
Madrid
Valencia

(No se muestra el valor de campo 'Barcelona' porque el actor correspondiente tiene más de 7 premios).

(La ciudad 'Valencia' SOLO se muestra una vez porque hemos agrupado los resultados de la consulta por ciudades).

(Nótese que si hubiéramos decidido mostrar algún campo como 'nombre' en lugar de 'ciudad' la consulta hubiera fallado, ya que habría que mostrar el nombre de los actores residentes en Valencia, que son dos, a la vez agrupar en un solo registro aquellos que vivan en Valencia, lo cual es una contradicción).

HAVING

Es posible además, añadir filtros a los grupos que creamos. Los filtros de grupos deben realizarse mediante el uso de la cláusula HAVING puesto que WHERE actúa antes de agrupar los registros. Lo escribimos así:

SELECT *campos* **FROM** *nombre_tabla* **WHERE** *condiciones* **GROUP BY** *nombre_campo* **HAVING** *condiciones_grupo;*

Ejemplo.:

SELECT *ciudad* **FROM** *actores* **WHERE** *premios<7* **GROUP BY** *ciudad* **HAVING** *ciudad* **LIKE** *'%ia';*

<u>Resultado:</u>

ACTORES

ciudad
Valencia

(La consulta es idéntica a la del ejemplo anterior, pero se han excluido los grupos que no acaban en 'ia', por tanto solo se muestra el registro de 'Valencia').

ORDENAR (ORDER BY)

Para mostrar ordenados un conjunto de registros utilizamos la cláusula ORDER BY de la sentencia SELECT.

SELECT *campos* **FROM** *nombre_tabla* **WHERE** *condiciones* **ORDER BY** *nombre_campo;*

Por defecto, la ordenación se realiza en orden ascendente, pero podemos modificarlo con las cláusulas **DESC** (descendente) y **ASC** (ascendente).

Ejemplo.:

SELECT * FROM *actores* **WHERE** *fecha_nac > '01/01/1990'* **ORDER BY** *premios* ***DESC;***

<u>Resultado:</u>

ACTORES

dni	nombre	fecha_nac	teléfono	premios	ciudad
98769823y	Pedro	06/06/1998	675456252	8	Barcelona
21349733p	Natalia	23/09/1995	555555555	6	Valencia

20969932j	Carmen	04/06/1994	767472456	6	Madrid
29879837k	Marcos	15/02/2003	876962983	1	Valencia

(Se muestran los registros ordenados según la cantidad de premios)

Es posible usar en la misma sentencia varios criterios de ordenación pudiendo mezclar incluso ordenaciones ascendentes y descendentes.

Ejemplo.:

SELECT * FROM *actores* **WHERE** *fecha_nac > '01/01/1990'* **ORDER BY** *premios* ***DESC,*** *nombre* ***ASC****;*

Resultado:

ACTORES

dni	nombre	fecha_nac	teléfono	premios	ciudad
98769823y	Pedro	06/06/1998	675456252	8	Barcelona
21349733p	Natalia	23/09/1995	555555555	6	Valencia
20969932j	Carmen	04/06/1994	767472456	6	Madrid
29879837k	Marcos	15/02/2003	876962983	1	Valencia

(Esta opción puede resumirse como: "Ordena los actores por número de premios, y si hay algún empate, ordena esos actores por su nombre")

La cláusula GROUP BY puede aparecer junto a ORDER BY de modo que la consulta de grupos puede a la vez ordenarse de forma ascendente (ASC) o descendente (DESC).

Ejemplo:

SELECT *campo1* **FROM** *nombre_tabla* **WHERE** *condiciones* **GROUP BY** *campo1* **HAVING** *condiciones_grupo* **ORDER BY** *campo1;*

SUBCONSULTAS

Las subconsultas se utilizan para realizar filtrados con los datos de otra consulta. Estos filtros pueden ser aplicados tanto en la cláusula WHERE para filtrar registros como en la cláusula HAVING para filtrar grupos.

