Realizando consultas

¡Comencemos!

¿Qué aprenderás?

- Importar datos de un csv para la automatización de sentencias SQL.
- Implementar alias en consultas SQL para un manejo personalizado de los campos y las tablas involucradas.
- Reconocer las funciones básicas que se pueden usar en consultas para obtener datos calculados.
- Realizar consultas con funciones a las tablas de la base de datos.
- Crear índices en las tablas 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?

Existen diferentes comandos que podemos definir en consultas SQL, y entre estos encontramos la palabra reservada SELECT con la que podemos obtener información de las columnas que cumplan las condiciones indicadas. El envío de consultas para obtener datos es el día a día en el mundo de las bases de datos. Cada vez que entras a un sistema que almacena datos se está generando por detrás consultas "SELECT" que obtienen de las tablas la información necesaria para el correcto funcionamiento de una aplicación.

Las bases de datos de forma paralelas a las tecnologías SQL se han creado en software como Excel, con el que podemos exportar datos en formato "csv", el cual es un acrónimo de "comma-separated values", que en español significa "valores separados por comas". Teniendo una tabla recién creada podremos importar la información con uno de estos archivos y evitamos la carga fila por fila en las tablas.

¡Vamos con todo!

Creando las tablas para los Pokemones

Supongamos que estamos creando un videojuego con la temática de pokémon, donde debemos registrar en nuestra base de datos los pokemones que se usarán en el juego. Cada pokémon debe ser identificable por un número definido por la pokédex y la trama original de la serie. El videojuego tiene un protagonista maestro pokémon y debe hacerse un registro por cada pokémon capturado y los datos de esta acción.

Para este ejemplo utilizaremos dos tablas, ***pokemones*** y ***mis_pokemones***, por lo que procedamos a crear las tablas que te muestro en la siguiente imagen:

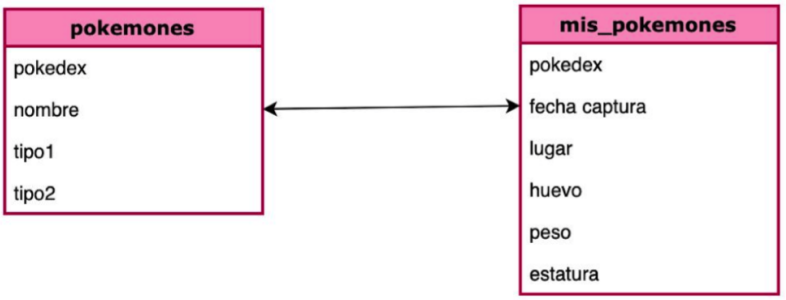

Imagen 1. Modelo relacional

Contexto

- Los pokemones para este ejemplo son correspondientes a la generación de Kanto, los cuales son 151 pokémons y **puedes encontrar el csv que los contiene en el apoyo lectura.**

- El primer campo corresponde al número de la pokédex, luego su nombre, tipo1 y tipo2 para identificar el tipo de pokémon que son. Algunos solo tienen un tipo y otros pueden tener un segundo tipo.

- En la tabla **mis_pokemones**, se encuentran los pokemones que hemos atrapado, donde se indica el número de la pokédex al que corresponde ese pokémon, 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.

Con la siguiente instrucción creamos la tabla "**pokemones**" y su estructura.

```
CREATE TABLE pokemones
  (
```

```
    pokedex NUMBER,
    nombre  VARCHAR2(10),
    tipo1   VARCHAR2(10),
    tipo2   VARCHAR2(10),
    PRIMARY KEY(pokedex)
  );

-- estructura
POKEDEX|NOMBRE|TIPO1|TIPO2|
-------|------|-----|-----|
```

Con la siguiente instrucción creamos la tabla "**mis_pokemones**" y su estructura correspondiente.

```
CREATE TABLE mis_pokemones
  (
    pokedex       NUMBER,
    fecha_captura DATE,
    lugar         VARCHAR2(20),
    huevo         CHAR(5),
    peso          NUMBER(10,2),
    estatura      NUMBER(10,2),
    FOREIGN KEY (pokedex) REFERENCES pokemones(pokedex)
  );

-- estructura
POKEDEX|FECHA_CAPTURA|LUGAR|HUEVO|PESO|ESTATURA|
-------|-------------|-----|-----|----|--------|
```

Importando datos de un .csv

Un archivo CSV (archivo separado por comas, aunque se puede usar otro criterio de separación), es un archivo de texto que almacena datos en forma de columnas, separadas 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.

Realizaremos entonces la carga de un archivo csv desde los clientes Sql Developer y DBeaver.

Cargando datos desde DBeaver

- Para eso tenemos que dirigirnos a la tabla que queremos cargar:

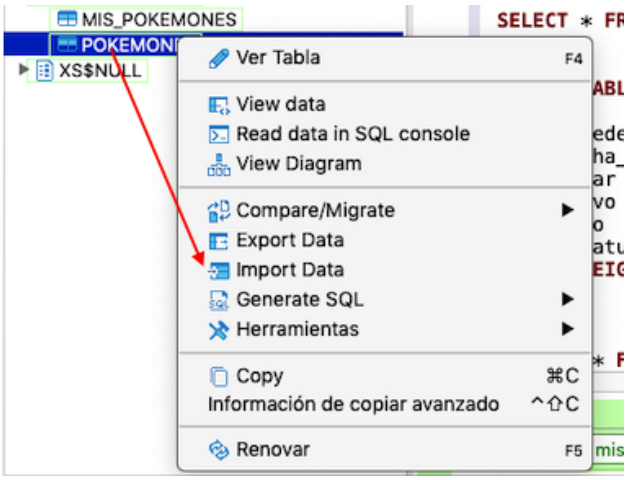

Imagen 2. Importar csv DBeaver 1.

