

Operaciones de unión entre tablas

Operaciones de unión entre tablas	1
¿Qué aprenderás?	2
Introducción	2
Unión de tablas	3
INNER JOIN	5
LEFT JOIN	8
LEFT JOIN Donde una columna en la tabla "B" es null	10
FULL (OUTER) JOIN	12
FULL OUTER JOIN Donde una columna en la tabla "A" o "B" es null	14
Subquery	16
Operador WHERE	17
Operador FROM	19
Observación sobre las subconsultas	20



¡Comencemos!

¿Qué aprenderás?

- Realizar consultas usando el comando JOIN para el consumo de datos entre tablas relacionadas.
- Realizar subconsultas para la anidación de datos entre tablas.

Introducción

Hasta el momento todas las consultas que hemos realizado las hemos hecho sólo sobre una tabla. Previamente explicamos cómo las tablas se relacionan a través de las claves primarias y claves foráneas, pero debemos destacar que el objetivo principal de implementar estas claves en nuestras tablas, es la posibilidad de generar operaciones entre estas entidades y extraer conjuntos de datos concatenados para consultas más personalizadas.

En un plano real, los casos de persistencia de datos requieren de la creación de diferentes entidades (tablas), en las que distribuimos la información correspondiente a una operación ejecutada desde el software y consumir esos datos tabla por tabla sería muy engorroso, conlleva a un costo de cómputo considerable, para esto tenemos el comando JOIN que nos servirá para navegar en los registros de tablas relacionadas y de esta manera lograr con una sola consulta obtener todos los datos relacionados en una situación determinada.

Unión de tablas

Como ya has aprendido en la unidad anterior, podemos relacionar dos tablas por medio de las claves primarias y foráneas, no obstante aún no le hemos sacado provecho a esto y es lo que te enseñaré a continuación.

Veamos el siguiente caso:

“Un sistema con diferentes secciones restringidas por niveles de usuarios, desea almacenar los datos de su personal e identificarlos según su rol en el proyecto, teniendo los roles de administrador, marketing, editor y subscriptor, cada uno con diferentes funciones.”

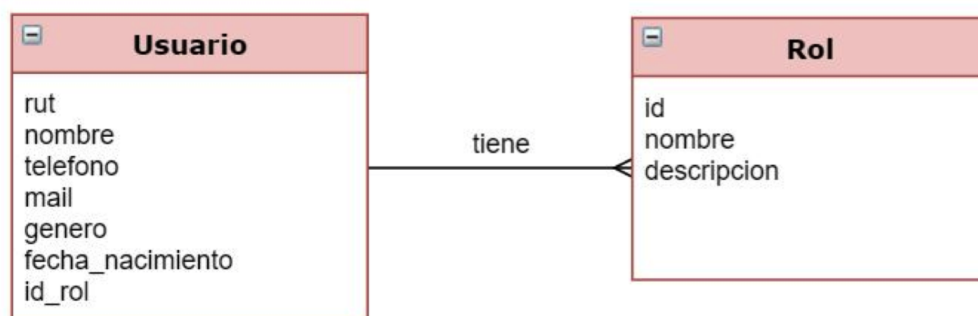


Imagen 1. Tabla Usuario y tabla Rol con llaves primarias y foráneas.

Como puedes ver en la imagen 1 tenemos una tabla de usuarios y una tabla de roles, como podrás intuir la primera tabla tiene un “id_rol” y estos campos dan punta a pie a que funcionan como punto de enlace de más información referente a este “rol”, por lo que debe representarse como una clave foránea, por otro lado tenemos la propia tabla “Rol” que por supuesto registra la información referente a cada rol y servirá como fuente informativa para la tabla Usuario por medio de su campo “id” representando la clave primaria. El caso de uso de este ejemplo será: Un usuario tiene ciertos atributos y puede poseer un rol en nuestro sitio.

La tabla Usuario, posee los siguientes registros:

rut	nombre	telefono	mail	género	fecha_nacimiento	id_rol
1-9	Juan Soto	99999999	juan.soto@gmail.com	M	24-06-2000	1
2-7	Jorge Perez	88888888	jorge.perez@hotmail.com	M	12-03-1998	4
3-5	Sara Morales	77777777	sara.morales@gmail.com	F	11-02-1990	

Tabla 1. Estructura y registros de la tabla Usuario.

