Realizando consultas

¡Comencemos!

¿Qué aprenderás?

- 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.

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:

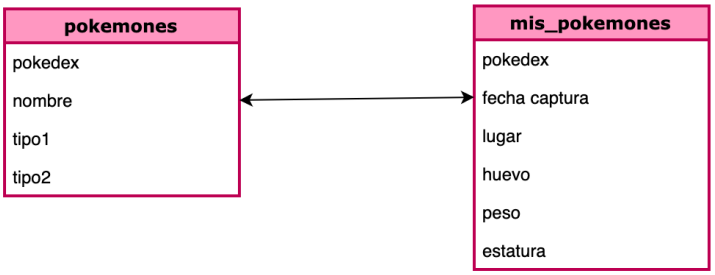

Imagen 1. Modelo relacional.
Fuente: Desafío Latam.

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 pokemon 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 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.

Con la siguiente instrucción creamos la tabla “Pokemones” y su estructura

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

pokedex	nombre	tipo1	tipo2

Tabla 1. Estructura de tabla Pokemones.
Fuente: Desafío Latam.

Con la siguiente instrucción creamos la tabla “mis_pokemones” y su estructura correspondiente

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

pokedex	fecha_captura	lugar	huevo	peso	estatura

Tabla 2. Estructura tabla mis_pokemones.
Fuente: Desafío Latam.

Importando datos de un .csv

Un archivo CSV 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 .

- Abrir el archivo en un editor de texto, de recomendación Visual Studio Code o Atom.
- Si las filas están encerradas entre comillas, elimínelas.
- Crear la tabla a la cual desea agregar el archivo csv.

Utilizar en la consola de PostgreSQL el siguiente comando:

```
\copy nombre_tabla FROM 'directorio/archivo.csv' csv [header];
```

O también:

```
COPY nombre_tabla FROM 'directorio/archivo.csv' csv header;
```

Si no agregas “csv header” podrías recibir un mensaje como este:

```
ERROR: error de sintaxis de entrada no es válida para tipo integer:
pokedex;
```

Esto puede significar que los datos que intentas exportar no están exactamente formateados con la estructura de la tabla, pero no te preocupes, esto se soluciona con el “csv header”.

Cuando en una sentencia se coloca un parámetro entre corchetes, es porque este valor puede ser opcional.

Nota: [header] es opcional.

- Si su archivo .csv tiene los nombres de las columnas en la primera fila, “[header]” reemplácelo por “header”, es decir, sin los corchetes.
- De ser el caso contrario, deje en blanco ese espacio.

Como si fuese poco, es posible hacer lo mismo pero en sentido contrario, es decir, a partir de una tabla generar un fichero csv con la información contenida. Este proceso es común

verlo al momento de migrar una base de datos de un servidor a otro. El comando que debes usar para conseguirlo es el siguiente:

```
\copy nombre_tabla TO 'directorio/archivo.csv' csv header;
```

Cargando la data

Ahora que hemos aprendido cómo importar datos de un csv y tenemos creadas nuestras tablas, vamos a importar los valores de los archivos pokemonesKanto.csv y mis_pokemones.csv que conseguirás en el apoyo lectura de esta sesión. Para importar la información procede con el siguiente comando:

```
\copy pokemones FROM 'directorio/pokemonesKanto.csv' csv header;
```

Donde dejaremos header como parámetro, ya que si revisamos nuestro archivo tiene la primera fila con los nombres de las columnas, y haremos lo mismo para la otra tabla.

```
\copy mis_pokemones FROM 'directorio/mis_pokemones.csv' csv header;
```

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. 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 busqueda 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 condicion;
```

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 3. Operadores condicionales.
Fuente: Desafío Latam.

Por último, si quisiéramos de 100 datos obtener solo 20, tenemos a disposición en el lenguaje SQL el comando LIMIT 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 LIMIT cantidad;
```

Ejemplo 2:

```
SELECT * FROM nombre_tabla LIMIT 20;
```

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

```
SELECT * FROM pokemones;
```

Obtendremos algo como la tabla 4

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	

Tabla 4. Registros de tabla pokemones.
Fuente: Desafío Latam.