- Seleccionamos la opción csv y presionamos siguiente:

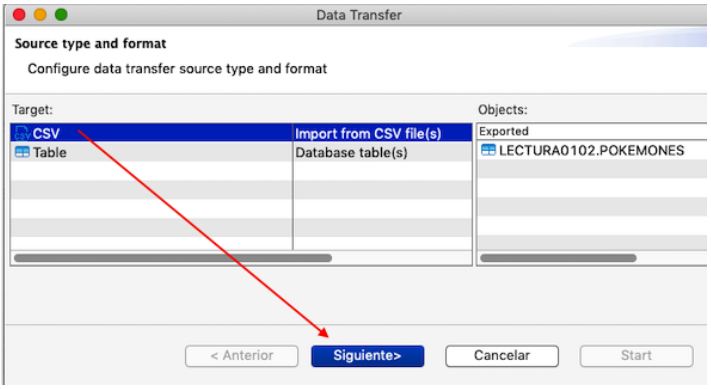

Imagen 3. Importar csv DBeaver 2.

- Seleccionamos el recurso.

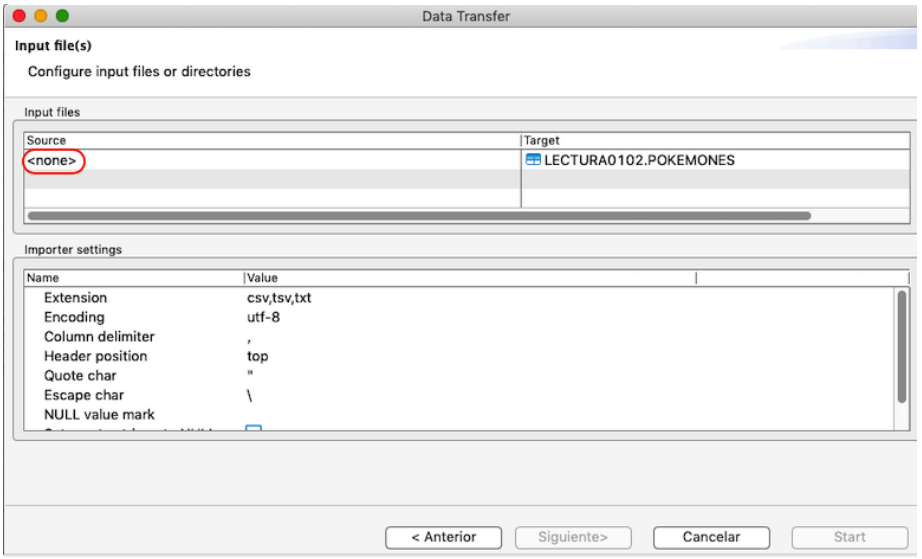

Imagen 4. Importar csv DBeaver 3.

- Seleccionamos nuestro archivo csv.

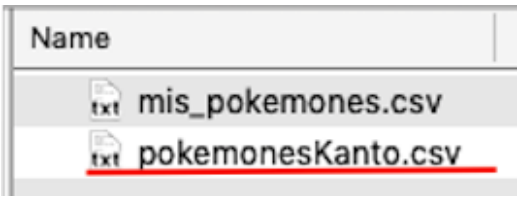

Imagen 5. Importar csv DBeaver 4.

- Dejamos las opciones por defecto.

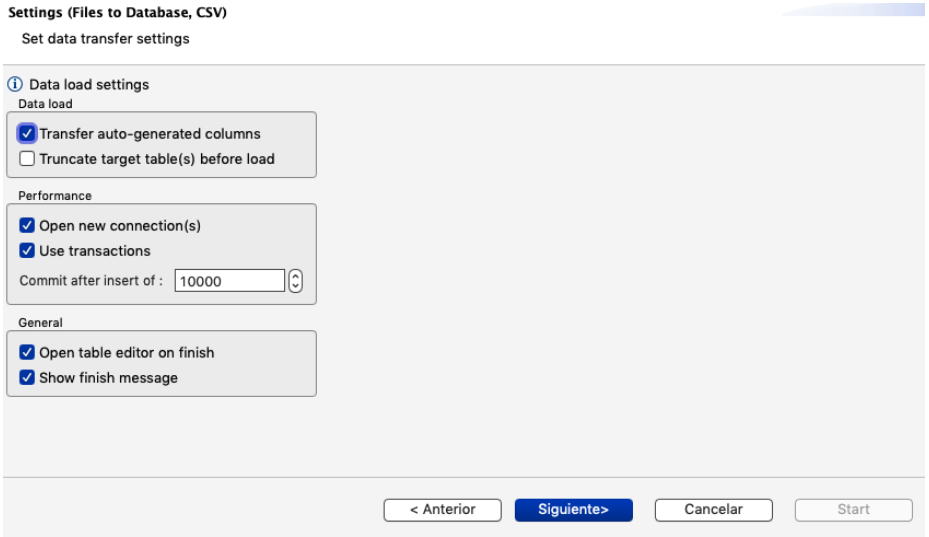

Imagen 6. Importar csv DBeaver 5.

- Podemos ver una previsualización de lo que se está importando, presionamos siguiente.