Mientras que la tabla Rol posee las siguientes filas:

id	nombre	descripcion
1	administrador	control total
2	marketing	editar y crear anuncios
3	editor	editar artículos
4	subscriber	leer artículos

Tabla 2. Estructura y registros de la tabla Rol.

Anteriormente, hemos visto cómo consultar datos de una tabla, pero en lo cotidiano necesitaremos saber datos de más de una entidad a la vez. ¿Cómo hacemos esto?.

Pensemos, por ejemplo, queremos saber a qué se refiere el rol que posee el usuario con rut 1-9. En la práctica le pediríamos a nuestro modelo algo así como "muéstrame el nombre y la descripción del rol que tenga relación con el usuario de rut 1-9".

En SQL, haríamos lo siguiente:

```
SELECT rol.nombre,  
       rol.descripcion  
FROM   rol  
       JOIN usuario  
       ON rol.id = usuario.id_rol  
WHERE  usuario.id = '1-9';
```

Es decir estaríamos cumpliendo la siguiente sintaxis:

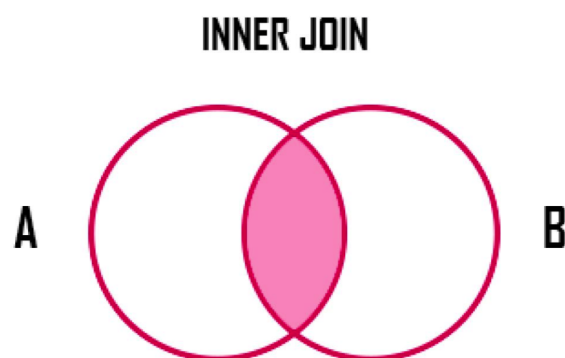
```
-- Seleccionamos las columnas desde la tabla1
SELECT columnas FROM tabla1
-- Posterior a la selección de la columna, indicamos que vamos a generar
la unión
-- con la columna de la tabla2
JOIN tabla2 ON tabla1.columna=tabla2.columna
[WHERE condicion];
```

Esto unirá las columnas indicadas, pero sólo se mostrarán las filas que cumplan la condición dada. Existen distintas formas de combinar filas y se ve traducido en distintos tipos de JOIN que te presento a continuación:

- **INNER JOIN:** Une sólo las columnas comunes entre ambas tablas.
- **LEFT JOIN:** Une todas las columnas de la primera tabla con las columnas en común de la segunda tabla.
- **FULL OUTER JOIN:** Une todas las columnas de ambas tablas.

Veamos ejemplos de consultas usando los distintos tipos de JOIN.

INNER JOIN



```
SELECT columnas
FROM A
INNER JOIN B
ON A.columna=B.columna
```

Imagen 2. Inner join

Observa detenidamente la imagen 2, pues usaré este tipo de gráfico para los siguientes ejemplos. En este caso se hace referencia a este tipo de relación, que nos permite conocer aquellos elementos que tienen en conjunto una o más tablas, como vimos anteriormente en el caso del rol de nuestro usuario Juan Soto.

En nuestro ejemplo si hiciéramos un JOIN entre las tablas usuario y rol obtendremos sólo dos registros. ¿Te imaginas por qué? Ya lo sabrás, por mientras comprobémoslo con el siguiente código:

```
SELECT *
FROM usuario
INNER JOIN rol
ON usuario.id_rol = rol.id;
```

El resultado sería la unión de los datos de ambas tablas, mostrando sólo aquellos datos que tienen en común la columna que utilizamos para relacionarlas y como solo existen 2 usuarios con roles asignados obtendremos la siguiente tabla.

rut	nombre	telefono	mail	genero	fecha_nacimiento	id_rol	id	nombre	descripcion
1-9	Juan Soto	999999999	juan.soto@gmail.com	M	24-06-2000	1	1	administrador	control total
2-7	Jorge Perez	888888888	jorge.perez@hotmail.com	M	12-03-1998	4	4	subscriber	leer articulos

Tabla 3. Inner Join Usuario y Rol.