SELECT *campos* FROM *tabla_1* WHERE *campo* → **(CONSULTA 2)**

Escribimos la segunda consulta entre paréntesis como “condición” de la primera consulta.

Algunos casos:

SELECT *campos* FROM *tabla_1* WHERE *campo*=**(SELECT *campo* FROM *tabla_2* WHERE *condiciones*);**

SELECT *campos* FROM *tabla_1* WHERE *campo* **BETWEEN (SELECT *campo* FROM *tabla_2* WHERE *condiciones*) AND (SELECT *campo* FROM *tabla_3* WHERE *condiciones*);**

SELECT *campos* FROM *tabla_1* WHERE *campo* **IN (SELECT *campo* FROM *tabla_2* WHERE *condiciones*);**

. . . . .

Ejemplo:

SELECT dni, nombre, ciudad FROM actores WHERE ciudad IN
(SELECT nombre FROM ciudades WHERE habitantes(millones)>=1,5);

Resultado:

ACTORES

dni	nombre	ciudad
20969932j	Carmen	Madrid
98769823y	Pedro	Barcelona

CIUDADES

cod_ciudad	nombre	habitantes(millones)
001	Madrid	3,223
002	Barcelona	1,62
003	Valencia	0,8

(Se muestran los registros de la tabla actores cuya ciudad tiene almenos 1,5 millones de habitantes)

ATENCIÓN: La subconsulta (consulta 2) debe producir como resultado un único valor.

Darían error los siguientes comandos:

SELECT nombre FROM jugadores WHERE altura ***= (SELECT max(altura), max(peso) FROM jugadores);***

SELECT nombre FROM jugadores WHERE altura ***= (SELECT max(altura) FROM jugadores GROUP BY Nombre_Equipo);***

En el primer caso de la subconsulta se obtienen dos resultados: altura máxima y peso máximo. En el segundo caso se obtienen tantos resultados como equipos haya.

El único caso en que es válido que la segunda consulta devuelva varios resultados es al utilizar la cláusula **IN**:

SELECT nombre FROM jugadores WHERE Nombre_equipo **IN (SELECT Nombre FROM equipos WHERE division='SouthWest');**

Subconsultas Anidadas

Es posible anidar subconsultas para realizar filtros de unas consultas sobre otras.

```
SELECT ciudad FROM equipos WHERE nombre =
(SELECT nombre_equipo FROM jugadores WHERE altura =
(SELECT MAX(altura) FROM jugadores))
```

Test Cuantificado (ALL y ANY)

Utilizamos los operadores ALL y ANY para extraer datos de una subconsulta que devuelve múltiples resultados.

ALL
La cláusula ALL relaciona la consulta con TODOS los resultados de la subconsulta.

SELECT *campos* **FROM** *nombre_tabla* **WHERE** *campo_n* = **ALL (CONSULTA 2);**

Ejemplo:

Dadas las tablas profesores y estudiantes:

profesores

nombre	edad
Carmen	43
Pedro	52
Natalia	51
Marcos	35
Eva	40

estudiantes

nombre	edad
Sergio	20
Abel	21
Maria	19
Lucia	37
Tomás	57

Mostrar los estudiantes que son mayores que todos los profesores:

SELECT * **FROM** *estudiantes* **WHERE** *edad* > **ALL (SELECT** *edad* **FROM** *profesores***);**

Resultado:

nombre	edad
Tomás	57

(Tomás es el único alumno mayor que todos los profesores [57 > 43, 52, 51, 35, 40])

ANY

La cláusula ANY relaciona la consulta con ALGUNOS de los resultados de la subconsulta.

SELECT *campos* **FROM** *nombre_tabla* **WHERE** *campo_n =* **ANY (CONSULTA 2);**

Ejemplo:

Dadas las tablas profesores y estudiantes del ejemplo anterior.