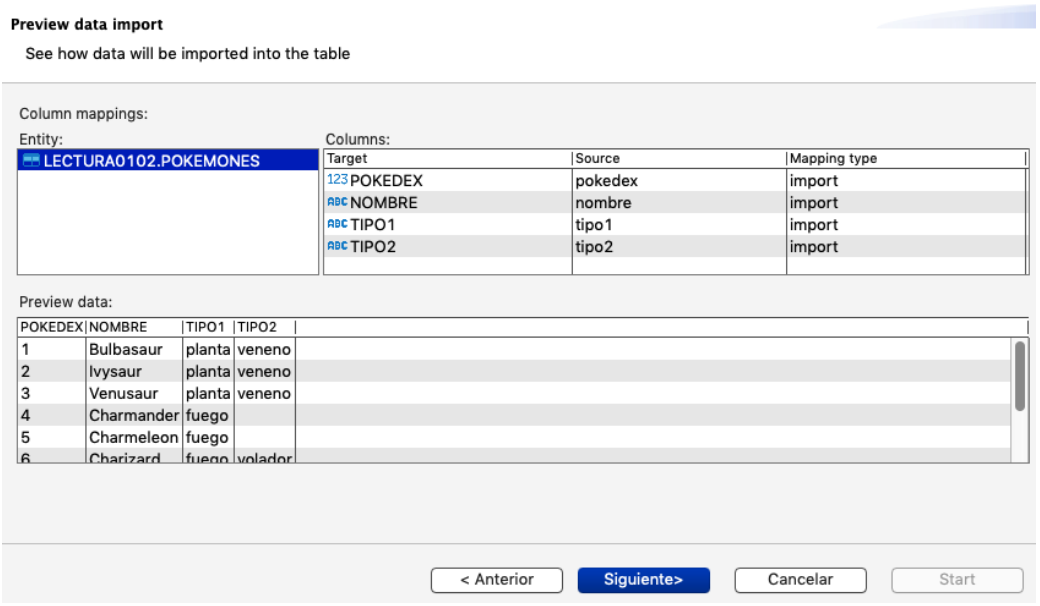

Imagen 7. Importar csv DBeaver 6.

- Confirmamos el resumen de la información de importación y presionamos start para comenzar.

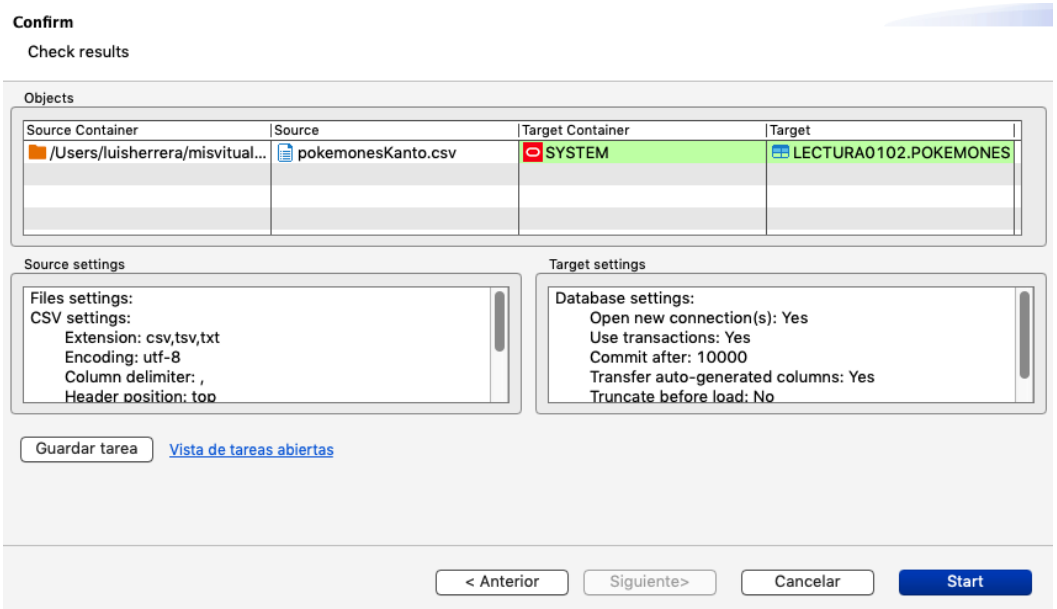

Imagen 8. Importar csv DBeaver 7.

Al finalizar podemos comprobar si se cargaron correctamente ejecutando una consulta:

```
SELECT * FROM POKEMONES;
POKEDEX|NOMBRE    |TIPO1    |TIPO2   |
-------|----------|---------|--------|
      1|Bulbasaur |planta   |veneno  |
      2|Ivysaur   |planta   |veneno  |
... total de filas 151.
```

Importando desde Sql Developer

- Luego de conectarnos nos vamos a la tabla que queremos poblar. Probemos la tabla que queda y ejecutamos la siguiente opción.

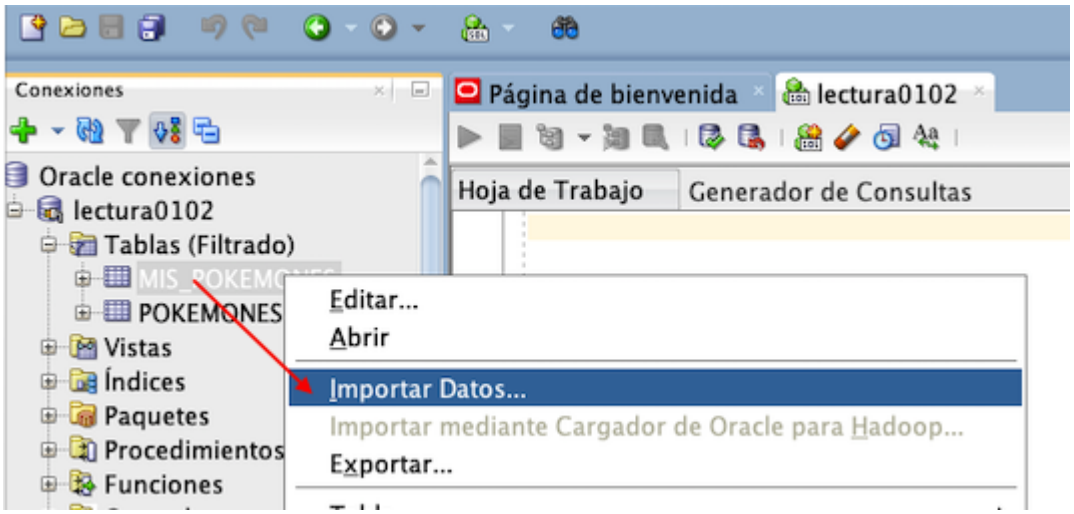

Imagen 9. Importar csv Sql Developer 1.

- Seleccionamos los criterios para importar, hay que tener en cuenta, que no siempre un csv está separado por comas. Seleccionamos el archivo y presionamos abrir.