Tómate unos minutos para volver al pasado y ver cómo encajaría esta nueva herramienta, pensemos en los datos que cargamos en sesiones anteriores sobre los pokemones. Si quisiéramos saber cuáles son los pokemones que ya poseemos y la información detallada, podríamos consultar primero nuestra tabla mis_pokemones y luego consultar el detalle en la tabla pokemones. Para dar contexto, en la siguiente imagen te muestro las tablas tal y como fueron creadas en la unidad pasada e identificando la clave primaria con la siglas “PK” y la clave foránea con “FK”.

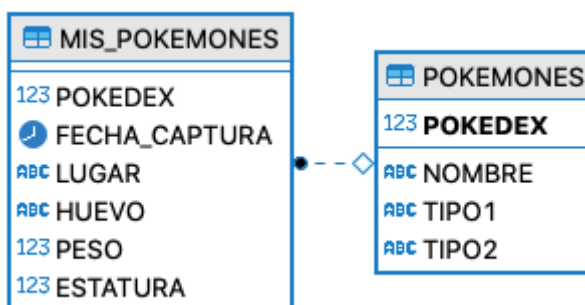


Imagen 3. Tablas creadas en la unidad anterior sobre pokemones.

Entonces INNER JOIN nos permite relacionar varias tablas y obtener aquellos elementos en común a través de una columna que las relacione (generalmente una llave primaria y una foránea), en el caso de los pokemones sería la columna pokedex.

Ejecuta la siguiente consulta, desde cualquier cliente:

```
SELECT *
-- selecciona todos los registros
FROM pokemones
-- de la tabla pokemones
INNER JOIN mis_pokemones
-- relacionada con la tabla mis_pokemones
ON pokemones.pokedex=mis_pokemones.pokedex
-- donde el campo pokedex de la tabla A, sea igual a de la tabla B
ORDER BY nombre;
-- ordenado por nombre
```

Resultado:

POKEDEX	NOMBRE	TIPO1	TIPO2	POKEDEX	FECHA_CAPT	LUGAR	HUEVO	PESO	ESTATURA
63	Abra	psiquico		63	2019-06-01	Huechuraba	N	101	6
63	Abra	psiquico		63	2019-07-19	Huechuraba	N	134	7
144	Articuno	hielo	volador	144	2019-07-13	Independencia	N	554	17
9	Blastoise	agua		9	2019-02-16	Independencia	N	1510	16
9	Blastoise	agua		9	2019-01-26	Independencia	N	1809	18
1	Bulbasaur	planta	veneno	1	2019-01-31	Independencia	N	209	5

... total 199 filas

LEFT JOIN

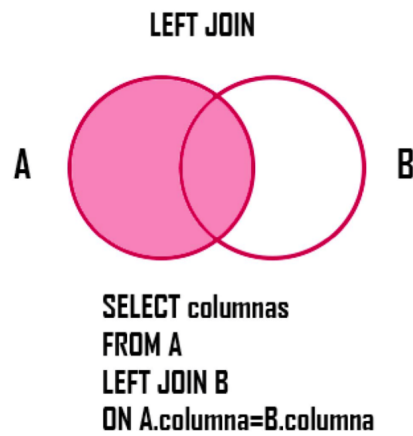


Imagen 4. Left Join.

En la imagen 4 se hace referencia a este tipo de unión, que nos permite obtener toda la información de una tabla y el conjunto de la relación con otra. La diferencia en el código radica en agregarle la palabra “LEFT” antes de “JOIN” .

Ejecuta la siguiente consulta:

```
SELECT *
FROM usuario
LEFT JOIN rol ON usuario.id_rol = rol.id;
```

Y deberás obtener la siguiente tabla:

rut	nombre	telefono	mail	genero	fecha_nacimiento	id_rol	id	nombre	descripcion
1-9	Juan Soto	99999999	juan.soto@gmail.com	M	24-06-2000	1	1	administrador	control total
2-7	Jorge Perez	88888888	jorge.perez@hotmail.com	M	12-03-1998	4	4	subscriber	leer articulos
3-5	Sara Morales	77777777	sara.moreales@gmail.com	F	11-02-1990				

Tabla 4. Left join usuario y rol.

