{desafío} latam_

Gestión de bases de datos (Parte II)

Realizando consultas

Competencias

- Importar datos de un csv.
- Realizar consultas a las tablas de la base de datos.
- Crear índices para agilizar las consultas.

Introducción

Hasta el momento hemos creado tablas, ingresado datos a estas tablas, pero no hemos realizado ninguna operación con estos datos. ¿Qué hacemos con ella? ¿De qué nos sirve tener todos esos datos ordenados?

Con el comando SELECT podemos obtener información de las columnas que cumplan las condiciones indicadas, pero antes, crearemos nuevas tablas para poblarlas con datos provenientes de un archivo .csv

Importando datos de un .csv

Antes de empezar a consultar datos, vemos a importar datos de un archivo .csv

Un archivo CSV es un archivo de texto que almacena datos en forma de columnas, separada generalmente por comas.

Por lo general la primera ingesta de datos a una base de datos se realizará mediante un .csv y a continuación se detalla el flujo con el cual podemos ingresar una serie de registros alojados en un .csv .

- Abra el archivo en un editor de texto, de recomendación Sublime Text o Atom
- Si las filas están encerradas entre comillas, elimínelas
- Crear la tabla a la cual desea agregar el archivo csv.
- Cargar el archivo a través de SQL Developer.

Creando las tablas para los pokemones

Para este problema utilizaremos dos tablas, `pokemones` y `mis_pokemones`, por lo que crearemos las siguientes tablas:

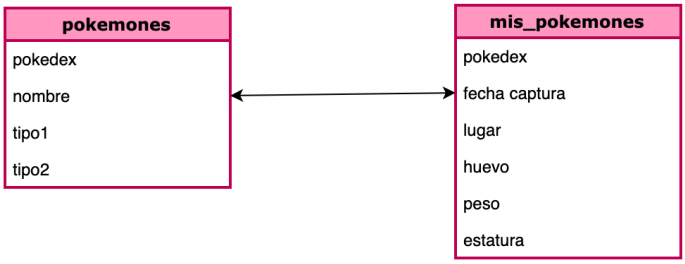


Imagen 1. Tabla pokemones.

> Contexto:
>
> - En la tabla pokemones estan los pokemones correspondientes a la generación de Kanto, los cuales son 151 pokemones.
>   El primer campo corresponde al número de la pokedex, luego su nombre, tipo1 y tipo2 para identificar el tipo de pokemon que son. Algunos solo tienen un tipo y otros pueden tener un segundo tipo.
> - En la tabla mis_pokeemones, son los pokemones que hemos atrapado, donde se indica el número de la pokedex al que corresponde ese pokemon, la fecha de captura, el lugar donde se capturó, si fue de huevo (si huevo es false es porque se capturó con una pokebola, si es true es porque salió de huevo), el peso y la estatura.

```
CREATE TABLE pokemones(
pokedex INT,
nombre VARCHAR(10),
tipo1 VARCHAR(10),
tipo2 VARCHAR(10),
PRIMARY KEY(pokedex)
);
```

pokedex	nombre	tipo1	tipo2

```
CREATE TABLE mis_pokemones(
pokedex INT,
fecha_captura DATE,
lugar VARCHAR(20),
huevo CHAR(8),
peso NUMBER,
estatura NUMBER,
FOREIGN KEY (pokedex) REFERENCES
pokemones(pokedex)
);
```

pokedex	fecha_captura	lugar	huevo	peso	estatura

Cargando la data

Ahora que tenemos creadas nuestras tablas, vamos a importar los valores de los archivos .csv pokemonesKanto.csv y mis_pokemones.csv.

Para eso seleccionaremos la tabla `pokemones` en el menú izquierdo de la pantalla, hacemos clic con el botón derecho y seleccionamos `importar datos`

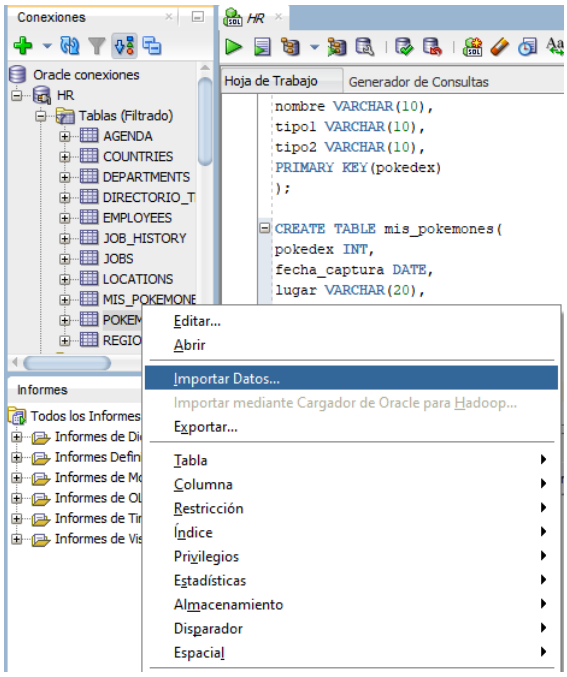


Imagen 2. Cargar tabla.