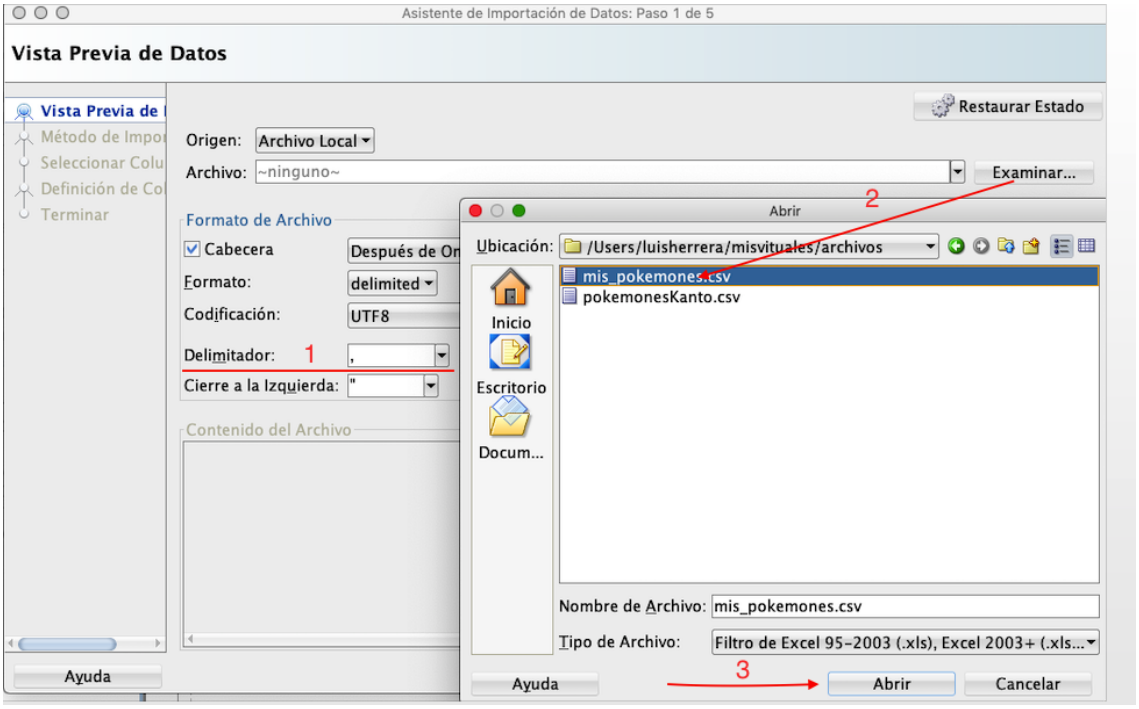

Imagen 10. Importar csv Sql Developer 2.

- Se nos presenta una vista previa con la información a importar; presionamos siguiente.

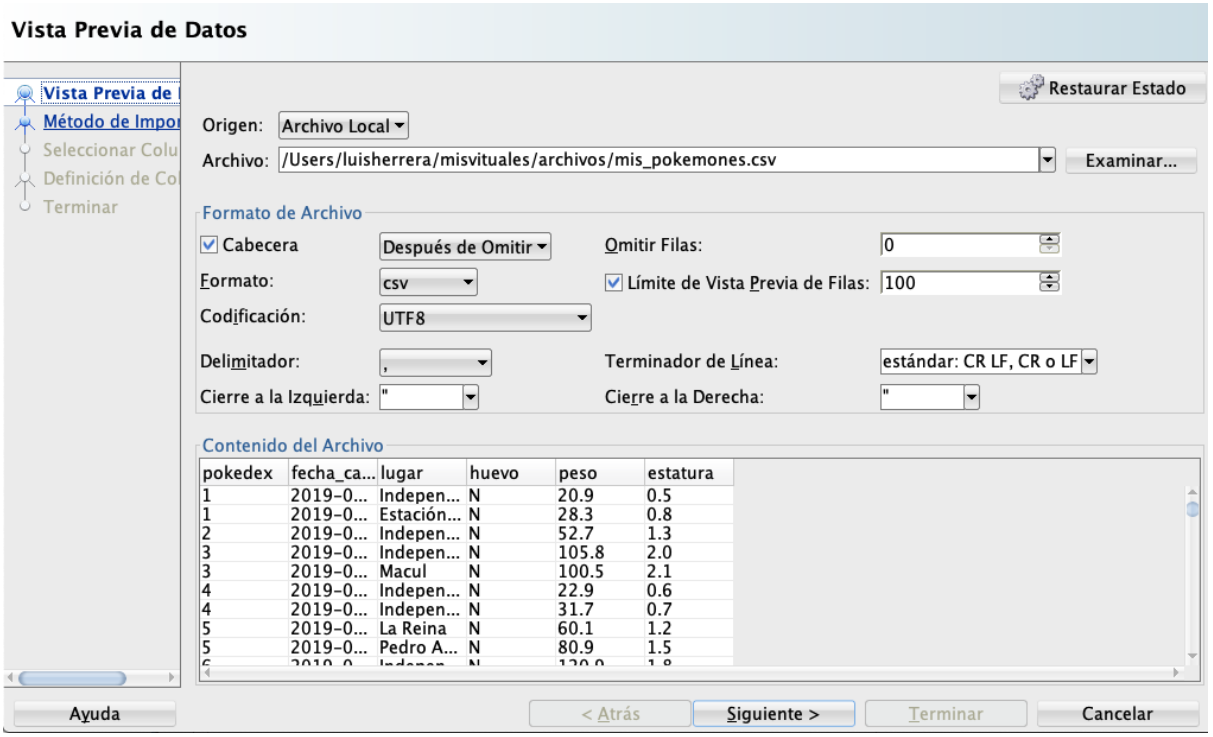

Imagen 11. Importar csv Sql Developer 3.

- Se nos presentan varias ventanas de confirmación, presionamos siguiente, hasta llegar a la siguiente ventana, presionamos terminar y terminamos la operación.