Observando el resultado de los registros obtenidos en la tabla 4, ¿Por qué esta vez nos trajo los datos de Sara?. Eso es porque el LEFT JOIN muestra todos los registros de la tabla 1 (usuario) sin importar que no posean una relación con la segunda tabla y Sara no tenía especificado el "id_rol" en su registro. Por el contrario, aquellos registros de la tabla 2 (rol) que no tengan relación, no los desplegamos.

Veamos otro ejemplo:

¿Qué ocurre si quiero obtener toda la información disponible de los pokemones, sin importar si lo he capturado o no?.

LEFT JOIN nos permite relacionar una o más tablas a través de una columna en común y obtener todos los elementos de la tabla pokemones, y los campos donde existe información de la segunda tabla. Si los registros no existen en la segunda tabla, se mostrará el campo vacío.

Obtengamos todos los pokemones de la siguiente manera:

```
SELECT *  
-- selecciona todos los registros  
FROM pokemones  
-- de la tabla pokemones (tabla izquierda)  
LEFT JOIN mis_pokemones  
-- en relación a la tabla mis_pokemones (tabla derecha)  
on pokemones.pokedex=mis_pokemones.pokedex  
-- relacionados a través de la columna pokedex  
ORDER BY nombre;  
-- ordenados por nombre
```

POKEDEX	NOMBRE	TIPO1	TIPO2	POKEDEX	FECHA_CAPTURA	LUGAR	HUEVO	PESO	ESTATURA
63	Abra	psiquico		63	2019-07-19 00:00:00	Huechuraba	N	134	7
63	Abra	psiquico		63	2019-06-01 00:00:00	Huechuraba	N	101	6
142	Aerodactyl	roca	volador						
65	Alakazam	psiquico							
24	Arbok	veneno							
59	Arcanine	fuego							
144	Articuno	hielo	volador	144	2019-07-13 00:00:00	Independen	N	554	17
15	Beedrill	bicho	veneno						
69	Bellsprout	planta	veneno						
9	Blastoise	agua		9	2019-02-16 00:00:00	Independen	N	1510	16
... truncado									

En este caso podemos ver que a pesar que en mis_pokemones (tabla derecha) no existan Alakazam, Arbok, Arcanite, por ejemplo, nos muestra sus registros de la tabla pokemones (tabla izquierda).

LEFT JOIN Donde una columna en la tabla "B" es null

Este tipo de unión nos permite obtener la información de los registros que no se encuentran en la relación, en la siguiente imagen te muestro esto gráficamente.

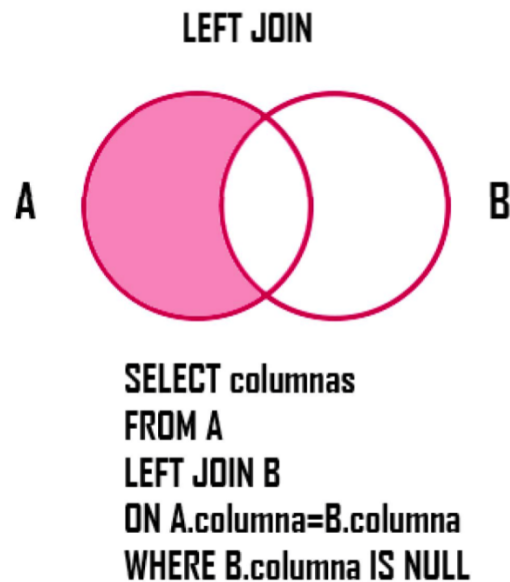


Imagen 5. Left join con columna NULL

Por ejemplo, nos gustaría saber aquellos roles que no han sido asignados a ningún usuario, entonces ejecutamos la siguiente consulta:

```
SELECT rol.*  
FROM rol  
LEFT JOIN usuario ON rol.id = usuario.id_rol  
WHERE usuario.id_rol IS NULL;
```

Y obtendremos la siguiente tabla.

id	nombre	descripcion
2	marketing	editar y crear anuncios
3	editor	editar articulos

Tabla 5. Left join entre rol y usuario con columna NULL.

Nota que en la consulta luego de "SELECT" está escrito "rol.*" es para obtener solamente las columnas correspondientes a la tabla "Rol".