Seleccionamos el archivo desde `directorio/pokemonesKanto.csv`:

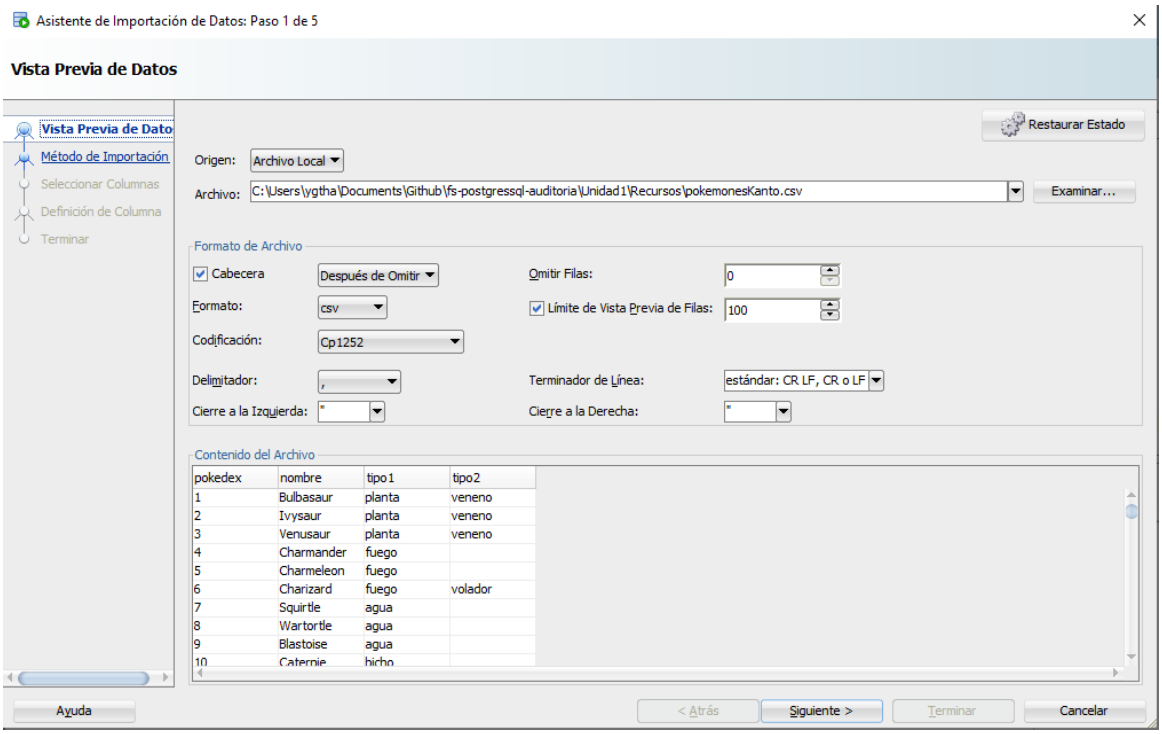


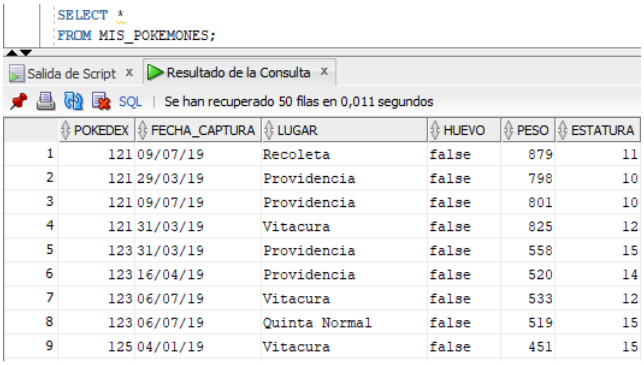
Imagen 3. Seleccionar archivo.

Mantenemos la configuración por defecto y presionamos siguiente, teniendo en cuenta que reconozca las columnas de la tabla `pokemones`.

Si todo ha salido correctamente, realizaremos la siguiente consulta para verificar que hemos cargado la información:

```
SELECT *
FROM POKEMONES;
);
```

Realizamos el mismo proceso para la tabla `mis_pokemones` y el archivo `mis_pokemones.csv`.