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

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

Obteniendo una respuesta como la tabla 5:

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

Tabla 5. Pokemones de tipo fuego.
Fuente: Desafío Latam.

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.

Sintaxis:

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

Ejemplo:

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

Si ejecutas esta instrucción obtendrías la siguiente tabla.

n	pkd
Bulbasau	1
Ivysaur	2
...	
Mewtwo	151

Tabla 6. Tabla de pokemones con Alias.
Fuente: Desafío Latam.

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_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

Tabla 7. Largo de nombre.
Fuente: Desafío Latam.

Para 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,LENGTH(nombre)
-- y le asignaremos el nombre largo_de_nombre
AS largo_del_nombre
-- de la tabla pokemones
FROM pokemones LIMIT 10;
```

Obteniendo una tabla como la tabla 8.

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)

Tabla 8. Nombre y largo del nombre.
Fuente: Desafío Latam.

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))
-- y le asignaremos el nombre suma_de_nombres
AS suma_de_nombres
-- de la tabla pokemones
FROM pokemones LIMIT 10;
```

Obteniendo lo siguiente:

suma_de_nombres
1100

(1 fila)

Tabla 9. Suma de nombres.
Fuente: Desafío Latam.

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';
```

Obteniendo la tabla 10:

total
32

(1 fila)

Tabla 10. Suma de nombres.
Fuente: Desafío Latam.

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

Obteniendo lo siguiente:

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)

Tabla 11. Cantidad de pokemones por tipo1.
Fuente: Desafío Latam.

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 descendente.
- 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;
```

Obteniendo entonces:

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	

Tabla 12. Pokemones ordenados alfabéticamente por nombre.
Fuente: Desafío Latam.

Creando índices

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, y es más aún aplicado cuando se almacenan grandes cantidades de información, me refiero a miles, cientos de miles y hasta millones. 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 a 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 * FROM pg_indexes WHERE tablename = nombre_tabla;
```

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

```
SELECT * FROM pg_indexes WHERE tablename = 'pokemones';
```

Al buscar en la tabla de pokemones, observamos lo siguiente:

schemaname	tablename	indexname	tablespace	indexdef
public	pokemones	pokemones_pkey		CREATE UNIQUE INDEX pokemones_pkey ON public.pokemones USING btree (pokedex)
public	pokemones	index_pokemon_nombre		CREATE INDEX index_pokemon_nombre ON public.pokemones USING btree (nombre)

Tabla 13. Índices de las tablas.
Fuente: Desafío Latam.

Podemos ver que ya existe un índice de carácter único sobre la clave primaria de la columna pokédex. Cuando se crea una clave primaria, PostgreSQL crea automáticamente un índice sobre esta columna.

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 visual, 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 reinauguración 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:

id	Marca	Modelo	Año	Color

Tabla 14. Modelo de la tabla Autos.
Fuente: Desafío Latam.

Fecha	id_auto	Cliente	Referencia	Cantidad

Tabla 15. Modelo de la tabla Ventas.
Fuente: Desafío Latam.

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. Agregar un registro a la tabla Autos
11. Hacer una actualización de información a una tabla.
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 de la base de datos “mawashicars”.

```
CREATE DATABASE mawashicars ;
```

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

```
\c 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 INT,
marca VARCHAR(25),
modelo VARCHAR(25),
año VARCHAR(4),
color VARCHAR(25),
PRIMARY KEY(id)
);
```

```
CREATE TABLE Ventas(
fecha VARCHAR(20),
id_auto INT,
cliente VARCHAR(25),
referencia INT,
cantidad FLOAT,
FOREIGN KEY (id_auto) REFERENCES
autos(id)
);
```

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

```
\copy Autos FROM 'directorio/autos.csv' csv [header];
```

Recuerda que el directorio dependerá del sistema de archivos en tu sistema operativo.

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

```
SELECT * 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 AS 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 FLOAT;
```

- ***Paso 10:*** Agregar un auto a la tabla “Autos” de color Blanco.

```
INSERT INTO Autos
      (id, marca, modelo, año, color, precio)
    VALUES (5, 'ferrari', 'Sport' , '2002', 'Blanco', 50000000);
```

- ***Paso 11:*** 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=15000000 WHERE id=5;
```

- ***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;
```