Mostrar los estudiantes que son mayores que alguno de los profesores:

SELECT * **FROM** *estudiantes* **WHERE** *edad >* **ANY (SELECT** *edad* **FROM** *profesores***);**

Resultado:

nombre	edad
Lucia	37
Tomás	57

*(Tomás es mayor que todos los profesores, pero Lucia también es mayor que uno de los profesores, Marcos [**57** > 43, 52, 51, 35, 40] y [**37** > 35, aunque menor que el resto])*

Test de existencia (EXISTS)

El test de existencia, **EXISTS**, permite filtrar los resultados de una consulta si existen filas en la subconsulta asociada, esto es, si la subconsulta genera un número de filas distinto de 0.

También existe su antónimo, **NOT EXISTS**.

SELECT *campos* **FROM** *nombre_tabla* **WHERE EXISTS (CONSULTA 2)***;*

Ejemplo:

Mostrar los equipos que no tengan jugadores españoles.

SELECT *nombre* **FROM** *equipos* **WHERE NOT EXISTS (SELECT** *nombre* **FROM** *jugadores* **WHERE** *equipos.nombre = jugadores.nombre_equipo* **AND** *procedencia='Spain'***)***;*

En el ejemplo anterior, en la subconsulta, se están seleccionando los jugadores con procedencia 'Spain'.
Luego, en la consulta externa, se utiliza NOT EXISTS para seleccionar los equipos que no incluyan ninguno de los jugadores seleccionados en la subconsulta.

CONSULTAS MULTITABLA

Una consulta multitabla es aquella en la que se puede consultar información de más de una tabla aprovechando los campos relacionados de las tablas para unirlas.

La realización de consultas multitabla se puede realizar de manera implícita y explícita (dos formas de escribir la consulta, pero con el mismo resultado). Primero se mostrarán las formas implícitas (más intuitivas), y como complemento, se mostrarán las formas explícitas.

Producto Cartesiano

El producto cartesiano muestra todas las combinaciones de las filas de una tabla unidas a las filas de la otra tabla.

SELECT * **FROM** *tabla1, tabla2, …* **;**

Para mostrar solo algunos campos indicamos el nombre de la tabla, un punto " **.** ", y el campo de esa tabla que queremos mostrar. Tras el **FROM**, indicamos las tablas implicadas.

SELECT *tabla1.campo1, tabla1.campo2, … , tabla2.campo1***,** *tabla2.campo2, …*
FROM *tabla1, tabla2, …* **;**

Ejemplo: Dadas las tablas actores y ciudades.

ACTORES

dni	nombre	premios	ciudad
20969932j	Carmen	4	Madrid
21349733p	Natalia	6	Valencia
98769823y	Pedro	8	Barcelona

CIUDADES

cod_ciudad	nombre	habitantes
001	Madrid	3,223
002	Barcelona	1,62
003	Valencia	0,8

Si ahora aplicamos un producto cartesiano, se muestran todas las combinaciones entre los registros de la tabla actores y la tabla ciudades.

SELECT *actores.dni, actores.nombre, actores.ciudad, ciudades.nombre, ciudades.habitantes*
FROM *actores, ciudades*;

Resultado:

dni	nombre	ciudad	nombre	habitantes
20969932j	Carmen	Madrid	Madrid	3,223
20969932j	Carmen	Madrid	Barcelona	1,62
20969932j	Carmen	Madrid	Valencia	0,8
21349733p	Natalia	Valencia	Madrid	3,223
21349733p	Natalia	Valencia	Barcelona	1,62
21349733p	Natalia	Valencia	Valencia	0,8
98769823y	Pedro	Barcelona	Madrid	3,223
98769823y	Pedro	Barcelona	Barcelona	1,62
98769823y	Pedro	Barcelona	Valencia	0,8

(Se muestran 9 filas en total: por cada uno de los 3 actores se muestran las 3 filas de la tabla ciudades)

Forma explícita

Un producto cartesiano se puede expresar de forma explícita con la siguiente sintaxis:

SELECT *tabla1.campo1, tabla1.campo2, … , tabla2.campo1*, *tabla2.campo2, …*
FROM *tabla1* **CROSS JOIN** *tabla2, …* ;

MUCHO OJO!!!