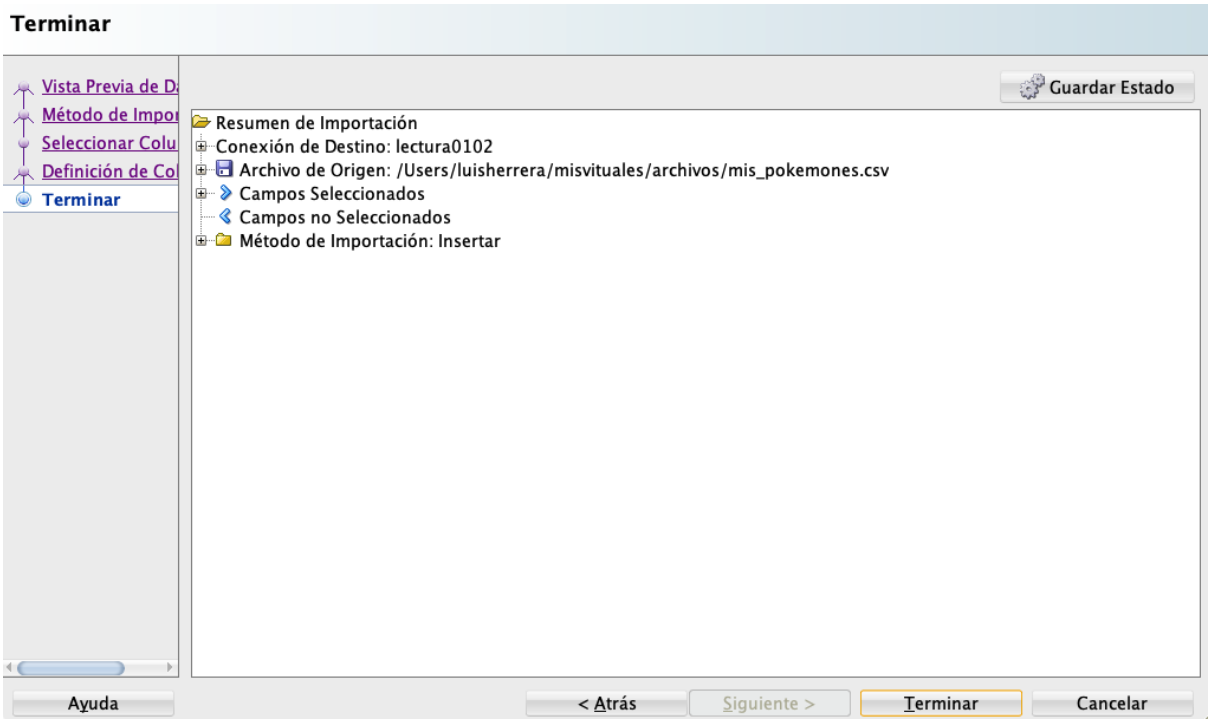

Imagen 12. Importar csv Sql Developer 4.

- Comprobamos que los datos se importaron correctamente:

```
SELECT * FROM mis_pokemones;

POKEDEX|FECHA_CAPTURA      |LUGAR              |HUEVO|PESO|ESTATURA|
-------|-------------------|-------------------|-----|----|--------|
      1|2019-01-31 00:00:00|Independencia      |N    | 209|       5|
      1|2019-01-30 00:00:00|Estación Central   |N    | 283|       8|
      2|2019-02-01 00:00:00|Independencia      |N    | 527|      13|
      3|2019-03-01 00:00:00|Independencia      |N    |1058|      20|
      3|2019-01-29 00:00:00|Macul              |N    |1005|      21|
      4|2019-02-01 00:00:00|Independencia      |N    | 229|       6|
      4|2019-01-27 00:00:00|Independencia      |N    | 317|       7|
      5|2019-02-03 00:00:00|La Reina           |N    | 601|      12|
... total de filas 199.
```

Pudimos entonces realizar la carga de las dos tablas que hemos creado, con datos importados desde un csv. Dejamos al lector la inquietud de investigar otros tipos de importaciones, por ejemplo importar un .xls.

Consultando tablas

Para obtener los datos de una tabla, ocupamos la palabra reservada SELECT que sirve para la selección de los campos de una tabla. 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 todos los datos de la tabla, en el espacio de la columna debemos usar "*" (asterisco), conocido como **un comodín**. Hay que mencionar que hacer esto en ambiente productivo, ya sea para reportes o en un código, no es una buena práctica, porque las tablas se actualizan con el tiempo y el comodín, podría traer en el futuro, columnas que no necesitemos y esto pondría en riesgo el resultado. Es por este motivo que si no estamos probando, no debemos usar el comodín; en su lugar, preferimos especificar las columnas. La sintaxis entonces queda de la siguiente manera:

```
SELECT * FROM nombre_tabla;
```

En estos casos, estamos mostrando las columnas de todas las filas. Si queremos ver una fila en específico, tendríamos que ocupar el comando WHERE para generar la búsqueda de un registro que tenga “x” propiedad definida con “x” valor o la condición de preferencia, por ejemplo:

Ejemplo 1

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

Ejemplo 2

```
SELECT * FROM nombre_tabla WHERE columna='valor;
```

Ejemplo 3

```
SELECT * FROM nombre_tabla WHERE columna > 10;
```

Las condiciones se crean basadas en la siguiente lista de opciones:

Operador	Descripción
=	Igual
>	Mayor que
<	Menor que
<>, !=	Distinto. El operador != es solo válido en algunas versiones de SQL
>=	Mayor o igual
<=	Menor o igual
BETWEEN	Entre cierto rango
LIKE	Por patrón
IN	Para especificar múltiples valores posibles para una columna

Tabla 1. Operadores condicionales.