	POKEDEX	FECHA_CAPTURA	LUGAR	HUEVO	PESO	ESTATURA
1	121	09/07/19	Recoleta	false	879	11
2	121	29/03/19	Providencia	false	798	10
3	121	09/07/19	Providencia	false	801	10
4	121	31/03/19	Vitacura	false	825	12
5	123	31/03/19	Providencia	false	558	15
6	123	16/04/19	Providencia	false	520	14
7	123	06/07/19	Vitacura	false	533	12
8	123	06/07/19	Quinta Normal	false	519	15
9	125	04/01/19	Vitacura	false	451	15

Imagen 4. Seleccionar tabla y archivo.

Usando SELECT

Como observamos anteriormente, `SELECT` permite consultar registros de nuestras tablas.

Si solamente queremos ver una columna en específico, la sintaxis es la siguiente:

```
SELECT columna1 FROM nombre_tabla;
```

Si queremos ver más de una columna, debemos separar las columnas con coma:

```
SELECT columna1, columna2 FROM nombre_tabla;
```

Si queremos seleccionar toda la tabla, en el espacio de la columna debemos usar "*" (asterísco) , conocido como un comodín. La sintaxis de uso es la siguiente:

```
SELECT * FROM nombre_tabla;
```

En estos casos, estamos mostrando las columnas. Si queremos ver filas en específico, tendremos que usar condiciones con comandos como el ya conocido WHERE, y otros como LIMIT, que mostrarán las primeras filas hasta la cantidad indicada.

```
SELECT * FROM pokemones;
```

pokedex	nombre	tipo1	tipo2
1	'Bulbasaur'	'planta'	'veneno'
2	'Ivysaur'	'planta'	'veneno'
3	'Venusaur'	'planta'	'veneno'
4	'Charmander'	'fuego'	
5	'Charmeleon'	'fuego'	
6	'Charizard'	'fuego'	'volador'
7	'Squirtle'	'agua'	
8	'Wartortle'	'agua'	
9	'Blastoise'	'agua'	
10	'Caterpie'	'bicho'	
11	'Metapod'	'bicho'	
12	'Butterfree'	'bicho'	'volador'
13	'Weedle'	'bicho'	'veneno'
151	'Mewtwo'	'psiquico'	

En estos casos, estamos mostrando todas las columnas. Si queremos ver filas específicas, tendremos que usar condiciones con comandos como el ya conocido `WHERE`.

> Hay algunos pokemones que son de un tipo específico, y otros pueden ser de dos tipos. Es por esto que tenemos dos columnas, donde en tipo1 siempre hay un valor, pero en tipo2 depende si este pokemón tiene un segundo tipo o no.

```
-- Seleccionamos todas las filas de una columna que cumplan con la condición
SELECT columna FROM tabla WHERE condicion;
```

```
-- Seleccionamos una cantidad limitada de una columna específica
SELECT columna FROM tabla WHERE ROWNUM  < 10;
```

En nuestro ejemplo:

Volviendo a los pokemones, vamos a consultar por aquellos que son de tipo fuego:

```
SELECT * FROM pokemones WHERE tipo1='fuego';
```

pokedex	nombre	tipo1	tipo2
4	'Charmander'	'fuego'	
5	'Charmeleon'	'fuego'	
6	'Charizard'	'fuego'	'volador'
37	'Vulpix'	'fuego'	
38	'Ninetales'	'fuego'	
58	'Growlithe'	'fuego'	
59	'Arcanine'	'fuego'	
77	'Ponyta'	'fuego'	
78	'Rapidash'	'fuego'	
126	'Magmar'	'fuego'	
136	'Flareon'	'fuego'	
146	'Moltres'	'fuego'	'volador'

SELECT nos permite crear nuevas columnas, aplicando operaciones a las columnas ya existentes. Algunas operaciones normalmente utilizadas son:

- MIN: Entrega el mínimo de los datos de una columna.
- MAX: Entrega el máximo de los datos de una columna.
- LENGTH: Calcula el largo de los datos en una columna.
- COUNT: Cuenta la cantidad de ocurrencias de las filas.

En base a estas operaciones, podemos extraer resultados intermedios que puedan ser asignados a tablas o impresos en la consola.

La sintaxis para extraer una operación específica de una tabla sería la siguiente:

```
SELECT operacion(columna) AS nombre_nueva_columna FROM tabla WHERE condicion;
```

Generaremos una consulta sobre el largo de cada nombre en nuestra tabla de `pokemones`

```
-- seleccionaremos el largo de los nombres
SELECT LENGTH(nombre)
-- y le asignaremos el nombre largo_de_nombre
AS largo_del_nombre
-- de la tabla pokemones
FROM pokemones;
```

largo_de_nombre
9
7
8
9
7
8
10
10
9
...
6

y si queremos mostrar nombre y largo del nombre de los primeros 10 pokemones, podemos hacer la siguiente consulta:

```
SELECT nombre,LENGTH(nombre)
-- y le asignaremos el nombre largo_de_nombre
AS largo_del_nombre
-- de la tabla pokemones
FROM pokemones
-- mostrando los 10 primeros resultados
WHERE ROWNUM  < 10;
```

nombre	largo_del_nombre
'Bulbasaur'	9
'Ivysaur'	7
'Venusaur'	8
'Charmander'	10
'Charmeleon'	10
'Charizard'	9
'Squirtle'	8
'Wartortle'	9
'Blastoise'	9
'Caterpie'	8