Hay que fijarse en que se escribe: tabla.campo, *(actores.ciudad) y no al revés,* ~~**campo.tabla**~~

INNER JOIN

En el caso del producto cartesiano, se mostraban todas las combinaciones posibles.

Sin embargo, si solo queremos mostrar las filas con algún tipo de coincidencia entre una tabla y otra (lo más habitual) podemos utilizar el filtro **WHERE** de forma especial: Relacionando la Clave Ajena de una tabla con la Clave Principal de la otra.

SELECT * **FROM** *tabla1, tabla2, …* **WHERE** *tabla2.ClaveAjena = tabla1.ClavePrimaria* **;**

Dadas las tablas del ejemplo anterior, podemos solucionar el problema de los grupos de repetición añadiendo una cláusula **WHERE** aprovechando las Claves Ajenas y Claves Primarias por las que se relacionan.

SELECT *actores.dni, actores.nombre, actores.ciudad, ciudades.nombre, ciudades.habitantes*
FROM *actores, ciudades* **WHERE** *actores.ciudad = ciudades.nombre***;**

Resultado:

dni	nombre	ciudad	nombre	habitantes
20969932j	Carmen	Madrid	Madrid	3,223
21349733p	Natalia	Valencia	Valencia	0,8
98769823y	Pedro	Barcelona	Barcelona	1,62

(Esta vez se muestran solo 3 filas en total: ya que la cláusula where relaciona cada actor solamente con la ciudad a la que pertenece)

Forma explícita

Un **INER JOIN** (o simplemente **JOIN**) se puede expresar de forma explícita con esta sintaxis:

SELECT *tabla1.campo1, tabla1.campo2, … , tabla2.campo1***,** *tabla2.campo2, …*
FROM *tabla1* **INNER JOIN** *tabla2, …*
ON *tabla2.ClaveAjena = tabla1.ClavePrimaria* **;**

ALIAS

A la hora de realizar consultas que impliquen más de una tabla, resulta práctico no tener que repetir el nombre de cada tabla cuando se necesite hacer referencia a ella. Es por ello que se pueden utilzar Alias en la sentencia FROM para hacer más cómoda la consulta.

Ejemplo:

SELECT *actores.dni, actores.nombre, actores.ciudad, ciudades.nombre, ciudades.habitantes*
FROM *actores, ciudades* **WHERE** *actores.ciudad = ciudades.nombre***;**

La consulta anterior podría escribirse de la siguiente forma:

Ejemplo:

SELECT *A.dni, A.nombre, A.ciudad, C.nombre, C.habitantes*
FROM *actores A, ciudades C* **WHERE** *A.ciudad = C.nombre***;**

Como puede observarse, escribiendo un alias escogido por nosotros al lado del nombre de la tabla en la sentencia FROM, podemos hacer referencia a campos de la misma sin tener que utilizar el nombre completo de la tabla en cuestión.

OUTER JOIN: LEFT, RIGHT y FULL JOIN

Los OUTER JOIN se utilizan para realizar consultas multitabla en que NO TODOS los registros de una tabla tienen correspondencia con algún registro de la otra tabla.

Para ilustrar qué resultados devuelven estos estilos de JOIN, se utilizarán las siguientes tablas a modo de ejemplo:

ACTORES

dni	nombre	premios	ciudad
20969932j	Carmen	4	Madrid
21349733p	Natalia	6	Valencia
98769823y	Pedro	8	Barcelona
29879837k	Marcos	1	NULL

CIUDADES

cod_ciudad	nombre	habitantes
001	Madrid	3,223
002	Barcelona	1,62
003	Valencia	0,8
004	Sevilla	0,6

Nótese que la tabla actores tiene un registro para el que el campo ‘ciudad’ es NULL, y que no existe ningún actor que resida en Sevilla pese a que aparece como registro en la tabla ciudades.