Por último, si quisiéramos de 100 datos obtener solo 20, tenemos a disposición en el lenguaje SQL el comando **ROWNUM,** para la definición de la cantidad de registros o filas que se quieran consultar de una tabla. Ejemplos de esta sintaxis sería la siguiente:

Ejemplo 1

```
-- Seleccionamos una cantidad limitada de una columna específica
SELECT columna FROM tabla WHERE rownum <= cantidad;
```

Ejemplo 2

```
SELECT * FROM nombre_tabla WHERE rownum <= 20;
```

Pongamos en práctica lo aprendido y consultemos ahora nuestros pokémons importados de la siguiente manera:

```
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     |        |
                 ...
... número de filas 151

SELECT * FROM POKEMONES WHERE rownum <= 3;

POKEDEX|NOMBRE   |TIPO1 |TIPO2 |
-------|---------|------|------|
      1|Bulbasaur|planta|veneno|
      2|Ivysaur  |planta|veneno|
      3|Venusaur |planta|veneno|
... número de filas 3
```

Podemos observar entonces, que esta técnica, se debe emplear para limitar la cantidad de información que se mostrará en pantalla. Esto varía según el motor, por ejemplo **en otros motores** se utilizan palabras como LIMIT o TOP; pero ese no es el caso de Oracle.

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 pokémon tiene un segundo tipo o no.

En nuestro ejemplo:

Volviendo a los pokemones, vamos a consultar por aquellos que son de tipo fuego usando la siguiente instrucción SQL:

```
SELECT pokedex,
```

```
       nombre,
       tipo1,
       tipo2
FROM   pokemones
WHERE  tipo1 = 'fuego';

-- resultado
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|
```

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, por ejemplo la siguiente instrucción usa la letra “n” para referirse al nombre de un pokémon y “pkd” para referirse a la pokédex.

En Oracle 11g no se prefiere no usar el uso del as en el alias de las tablas. Debemos poner el alias directamente sin el as.

```
SELECT
  nombre_tabla_referencia.nombre_columna AS nombre_referencia
FROM
  nombre_tabla nombre_tabla_referencia;
```

Ejemplo

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

-- resultado:
N         |PKD|
----------|---|
Bulbasaur |  1|
Ivysaur   |  2|
Venusaur  |  3|
Charmander|  4|
Charmeleon|  5|
... 151 filas.
```

Funciones en consultas

Como ya se aprendió, la palabra SELECT la utilizamos para seleccionar columnas en una base de datos, sin embargo, podemos agregar a la instrucción operaciones o funciones para personalizar aún más la consulta. Algunas funciones 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.
- **SUM:** Suma los valores de una columna ignorando los valores null.

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

Generemos 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_del_nombre
AS largo_del_nombre
-- de la tabla pokemones
FROM pokemones;

-- resultado
```

```
LARGO_DEL_NOMBRE|
----------------|
               9|
               7|
               8|
              10|
              10|
... 151 filas en total.
```

Continuando con nuestro ejemplo, si queremos mostrar la columna nombre y el largo del nombre de los primeros 10 pokemones, podemos hacer la siguiente consulta:

```
SELECT nombre,
-- y le asignaremos el nombre largo_de_nombre
       Length(nombre) AS largo_del_nombre
-- de la tabla pokemones
FROM   pokemones
WHERE  rownum <= 10;
```

Resultando:

```
NOMBRE    |LARGO_DEL_NOMBRE|
----------|----------------|
Bulbasaur |               9|
Ivysaur   |               7|
Venusaur  |               8|
Charmander|              10|
Charmeleon|              10|
Charizard |               9|
Squirtle  |               8|
Wartortle |               9|
Blastoise |               9|
Caterpie  |               8|
```

Ahora veamos el comando SUM, como su nombre lo indica, nos permite sumar valores numéricos de la siguiente manera:

Si, para el ejercicio anterior, quisiéramos saber cuánto suma el largo de los nombres de los primeros 10 pokemones, haríamos lo siguiente:

```
SELECT Sum(Length(nombre)) AS suma_de_nombres
FROM   pokemones
WHERE  rownum <= 10;
```

Resultando:

```
SUMA_DE_NOMBRES|
---------------|
             87|
```

Como ves, también podemos combinar distintos comandos para obtener la información que necesitamos.

Ahora, si queremos saber cuantos pokemones son de tipo agua, podemos usar la función COUNT y aplicar lo siguiente:

```
SELECT Count (pokedex) AS total
FROM   pokemones
WHERE  tipo1 = 'agua' OR tipo2 = 'agua';
```

Resultado

```
TOTAL|
-----|
   32|
```

Agrupación con GROUP BY

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

La instrucción GROUP BY agrupa filas que tienen los mismos valores en filas de resumen, como "buscar el número de clientes en cada país". La instrucción GROUP BY a menudo se usa con funciones agregadas (COUNT, MAX, MIN, SUM) para agrupar el conjunto de resultados por una o más columnas.

```
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(*) AS total
FROM   pokemones
GROUP  BY tipo1;
```

Resultando:

```
TIPO1     |TOTAL|
----------|-----|
fuego     |   12|
lucha     |    7|
agua      |   28|
veneno    |   14|
planta    |   12|
bicho     |   12|
electrico |    9|
tierra    |    8|
hada      |    2|
hielo     |    2|
psiquico  |    8|
dragon    |    3|
normal    |   22|
roca      |    9|
fantasma  |    3|
```

Podemos ver que a diferencia de otros motores, Oracle no posee un orden predeterminado.

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 con "DESC" o ascendente con "ASC", ubicando el comando después de la columna seleccionada para ordenar las filas. Por ejemplo, si queremos ver las filas ordenadas con los nombres de los pokemones **desde la Z hasta la A**, debemos hacer lo siguiente:

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

Resultando:

```
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    |        |
... 151 filas en total.
```

Creando indices

Cuando revisamos un libro y necesitamos buscar un capítulo en particular o bien buscar sobre alguna temática, acostumbramos a usar su índice para poder llegar de manera más rápida a esta información, esto ocurre de manera similar en una base de datos. Cada vez que se realiza una consulta a una base de datos SQL, si esta no está indexada la búsqueda será fila por fila hasta conseguir el dato deseado, entenderás que esto tiene un costo de cómputo y un tiempo de espera.

Cuando creamos un índice en una tabla de la base de datos, lo que hacemos es agilizar el tiempo de respuesta de esta búsqueda pues preparamos al motor de base de datos para próximas búsquedas relacionadas con las columnas que tengan índices.