(10 rows)

Ahora, si queremos saber cuantos pokemones son de tipo agua, debemos aplicar lo siguiente:

```
SELECT COUNT (pokedex)
FROM pokemones
WHERE tipo1='agua' OR tipo2='agua';
```

Alias

El comando `AS` también puede ser usado en columnas que no necesariamente se les haya aplicado una operación, a este nuevo nombre lo conoceremos como Alias, esto se aplica cuando se quiere dar un nombre más pequeño o más fácil de usar cuando este se utilizará varias veces.

```
SELECT
pkm.nombre as n,
pkm.pokedex as pkd
FROM pokemones pkm;
```

Agrupación

A veces nos vemos en la necesidad de agrupar filas que tengan datos iguales para poder trabajar con ellos de manera más sencillas, para esto tenemos el comando `GROUP BY`, que dejará las filas juntas según los valores de la columna indicada.

```
SELECT columna1, columna2, columna3
FROM tabla
GROUP BY columna1;
```

por ejemplo, si queremos saber cuantos pokemones son de agua, hada, etc, podemos consultarlo de la siguiente manera:

```
SELECT tipo1, count(*)
FROM pokemones GROUP BY tipo1;
```

tipo1	count
'hada'	2
'dragon'	3
'tierra'	8
'roca'	9
'psiquico'	8
'normal'	22
'veneno'	14
'fuego'	12
'planta'	12
'electrico'	9
'agua'	28
'fantasma'	3
'bicho'	12
'hielo'	2
'lucha'	7

(15 rows)

y los de tipo2:

```
SELECT tipo1, count(*)
FROM pokemones GROUP BY tipo1;
```

tipo2	count
'hielo'	3
'lucha'	1
'planta'	2
'hada'	3
'tierra'	6
'agua'	4
'roca'	2
'psiquico'	6
'volador'	19
'veneno'	19
'acero2'	2

(12 rows)

Ordenamiento

Similar a lo que es agrupar, puede que necesitemos que nuestras filas estén ordenadas según los valores que tengan en alguna columna. Un ejemplo común sería ordenar de menor a mayor. Podemos ordenar las filas utilizando el comando ORDER BY:

```
SELECT columna1, columna2, ...
FROM nombre_tabla
[WHERE condiciones]
ORDER BY columna2 [DESC | ASC];
```

- `DESC`: orden decreciente
- `ASC`: orden ascendente

Podemos ordenar de forma decreciente DESC o ascendente ASC, ubicando el comando después de la columna seleccionada para ordenar las filas.

Por ejemplo, si queremos ver las filas ordenadas con nombres desde la Z hasta la A, debemos hacer lo siguiente:

```
SELECT *
FROM pokemones
ORDER BY nombre DESC;
```

pokedex	nombre	tipo1	tipo2
41	'Zubat'	'veneno'	'volador'
145	'Zapdos'	'electrico'	'volador'
40	'Wigglytuff'	'normal'	'hada'
110	'Weezing'	'veneno'	
70	'Weepinbell'	'planta'	'veneno'
13	'Weedle'	'bicho'	'veneno'
8	'Wartortle'	'agua'	
37	'Vulpix'	'fuego'	
100	'Voltorb'	'electrico'	
45	'Vileplume'	'planta'	'veneno'
63	'Abra'	'psiquico'	

Creando índices

Cuando revisamos un libro, y necesitamos buscar un capítulo en particular o bien buscar sobre alguna temática, debemos buscar en su índice para poder llegar de manera más rápida a esta información y ocurre de manera similar en una base de datos.

Cuando creamos un índice en una tabla de la base de datos, lo que hacemos es agilizar el tiempo de respuesta de esta búsqueda.

La sintaxis para agregar un índice a una tabla es la siguiente:

```
CREATE INDEX nombre_indice on nombre_tabla(nombre_columna);
```

Por ejemplo, en nuestra tabla `pokemones`, nosotros bien cuando hacemos consultas, buscamos por nombres de pokemones más que por número en la pokedex. Entonces podemos agregar un índice a la columna de nombre.

```
CREATE INDEX index_pokemon_nombre on pokemones(nombre);
COMMIT;
```

Si queremos saber que columnas de nuestras tablas tienen índice, podemos usar el siguiente comando:

```
SELECT * FROM all_indexes WHERE table_name = nombre_tabla;
```

En nuestro caso, al buscar en la tabla de pokemones , donde observamos lo siguiente:

```
SELECT * FROM all_indexes WHERE table_name = 'POKEMONES';
```

	OWNER	INDEX_NAME	INDEX_TYPE	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	UNIQUENESS	COMPRESSION	PREFIX_LENGTH	TABLESPACE_NAME	INI_TRANS	MAX_TRANS
1	HR	SYS_C007009	NORMAL	HR	POKEMONES	TABLE	UNIQUE	DISABLED	(null)	USERS	2	255
2	HR	INDEX_POKEMON_NOMBRE	NORMAL	HR	POKEMONES	TABLE	NONUNIQUE	DISABLED	(null)	USERS	2	255

Imagen 5. Crear índice.