LEFT JOIN → ... = (+)

Utilizamos **LEFT JOIN** o **LEFT OUTER JOIN** cuando queremos mostrar TODOS los registros de la primera tabla y solamente aquellos de la segunda tabla que tengan alguna coincidencia con los de la primera, de la siguiente forma:

SELECT *tabla1.campo1, tabla1.campo2, ... , tabla2.campo1*, *tabla2.campo2, ...*
FROM *tabla1, tabla2, ...*
WHERE *tabla2.ClaveAjena = tabla1.ClavePrimaria* **(+) ;**

Si realizamos ahora la consulta:

SELECT *actores.dni, actores.nombre, actores.ciudad, ciudades.nombre, ciudades.habitantes*
FROM *actores, ciudades* **WHERE** *actores.ciudad = ciudades.nombre* **(+) ;**

Resultado:

dni	nombre	ciudad	nombre	habitantes
20969932j	Carmen	Madrid	Madrid	3,223
21349733p	Natalia	Valencia	Valencia	0,8
98769823y	Pedro	Barcelona	Barcelona	1,62
29879837k	Marcos	**NULL**	**NULL**	**NULL**

(Se muestran todos los registros de la primera tabla [actores] y los registros correspondientes de la segunda tabla [ciudades], incluso para los que no tienen asignada una ciudad [Se muestra NULL, en los campos en cuestión])

Forma explícita

Un **LEFT JOIN** se puede expresar de forma explícita con esta sintaxis:

SELECT *tabla1.campo1, tabla1.campo2, ... , tabla2.campo1*, *tabla2.campo2, ...*
FROM *tabla1* **LEFT JOIN** *tabla2, ...*
ON *tabla2.ClaveAjena = tabla1.ClavePrimaria* **;**

RIGHT JOIN → (+) = …

Utilizamos **RIGHT JOIN** o **RIGHT OUTER JOIN** con el mismo objetivo que LEFT JOIN pero para referirnos a la segunda tabla:

SELECT *tabla1.campo1, tabla1.campo2, … , tabla2.campo1*, *tabla2.campo2, …*
FROM *tabla1, tabla2, …*
WHERE *tabla2.ClaveAjena* **(+)** = *tabla1.ClavePrimaria*;

SELECT *actores.dni, actores.nombre, actores.ciudad, ciudades.nombre, ciudades.habitantes*
FROM *actores, ciudades* **WHERE** *actores.ciudad* **(+)** = *ciudades.nombre*;

Resultado:

dni	nombre	ciudad	nombre	habitantes
20969932j	Carmen	Madrid	Madrid	3,223
21349733p	Natalia	Valencia	Valencia	0,8
98769823y	Pedro	Barcelona	Barcelona	1,62
NULL	**NULL**	**NULL**	Sevilla	0,6

(Se muestran todos los registros de la segunda tabla [ciudades] y los registros correspondiente de la primera tabla[actores], incluso para las ciudades que no tienen actores asociados [Se muestra NULL, en los campos en cuestión])

Forma explícita

Un **RIGHT JOIN** se puede expresar de forma explícita con esta sintaxis:

SELECT *tabla1.campo1, tabla1.campo2, … , tabla2.campo1*, *tabla2.campo2, …*
FROM *tabla1* **RIGHT JOIN** *tabla2, …*
ON *tabla2.ClaveAjena = tabla1.ClavePrimaria* ;

FULL JOIN → (+) = (+)

Utilizamos **FULL JOIN** o **FULL OUTER JOIN** para mostrar todos los registros de ambas tablas tengan o no coincidencia en alguno de los campos.