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

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

Siguiendo con nuestro videojuego y considerando que cuando buscamos pokemones lo haremos mayormente por el nombre, agreguemos a esa columna un índice con la siguiente instrucción SQL

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

Importante destacar que el nombre del índice puede ser el que quieras, no obstante muchos programadores están acostumbrados a poner “index_” antes del nombre de la columna.

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

```
SELECT  index_name
FROM    all_indexes
WHERE   table_name = nombre de tabla;
```

Para el caso de nuestro ejemplo la instrucción quedaría de la siguiente manera:

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

Resultando:

```
INDEX_NAME           |
---------------------|
SYS_C006987          |
INDEX_POKEMON_NOMBRE|
```

Podemos ver que entre otros índices que pueda contener esta tabla (pueden variar) es este caso encontraremos nuestro índice recién creado. Notamos que el índice `SYS_C006987` es del tipo único, ya que tenemos un primary key. Probar agregando el campo UNIQUENESS a la consulta para ver el resultado.

Eliminando índices

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

```
drop index nombre_indice;
```

Es importante entender que agregar índices a una o varias columnas **no generan ningún cambio visua**l, pues su uso es meramente funcional y estructural, nos sirve para optimizar las búsquedas y por consecuencia los tiempos de respuesta. Una consulta normal en una base de datos masiva puede durar hasta 4 segundos, que son equivalentes a 4000 ms, no obstante si la consulta incluye una columna con índice el tiempo de respuesta puede llegar a ser 60ms, es decir casi 70 veces más rápido.

Ejercicio guiado - Pongamos a prueba los conocimientos

La empresa japonesa Mawashi Cars Spa, hará una re-inauguración muy pronto y ha decidido dejar de registrar todos los movimientos en excel y empezar a usar el software que compró el año pasado, pero nunca estrenó, sin embargo, cuando lo intentó utilizar se llevó la sorpresa de que la aplicación necesita conectarse a una base de datos. La empresa se contactó con un desarrollador para esto y le envió un archivo en formato csv que exportaron de excel con unos pocos registros de autos para que proceda con la creación de la base de datos y pueda hacer las pruebas que necesite. En el apoyo lectura de esta sesión encontrarás el fichero .csv con la data de los autos.

Luego del levantamiento de requerimientos que le hizo el desarrollador a la empresa se concluyó que necesitaremos crear las siguientes tablas con los siguientes campos:

Autos

- Id.
- Marca.
- Modelo.
- Año.
- Color.

Ventas

- Fecha.
- Id Auto.

- Cliente.
- Referencia.
- Cantidad.

Las pruebas a realizar deben ser:

1. Creación de la base de datos.
2. Conexión con la base de datos.
3. Crear las tablas Autos y ventas enlazadas por claves primaria-foranea.
4. Importar el archivo autos.csv en la tabla Autos.
5. Verificar la carga exitosa de la data en la tabla Autos.
6. Hacer una consulta con Alias.
7. Realizar 2 inserciones a la tabla Ventas.
8. Realizar una consulta con la función SUM.
9. Agregar una columna a una tabla.
10. Hacer una actualización de información a una tabla.
11. Agregar un registro a la tabla Autos.
12. Agrupar columnas en una tabla.
13. Ejecutar una consulta con ORDER BY.
14. Agregar dos índices a dos columnas.
15. Eliminar un índice a una columna.

Pasos a seguir

Paso 1: Creación del usuario y base “mawa shicars”.

```
create user mawashicars identified by 1234;
grant all privileges to mawashicars;
```

Paso 2: Conexión a la base de datos “mawashicars”.

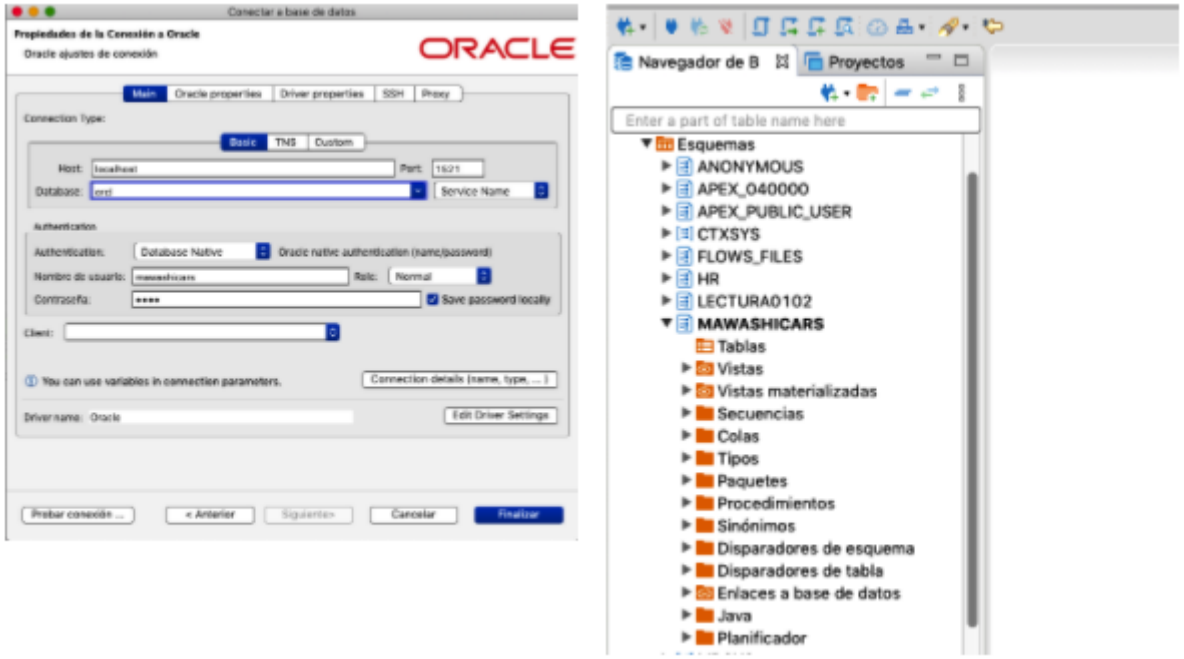

Imagen 13. Base de datos mawashicars.