Esto tiene mucha utilidad para obtener información y discriminar registros relevantes para nuestros análisis. Por ejemplo, es posible que nos interese saber qué pokemones aún no tenemos. Para eso usamos un LEFT JOIN con una condición WHERE, de la siguiente manera:

```
SELECT *
-- selecciona todos los registros
FROM pokemones
-- de la tabla pokemones
    LEFT JOIN mis_pokemones
-- en relación a la tabla mis_pokemones
        ON pokemones.pokedex = mis_pokemones.pokedex
WHERE mis_pokemones.estatura IS NULL
-- donde el registro ESTATURA no exista en la tabla mis_pokemones
ORDER BY nombre;
```

Resultado Left join entre pokemones y mis_pokemones con columna NULL.

POKEDEX	NOMBRE	TIPO1	TIPO2	POKEDEX	FECHA_CAPTURA	LUGAR	HUEVO	PESO	ESTATURA
142	Aerodactyl	roca	volador						
65	Alakazam	psiquico							
24	Arbok	veneno							
59	Arcanine	fuego							
15	Beedrill	bicho	veneno						
69	Bellsprout	planta	veneno						
91	Cloyster	agua	hielo						
104	Cubone	tierra							
132	Ditto	normal							
133	Eevee	normal							
83	Farfetchd	normal	volador						

... truncado, 43 filas

Podemos ver que nos muestra los registros que corresponden sólo a aquellos pokemones de la tabla pokemones que no existen en mis_pokemones.estatura.

Ten cuidado con la aplicación de estas consultas porque es común cuando se aprende esto confundirlo con el símbolo "=" al momento de definir la condición, no obstante no aplica en este tipo de consulta por lo que debes usar por sintaxis la palabra "IS".

FULL (OUTER) JOIN

Este tipo de unión nos permite obtener todos los registros disponibles en las tablas que estamos relacionando, sin filtrar la información así como lo puedes inferir al ver la imagen 6.

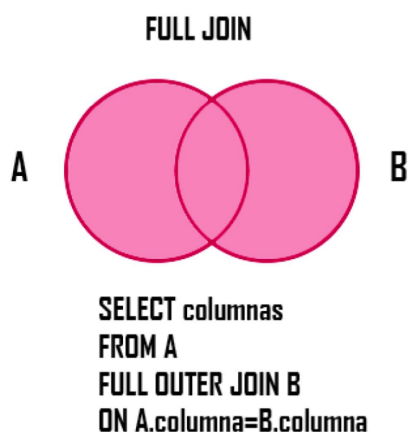


Imagen 6. Full outer join.

Aplicado en el ejemplo de usuarios y roles usamos la siguiente consulta.

```
SELECT *  
FROM usuario  
FULL OUTER JOIN rol  
ON usuario.id_rol=rol.id;
```

Obtenemos el siguiente resultado.

rut	nombre	telefono	mail	genero	fecha_nacimiento	id_rol	id	nombre	descripcion
1-9	Juan Soto	99999999	juan.soto@gmail.com	M	24-06-2000	1	1	administrador	control total
2-7	Jorge Perez	88888888	jorge.perez@hotmail.com	M	12-03-1998	4	4	subscriber	leer articulos
3-5	Sara Morales	77777777	sara.moreales@gmail.com	F	11-02-1990				

							2	marketing	editar y crear anuncios
							3	editor	editar artículos

Tabla 6. Full join entre usuario y rol.

En el caso de los pokemones, podríamos querer saber toda la información disponible, sin importar si poseemos o no el pokemon. Para eso usamos el FULL OUTER JOIN de la siguiente manera:

```
SELECT *
-- selecciona todos los registros
FROM pokemones
-- de la tabla pokemones
FULL OUTER JOIN mis_pokemones
-- en relación con la tabla mis_pokemones
ON pokemones.pokedex = mis_pokemones.pokedex
-- a través de la columna pokedex
ORDER BY nombre;
-- ordenados por nombre
```

Obteniendo el siguiente resultado:

POKEDEX	NOMBRE	TIP01	TIP02	POKEDEX	FECHA_CAPTURA	LUGAR	HUEVO	PESO	ESTATURA
63	Abra	psiquico		63	2019-07-19 00:00:00	Huechuraba	N	134	7
63	Abra	psiquico		63	2019-06-01 00:00:00	Huechuraba	N	101	6
142	Aerodactyl	roca	volador						
65	Alakazam	psiquico							
24	Arbok	veneno							
59	Arcanine	fuego							
144	Articuno	hielo	volador	144	2019-07-13 00:00:00	Independencia	N	554	17
15	Beedrill	bicho	veneno						
69	Bellsprout	planta	veneno						
9	Blastoise	agua		9	2019-02-16 00:00:00	Independencia	N	1510	16

Imagen 7. Full join entre pokemones y mis_pokemones.