SELECT *tabla1.campo1, tabla1.campo2, … , tabla2.campo1, tabla2.campo2, …*
FROM *tabla1, tabla2, …*
WHERE *tabla2.ClaveAjena* **(+)** *= tabla1.ClavePrimaria* **(+) ;**

Si realizamos ahora la consulta:

SELECT *actores.dni, actores.nombre, actores.ciudad, ciudades.nombre, ciudades.habitantes*
FROM *actores, ciudades* **WHERE** *actores.ciudad* **(+)** *= ciudades.nombre***;**

Resultado:

dni	nombre	ciudad	nombre	habitantes
20969932j	Carmen	Madrid	Madrid	3,223
21349733p	Natalia	Valencia	Valencia	0,8
98769823y	Pedro	Barcelona	Barcelona	1,62
29879837k	Marcos	**NULL**	**NULL**	**NULL**
NULL	**NULL**	**NULL**	Sevilla	0,6

(Se muestran todos los registros de ambas tablas incluso para los que no tienen un registro correspondiente en la otra tabla [Se muestra NULL, en los campos en cuestión])

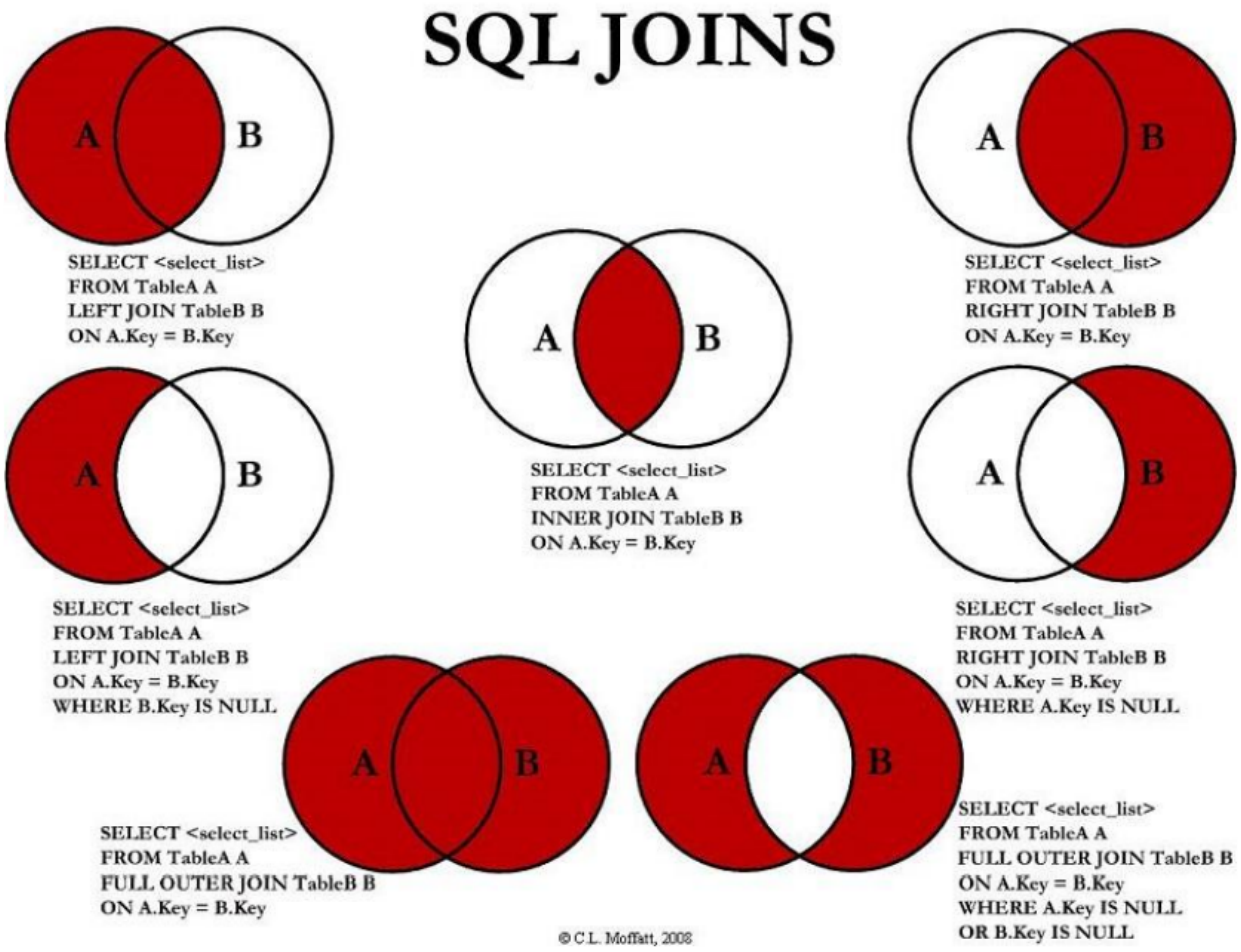
Forma explícita

Un **FULL JOIN** se puede expresar de forma explícita con esta sintaxis:

SELECT *tabla1.campo1, tabla1.campo2, … , tabla2.campo1, tabla2.campo2, …*
FROM *tabla1* **FULL JOIN** *tabla2, …*
ON *tabla2.ClaveAjena = tabla1.ClavePrimaria* **;**

Otras consultas multitabla

Además de las consultas multitabla explicadas, existen otros tipos de consultas no tan utilizadas, pero que se resumen gráficamente en el siguiente esquema:



Por supuesto, es posible hacer una consulta multitabla de más de 2 tablas a la vez, hacer subconsultas y también es posible agrupar y ordenar.

Ejemplo: Número de etapas que ha ganado cada ciclista y cual es su director:

SELECT count(etapa.netapa), ciclista.nombre, equipo.director
FROM ciclista, etapa, equipo
WHERE ciclista.dorsal = etapa.dorsal **and** ciclista.nomeq=equipo.nomeq
GROUP BY ciclista.nombre, equipo.director
ORDER BY ciclista.nombre

CONSULTAS REFLEXIVAS

A veces, es necesario obtener información de relaciones reflexivas, por ejemplo, informe de empleados donde junto a su nombre y apellidos apareciera el nombre apellidos de su jefe. Para ello, es necesario hacer una JOIN entre registros de la misma tabla.

Ejemplo:

~~**SELECT** emp.Nombre as Empleado, emp.Nombre as Jefe **FROM** Empleados emp~~

Sin embargo, la consulta anterior presenta ambigüedades y no muestra los resultados que debería. Para solucionar este problema necesitamos hacer uso de 2 alias para la misma tabla:

Ejemplo:

SELECT emp.Nombre as Empleado, jefe.Nombre as Jefe
FROM Empleados emp, Empleados jefe
WHERE emp.CodigoEmpleado=jefe.CodigoJefe;

En esta consulta, ahora sí, correcta, la tabla Empleados tiene 2 alias (emp y jefe), pudiendo tratarlas como dos tablas distintas y así trabajar la consulta con normalidad.

MULTICONSULTAS

Es posible mostrar los resultados de más de una consulta a la vez dentro de una misma sentencia. En función de la información que debamos mostrar podemos hacer uso de las sentencias UNION, INTERSECT y MINUS/EXCEPT.

UNION (Unión)

Para mostrar los resultados completos de dos o más consultas en una única tabla de resultados totales. Utilizamos las sentencia “UNION”.

SELECT *CONSULTA_1* **UNION SELECT** *CONSULTA_2 … ;*

Ejemplo:

Seleccionar los actores que vivan en Madrid o tengan más de 5 premios

SELECT *nombre, premios, ciudad* **FROM** *actores* **WHERE** ciudad=’Madrid’ **UNION**
SELECT *nombre, premios, ciudad* **FROM** *actores* **WHERE** premios>5*;*

Resultado:

ACTORES

nombre	premios	ciudad
Carmen	4	Madrid
Natalia	6	Valencia
Pedro	8	Barcelona

Para poder realizar una unión existen varias restricciones:

- Ambas consultas deben devolver el mismo número de columnas.
- El tipo de datos de cada columna en la primera tabla debe ser el mismo que el tipo de datos en la columna correspondiente en la segunda tabla.
- Ninguna de las dos tablas puede estar ordenadas con la cláusula ORDER BY (pero sí el resultado final de la consulta).

UNION elimina por defecto las filas duplicadas. Para mantener las filas duplicadas en una operación UNION, se puede especificar la palabra clave **ALL** a continuación de UNION.