Podemos ver que ya existe un índice de caracter único sobre la clave primaria de la columa pokedex. Cuando se crea una clave primaria, postgres crea automáticamente un índice sobre esta columna.

Si queremos eliminar un índice, basta con ejecutar el comando:

```
drop index nombre_indice;
commit;
```

Operaciones de unión entre Tablas

Competencias

- Realizar consultas usando el comando JOIN.
- Realizar sub-consultas o subquery.

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, peor debemos destacar que una de las virtudes de implementar claves primarias y foráneas en nuestas tablas es la posibilidad de generar operaciones entre tablas para incorporar información de distintas fuentes.

Tomemos el siguiente caso hipotético: Se necesita unir las filas de tablas que cumplan con algún dato común, lo cual podemos expresar así:

```
-- 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, esto se ve traducido en distintos tipos de JOIN que se presentan 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 las tablas

Veamos ejemplos de consultas usando los distintos tipos de `JOIN`.

INNER JOIN

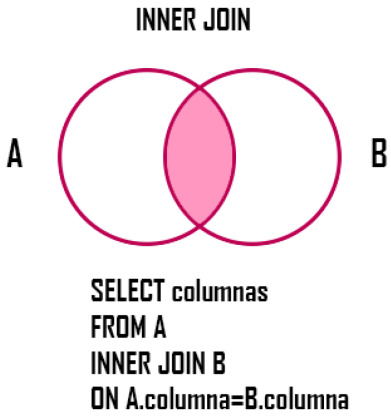


Imagen 6. Inner join.

```
SELECT * FROM pokemones INNER JOIN  mis_pokemones on
pokemones.pokedex=mis_pokemones.pokedex ORDER BY nombre;
```

pokedex	nombre	tipo1	tipo2	pokedex	fecha_captura	lugar	huevo
63	Abra	psiquico		63	2019-07-19	Huechuraba	f
63	Abra	psiquico		63	2019-06-01	Huechuraba	f
144	Articuno	hielo	volador	144	2019-07-13	Independencia	f
9	Blastoise	agua		9	2019-01-26	Independencia	f
9	Blastoise	agua		9	2019-02-16	Independencia	f
1	Bulbasaur	planta	veneno	1	2019-01-30	Estación Central	f
1	Bulbasaur	planta	veneno	1	2019-01-31	Independencia	f
12	Butterfree	bicho	volador	12	2019-03-04	Recoleta	f
10	Caterpie	bicho		10	2019-01-26	Independencia	f
10	Caterpie	bicho		10	2019-02-13	Cerro Navia	f
113	Chansey	normal		113	2019-04-20	Independencia	f
6	Charizard	fuego	volador	6	2019-02-08	Independencia	f
6	Charizard	fuego	volador	6	2019-01-27	Independencia	f
41	Zubat	veneno	volador	41	2019-02-22	Santiago	f

(199 rows)

En este caso, se unen ambas tablas cuando se cumple que tienen el mismo número de pokedex. Es por eso que se muestran los 199 registros que tenemos en la tabla `mis_pokemones` y se añade como información los datos de nombre, tipo1 y tipo2.

LEFT JOIN

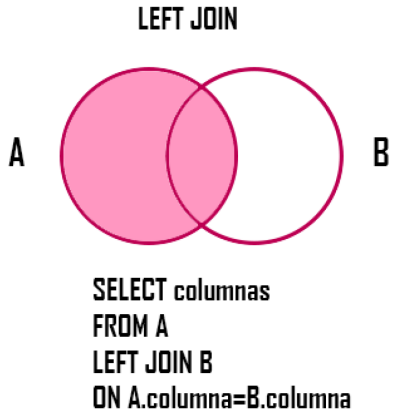


Imagen 7. Left join.