Para este caso, nos muestra todos los registros independientemente que no se encuentren en una tabla o la otra.

FULL OUTER JOIN Donde una columna en la tabla "A" o "B" es null

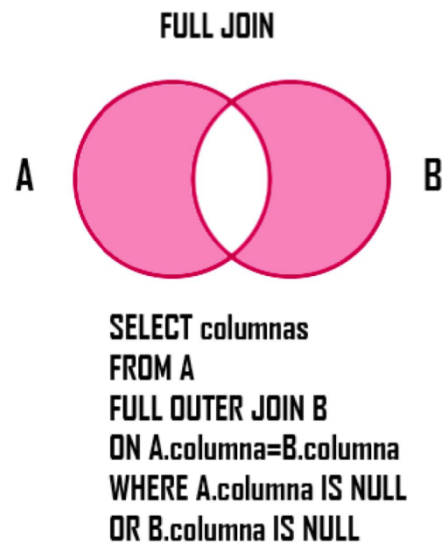


Imagen 8. Full join entre pokemones y mis_pokemones.

Se puede inferir por lo que se muestra en la imagen 8 que FULL OUTER JOIN nos permite conocer aquellos registros que no tienen en común las tablas que estamos relacionando.

Aplicado en el ejemplo de usuarios y roles haríamos la siguiente consulta.

```
SELECT *
FROM usuario
FULL OUTER JOIN rol
ON usuario.id_rol=rol.id
WHERE usuario.id_rol is null OR rol.id is null;
```

Y obtendremos la siguiente tabla.

rut	nombre	telefono	mail	genero	fecha_nacimiento	id_rol	id	nombre	descripcion
3-5	Sara Morales	77777777	sara.moreales@gmail.com	F	11-02-1990				
							2	marketing	editar y crear anuncios
							3	editor	editar articulos

Tabla 7. Full outer join entre usuario y rol.

Y en el ejemplo de los pokemones, la consulta se realizaría de la siguiente manera:

```
SELECT *
-- selecciona todos los registros
FROM pokemones
-- de la tabla pokemones
FULL OUTER JOIN mis_pokemones on
pokemones.pokedex=mis_pokemones.pokedex
-- en relación con la tabla mis_pokemones a través de la columna pokedex
WHERE pokemones.tipo2 is null or mis_pokemones.pokedex is null
-- donde la columna tipo2 es una o la columna pokedex es nula
ORDER BY nombre;
-- ordenados por nombre
```

Para obtener el siguiente resultado, variable en cantidad si es que no ingresamos los mismos datos, pero podemos notar que la columna tipo2 y pokedex, no tienen ambos valores, como indica la instrucción.

POKEDEX	NOMBRE	TIPO1	TIPO2	POKEDEX	FECHA_CAPTURA	LUGAR			
HUEVO	PESO	ESTATURA							
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

6	63	Abra	psiquico		63	2019-06-01 00:00:00	Huechuraba	N	101
7	63	Abra	psiquico		63	2019-07-19 00:00:00	Huechuraba	N	134
	142	Aerodactyl	roca	volador					
	65	Alakazam	psiquico						
	24	Arbok	veneno						
	59	Arcanine	fuego						
	15	Beedrill	bicho	veneno					
	69	Bellsprout	planta	veneno					
18	9	Blastoise	agua		9	2019-01-26 00:00:00	Independencia	N	1809
16	9	Blastoise	agua		9	2019-02-16 00:00:00	Independencia	N	1510
2	10	Caterpie	bicho		10	2019-01-26 00:00:00	Independencia	N	71
5	10	Caterpie	bicho		10	2019-02-13 00:00:00	Cerro Navia	N	101
10	113	Chansey	normal		113	2019-04-20 00:00:00	Independencia	N	388
...									