Paso 3: Crear las tablas “Autos” y “Ventas” con el id en “autos” como clave primaria y “id_auto” en “Ventas” como clave foránea.

```
CREATE TABLE autos
  (
     id     NUMBER,
     marca  VARCHAR2(25),
     modelo VARCHAR2(25),
     anio   VARCHAR2(4),
     color  VARCHAR2(25),
     PRIMARY KEY(id)
  );
```

```
CREATE TABLE ventas
  (
     fecha      VARCHAR2(20),
     id_auto    NUMBER,
     cliente    VARCHAR2(25),
     referencia NUMBER,
     cantidad   NUMBER(10, 2),
     FOREIGN KEY (id_auto) REFERENCES autos(id)
  );
```

Paso 4: Importación del archivo autos.csv en la tabla Autos.

Repetimos lo que vimos al importar un csv, pero esta vez aplicado a nuestro archivo csv.

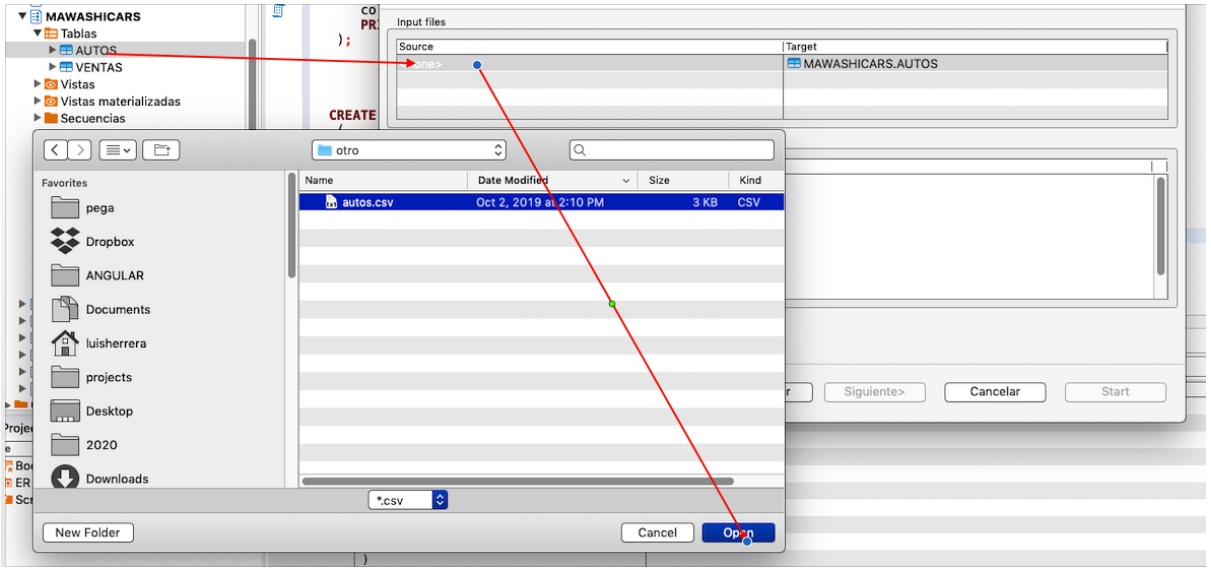

Imagen 14. Carga autos.

Paso 5: Consultar la tabla Autos para verificar que se cargaron exitosamente los datos.

```
SELECT id,
       marca,
       modelo,
       anio,
       color
FROM   autos;
```

Paso 6: Consultar las columnas marca, modelo y color de la tabla Autos usando las alias “mar”, “mod” y “c” respectivamente y la tabla como “a”.

```
SELECT a.marca  AS mar,
       a.modelo AS mod,
       a.color  AS c
FROM   autos a;
```

Paso 7: Registrar 2 ventas.

```
INSERT INTO ventas
            (fecha,
             id_auto,
             cliente,
             referencia,
             cantidad)
VALUES      ('2019-02-20',
             2,
             'Alexander Cell',
             54322,
             2000000);
```

```
INSERT INTO ventas
            (fecha,
             id_auto,
             cliente,
             referencia,
             cantidad)
VALUES      ('2009-03-14',
             3,
             'Katherine Smith',
             12325,
             1500000);
```

Paso 8: Sumar la columna “cantidad” en la tabla Ventas con el alias “total”.

```
SELECT Sum(cantidad) AS Total
FROM   ventas;
```

Paso 9: Agregar la columna “precio” a la tabla Autos.

```
ALTER TABLE autos
  ADD precio NUMBER(10, 2);
```

Paso 10: Actualizar los precios a todos los Autos.

```
UPDATE Autos SET precio=15000000 WHERE id=1;
UPDATE Autos SET precio=15000000 WHERE id=2;
UPDATE Autos SET precio=2000000 WHERE id=3;
UPDATE Autos SET precio=15000000 WHERE id=4;
UPDATE Autos SET precio=25000000 WHERE id=5;
```

Paso 11: Agregar un auto a la tabla “Autos” de color Blanco.

```
INSERT INTO autos
  (id, marca, modelo, anio, color, precio)
VALUES
  (6, 'Ferrari', 'Sport', '2002', 'Blanco', 50000000);
```

Paso 12: Agrupar la tabla Autos por columna de precio.

```
SELECT precio, COUNT(*) FROM Autos GROUP BY precio;
```

Paso 13: Consultar la tabla “Autos” ordenada por los precios de menor a mayor.

```
SELECT *
FROM Autos
ORDER BY precio ASC;
```

Paso 14: Crear un índice a la columna “id_auto” de la tabla ventas y a la columna “id” de la tabla “Autos”.

```
CREATE INDEX index_id_auto on Ventas(id_auto);
CREATE INDEX index_id on Autos(id);
```

Paso 15: Eliminar el índice en la tabla Autos.

```
drop index index_id;
```