```
SELECT * FROM pokemones LEFT JOIN  mis_pokemones on
pokemones.pokedex=mis_pokemones.pokedex ORDER BY nombre;
```

pokedex	nombre	tipo1	tipo2	pokedex	fecha_captura	lugar	huevo
63	'Abra'	'psiquico'		63	'2019-07-19'	'Huechuraba'	f
63	'Abra'	'psiquico'		63	'2019-06-01'	'Huechuraba'	f
142	'Aerodactyl'	'roca'	'volador'				
65	'Alakazam'	'psiquico'					
24	'Arbok'	'veneno'					
59	'Arcanine'	'fuego'					
144	'Articuno'	'hielo'	'volador'	144	'2019-07-13'	'Independencia'	f
15	'Beedrill'	'bicho'	'veneno'				
69	'Bellsprout'	'planta'	'veneno'				
9	'Blastoise'	'agua'		9	'2019-01-26'	'Independencia'	f
9	'Blastoise'	'agua'		9	'2019-02-16'	'Independencia'	f
1	'Bulbasaur'	'planta'	'veneno'	1	'2019-01-31'	'Independencia'	f
1	'Bulbasaur'	'planta'	'veneno'	1	'2019-01-30'	'Estación Central'	f
12	'Butterfree'	'bicho'	'volador'	12	'2019-03-04'	'Recoleta'	f

(242 rows)

En este caso podemos ver que a pesar que no tengamos pokemones tales como Alakazam, Arbok, Arcanite, etc, nos muestra de igual manera los registros de la tabla `pokemones`.

LEFT JOIN NULL

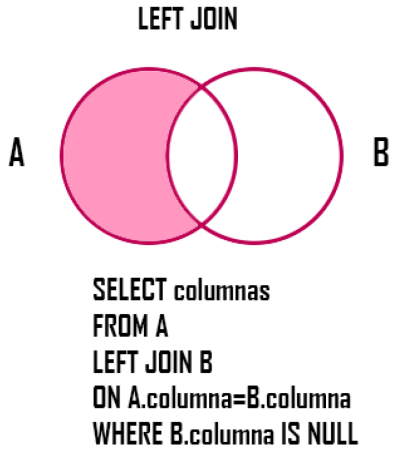


Imagen 8. Left join null.

```
SELECT * FROM pokemones LEFT JOIN  mis_pokemones on pokemones.pokedex=mis_pokemones.pokedex WHERE mis_pokemones.pokedex IS NULL ORDER BY nombre;
```

pokedex	nombre	tipo1	tipo2	pokedex	fecha_captura	lugar	huevo
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'				
74	'Geodude'	'roca'	'tierra'				
42	'Golbat'	'veneno'	'volador'				
118	'Goldeen'	'agua'					
75	'Graveler'	'roca'	'tierra'				
130	'Gyarados'	'agua'	'volador'				
107	'Hitmonchan'	'lucha'					
106	'Hitmonlee'	'lucha'					
39	'Jigglypuff'	'normal'	'hada'				

pokedex	nombre	tipo1	tipo2	pokedex	fecha_captura	lugar	huevo
124	'Jynx'	'hielo'	'psiquico'				
110	'Weezing'	'veneno'					

(43 rows)

Aca podemos ver que los registros que nos muestra son los que corresponden a la tabla de pokemones, bajo la condicion de que no tengan registro en la pokedex.

FULL JOIN

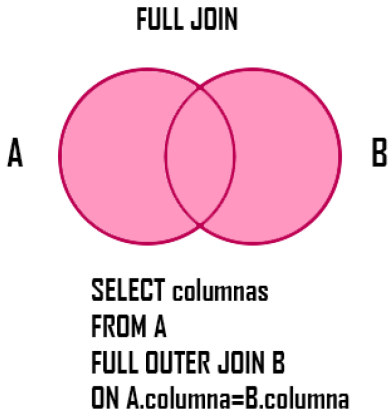


Imagen 9. Full join.

```
SELECT * FROM pokemones FULL OUTER JOIN  mis_pokemones on
pokemones.pokedex=mis_pokemones.pokedex ORDER BY nombre;
```

pokedex	nombre	tipo1	tipo2	pokedex	fecha_captura	lugar	huevo
63	'Abra'	'psiquico'		63	'2019-07-19'	'Huechuraba'	f
63	'Abra'	'psiquico'		63	'2019-06-01'	'Huechuraba'	f
142	'Aerodactyl'	'roca'	'volador'				
65	'Alakazam'	'psiquico'					
24	'Arbok'	'veneno'					
59	'Arcanine'	'fuego'					
144	'Articuno'	'hielo'	'volador'	144	'2019-07-13'	'Independencia'	f
15	'Beedrill'	'bicho'	'veneno'				
69	'Bellsprout'	'planta'	'veneno'				
9	'Blastoise'	'agua'		9	'2019-01-26'	'Independencia'	f
9	'Blastoise'	'agua'		9	'2019-02-16'	'Independencia'	f
1	'Bulbasaur'	'planta'	'veneno'	1	'2019-01-31'	'Independencia'	f
41	'Zubat'	'veneno	'volador'	41	'2019-06-26'	'Peñalolén'	f

(242 rows)

En este caso muestra información de todos los registros existentes en ambas tablas.

FULL JOIN NULL

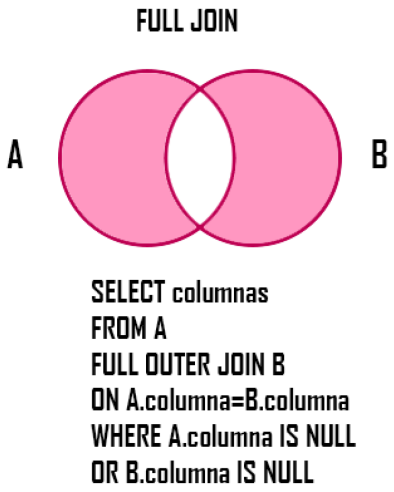


Imagen 10. Full join null.