Subquery

Una subquery (o consulta interna) es una query implementada dentro de otra query principal de SQL, la cual debe enmarcarse dentro de la cláusula WHERE.

Una de las principales aplicaciones de las subqueries, es para retornar datos que serán utilizados posteriormente en una consulta principal, de esta manera funcionan como una condición para restringir los datos. Es decir, las subconsultas son consultas o tablas temporales que sólo estarán para poder complementar el resultado de nuestra consulta principal.

```
SELECT
  *
FROM
  (SELECT nombre FROM POKEMONES);

-- resultado
NOMBRE      |
-----|
Bulbasaur   |
Ivysaur     |
Venusaur    |
Charmander  |
...
```


Operador WHERE

Por ejemplo, queremos obtener como resultado los nombres de los pokemones que fueron obtenidos por huevos.

Ya que en oracle 11g no poseemos el tipo booleano, para que sea verdadero o falso, se basa en si su tipo CHAR es una Y o una N, respectivamente.

Con la siguiente consulta, podemos obtener el número de la pokédex de los pokemones que están en la tabla mis_pokemones; pero no sus nombres.

```
SELECT pokedex
FROM   mis_pokemones
WHERE  huevo = 'Y';
```

-- resultado

```
POKEDEX|
-----|
      7|
     11|
     13|
     16|
     19|
     23|
     33|
     48|
     58|
     62|
     70|
     93|
     99|
    105|
```

Para obtener los nombres, debemos utilizar el operador WHERE de la siguiente manera:

```
SELECT
    columna1,columna2,...
FROM
    nombre_tabla
WHERE
    columna1 IN
        (SELECT columna1 FROM nombre_tabla2 WHERE condicion);
```

Por lo que en nuestro ejemplo la consulta quedaría de la siguiente forma:

```
SELECT pokedex,
       nombre
FROM   pokemones
WHERE  pokedex IN (SELECT pokedex
                    FROM   mis_pokemones
                    WHERE  huevo = 'Y');
```

-- Resultado

POKEDEX	NOMBRE
7	Squirtle
11	Metapod
13	Weedle
16	Pidgey
19	Rattata
23	Ekans
33	Nidorino
48	Venonat
58	Growlithe
62	Poliwrath
70	Weepinbell
93	Haunter
99	Kingler
105	Marowak

Operador FROM

En este caso, la subquery obtiene como resultado una subtabla temporal, sobre la cual se hará una nueva consulta. La sintaxis está dada de la siguiente manera.

```
SELECT
    x.columna1, x.columna2, ...
FROM
    (
        SELECT
            columna1, columna2, ...
        FROM
            nombre_tabla2
        WHERE
            condicio
    ) x
INNER JOIN nombre_tabla1 as y on x.columna1 = y.columna1;
```

Si queremos obtener los nombres de los pokemones capturados que tienen un peso mayor a 200 kilos, debemos hacer la siguiente consulta:

```
SELECT y.nombre,
       x.pokedex,
       x.peso
FROM   (SELECT peso,
               pokedex
        FROM   mis_pokemones
        WHERE  peso > 200) x
INNER JOIN pokemones y
        ON x.pokedex = y.pokedex;
```

-- Resultado

NOMBRE	POKEDEX	PESO
Golem	76	305.8
Golem	76	320.1
Onix	95	240.8
Lapras	131	252.8
...		

Hay que recordar que en Oracle 11g no usamos la palabra **as** en los nombres de las tablas o tendremos un error en la ejecución de la consulta.

Donde la subquery:

```
SELECT peso,  
       pokedex  
FROM   mis_pokemones  
WHERE  peso > 200
```

Nos entregará el peso y el número de la pokedex de aquellos pokemones que cumplen la condición de superar los 200 kilos.

Observación sobre las subconsultas

Las subconsultas son resultados de malos análisis de las propias tablas. Ya que para combinar resultados tenemos los joins y más. Por ahora que comenzamos esta bien realizar subconsultas para resolver problemas, luego iremos aprendiendo técnicas para evitarlas completamente. En Oracle 11g, una subconsulta, no es más que una consulta o tabla temporal que debe ir entre paréntesis y ser nombrada con un alias sin la palabra **AS**.