SELECT *CONSULTA_1* **UNION ALL SELECT** *CONSULTA_2 … ;*

INTERSECT (Intersección)

Utilizamos la intersección para mostrar solamente un conjunto de resultados comunes a ambas consultas.

SELECT *CONSULTA_1* **INTERSECT SELECT** *CONSULTA_2 … ;*

Ejemplo:

Obtener el listado de equipos que han jugado como locales y visitantes

SELECT equipo_local FROM partidos **INTERSECT SELECT** equipo_visitante FROM partidos;

MINUS (Diferencia)

Utilizamos la intersección para mostrar solamente un conjunto de resultados comunes a ambas consultas.

SELECT *CONSULTA_1* **MINUS SELECT** *CONSULTA_2 … ;*

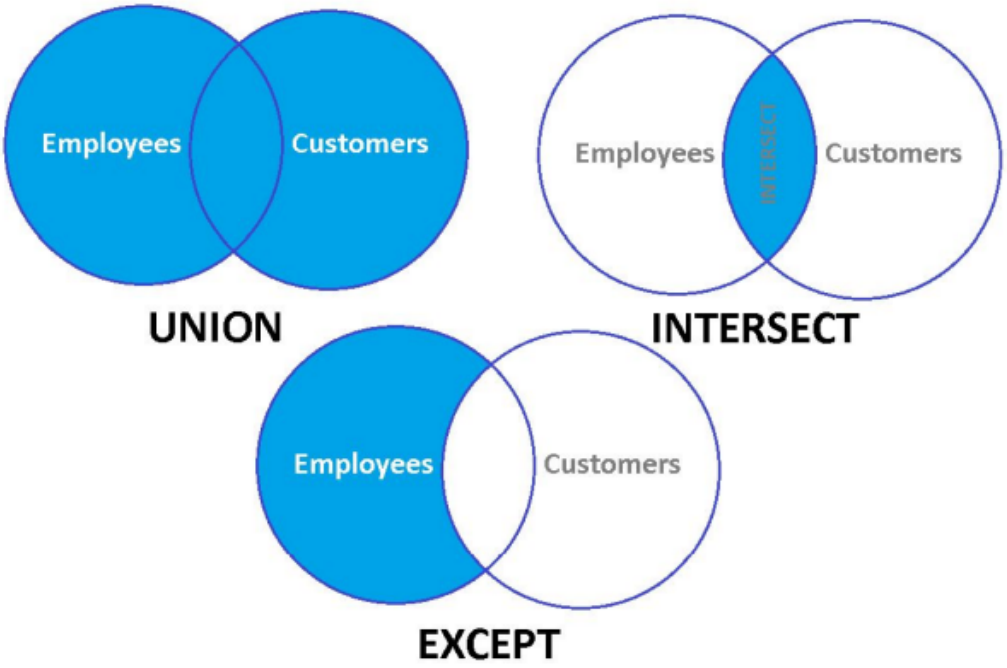
Ejemplo:

Obtener los jugadores cuyo apellido es ‘Smith’ que no pertenezcan a los Cavaliers.

SELECT nombre FROM jugadores WHERE nombre like '% Smith' **MINUS**
SELECT nombre FROM jugadores WHERE nombre_equipo='Cavaliers'

NOTA: En otros gestores de bases de datos la cláusula se llama “**EXCEPT**”.

A modo de esquema, los resultados de las operaciones de consultas: UNION, INTERSECT y MINUS/EXCEPT, pueden representarse como se muestra a continuación:



TABLAS DERIVADAS

Las consultas con tablas derivadas, son aquellas que utilizan sentencias SELECT en la cláusula FROM en lugar de nombres de tablas.

SELECT *campos* FROM (**SELECT** *campos* FROM *tabla* WHERE *condiciones*) ;

Ejemplo:

SELECT * FROM (**SELECT** CodigoEmpleado, Nombre FROM Empleados WHERE CodigoOficina='TAL-ES') ;