```
SELECT * FROM pokemones FULL OUTER JOIN  mis_pokemones on
pokemones.pokedex=mis_pokemones.pokedex WHERE pokemones.tipo2 is null or
mis_pokemones.pokedex is null ORDER BY nombre;
```

pokedex	nombre	tipo1	tipo2	pokedex	fecha_captura	lugar	huevo
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'					
110	'Weezing'	'veneno'					

(43 rows)

Muestra los registros que no se relacionen entre ambas tablas.

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.

En algunas ocasiones, para poder hacer una consulta a una tabla es necesaria la consulta de otra tabla, la cual no es nuestro objetivo final. Para eso, existen las subqueries (sub-consultas), las cuales son consultas temporales que sólo estarán para poder obtener el resultado de nuestra consulta. Si está familiarizado con el concepto de recursión, esto resultará familiar para usted.

Operador WHERE

Por ejemplo, queremos obtener como resultado los nombres de los pokemones que fueron obtenidos por huevos.
Con la siguiente consulta, podemos obtener el n{umero de la pokedex de los pokemones que están en la tabla `mis_pokemones` pero no sus nombres.

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

pokedex
7
11
13
16
19
23
33
48
58
62
70
93
99
105

(14 rows)

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= 'true'
   );
```

Donde la subquery es equivalente a la que habíamos realizado inicialmente, y obtenemos el número de la pokedex y el nombre de los pokemones que fueron eclosionados.

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'

(14 rows)

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.columa1, xcoluma2, ...
FROM (
    SELECT columa1, columna2, ...
    FROM nombre_tabla2
    WHERE condicion
  ) as x
INNER JOIN nombre_tabla1 as y on x.columna1 = y.columna1;
```

Si queremos obtener los nombres de los pokemones 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;
```

nombre	pokedex	peso
'Golem'	76	305.8
'Golem'	76	320.1
'Onix'	95	240.8
'Lapras'	131	252.8
'Snorlax'	143	460.1
'Dragonite'	149	288.9
'Dragonite'	149	279.5
'Dragonite'	149	290.1

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.

Algunas reglas que deben seguir las subqueries

- Las consultas internas **deben** estar encapsuladas entre paréntesis.
- Una subquery puede tener **sólo una columna** especificada en `SELECT`, con la excepción de múltiples columnas definidas en la consulta principal.
- El comando `ORDER BY` **no se puede utilizar** en una consulta interna. La excepción es que esta instrucción si puede ser incluída en la consulta principal.
- Para obtener un resultado similar a `ORDER BY` dentro de una consulta interna, se puede implementar el comando `GROUP BY`.
- Aquellas consultas internas que retornen **más de una fila** sólo pueden ser utilizadas con operadores de múltiples valores como `IN`.

Transacciones

Competencias:

- Conocer que son las transacciones y que utilidad nos genera al momento de gestionar una base de datos.
- Cargar una base de datos utilizando dump.

Introducción

Hasta el momento hemos creado un par de tablas en nuestra base de datos, hemos cargado datos de manera manual y a través de formato .csv, para luego realizar distintos tipos de consultas para obtener información. Pero se han preguntado ¿Qué pasa si al momento de crear una consulta ya sea para recuperar información, crear nuevos registros, entre otros, se crea algún problema y nuestra operación no puede realizarse? Es por esto que existen las transacciones en Oracle SQL.

Transacciones

Las transacciones son secuencias de instrucciones ordenadas, las cuáles pueden ser indicadas de forma manual o pueden ser aplicadas automáticamente. Estas realizan cambios en las bases de datos a la hora de aplicar comandos de manipulación de columnas y registros.

Estas transacciones tienen las siguientes propiedades:

- Atomicidad: Todas las operaciones realizadas en la transacción deben ser completadas. En el caso que ocurra un fallo, esta la transacción es abortada y devuelve todo al estado previo a la transacción.
- Consistencia: La base de datos cambiará solamente cuando la transacción se haya realizado.
- Aislamiento: Las transacciones pueden ocurrir independientes unas de otras.
- Durabilidad: El resultado de la transacción persiste a pesar de que el sistema falle.

Es posible tener control de las transacciones. Para eso, existen los siguientes comandos:

- `BEGIN` : El sistema permite que se ejecuten todas las sentencias SQL que necesitemos.
- `COMMIT` : Guarda los cambios de la transacción.
- `ROLLBACK` : Retrocede los cambios realizados.
- `SAVEPOINT` : Guarda el punto de partida al cual volver a la hora de aplicar ROLLBACK
- `SET TRANSACTION` : Le asigna nombre a la transacción.

Estos comandos sólo pueden ser usados con las operaciones INSERT, UPDATE y DELETE, ya que aquellos que manipulan toda la tabla hacen este proceso automáticamente. La sintaxis de estos comandos es la siguiente:

- `COMMIT;`
- `SAVEPOINT nombre_savepoint;`
- `ROLLBACK [TO nombre_savepoint];`

Lo que esta entre corchetes es de carácter opcional, por lo que podemos decirle a ROLLBACK a que punto volver, o este volverá al último punto por defecto.

```
SET TRANSACTION [READ ONLY|WRITE][NAME nombre_transaccion];
```

En este caso, podemos usar READ ONLY para solamente leer la base de datos, READ WRITE para leer y escribir sobre ella, y poder nombrar la transacción con el comando NAME.

Veamos el siguiente ejemplo:

```
SET TRANSACTION READ WRITE NAME primera_transaccion;
INSERT INTO nombre_tabla (columna1, columna2, columna3) VALUES (valor1,
valor2, valor3);
SAVEPOINT registro_ingresado;
```

En este caso creamos una transacción en la que se puede leer y escribir sobre la base de datos que tiene de nombre 'primera_transaccion', luego insertamos un valor y hicimos SAVEPOINT para poder volver a este punto en cualquier caso.

Veamos el siguiente ejemplo:

```
UPDATE nombre_tabla SET columna1=valor_nuevo WHERE condicion;
ROLLBACK TO registro_ingresado;
COMMIT;
```

Ahora, realizamos un UPDATE, sin embargo, nos devolvimos al punto guardado en SAVEPOINT, para así finamente guardarlo haciendo COMMIT.

Respaldar una base de datos utilizando SQL Developer

SQL Developer contiene herramientras que nos permiten de manera simple generar copias de nuestra base de datos, para así respaldarla o bien cargar datos de este respaldo en un momento determinado.

Respaldo de una tabla

Para crear un archivo a partir de una base de datos, podemos utilizar las herramientras de SQL Developer nos propociona.

Hacemos clic derecho en la tabla que queremos respaldar y seleccionamos `exportar`

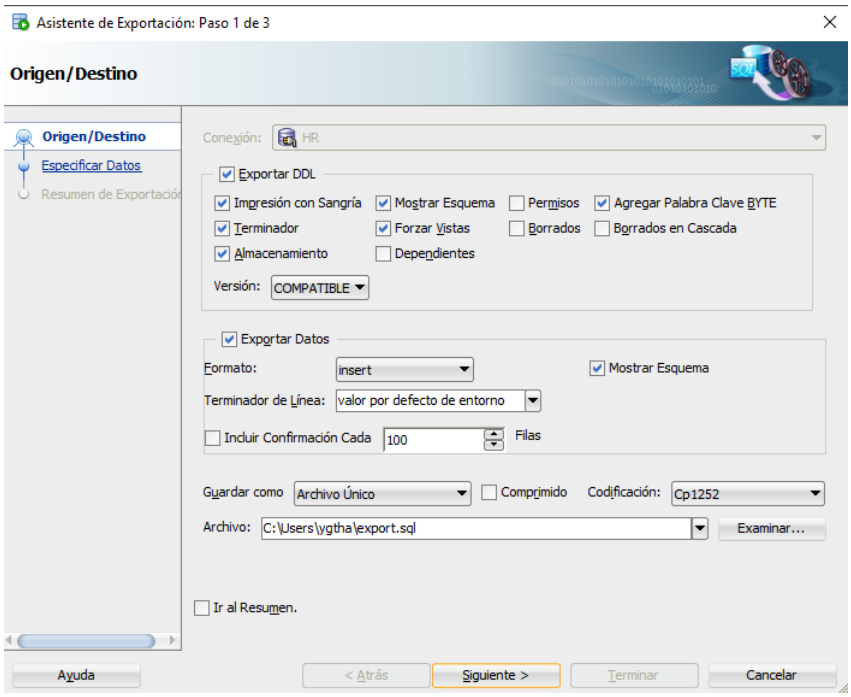


Imagen 11. Respaldar tabla.

En general dejaremos la configuración por defecto, teniendo presente la ruta en que se creará el archivo `export.sql`

Una vez terminado el proceso, se genera un script que permite replicar la estructura y datos que contiene la tabla seleccionada.

Restaurar una base de datos

Para restaurar una base de datos, sólo necesitamos ejecutar el script generado.