Reto Zoológico

Presentado por:

Juan David Cardona Velasquez

Presentado a:

Juan Pineda

Sofka University

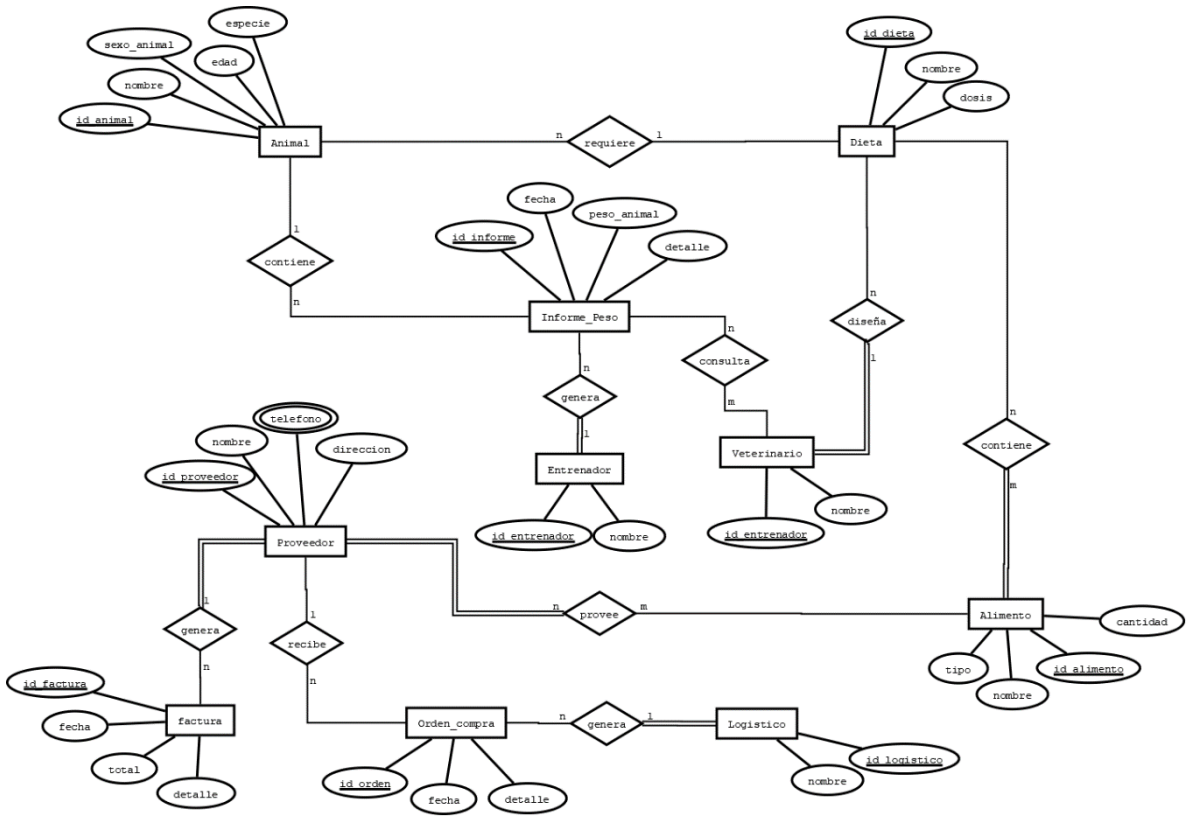
18/02/2023

Reto Zoológico

El reto que se me asigno fue el reto B que corresponde al reto del ZOO

En el reto del zoológico nos pedía crear un esquema de base de datos que le permitiera al gerente del zoológico saber el alimento que están consumiendo los animales del zoológico. Por lo tanto, no se tuvieron en cuenta algunas entidades que mencionaba el enunciado como lo son CUIDADOR y VISITANTE ya que no se consideran necesarias para el fin principal de esta base de datos.

A continuación, se muestra como quedo el Modelo Entidad Relación y posteriormente se explica el por qué de este mismo.



Se definen las funcionalidades principales requeridas y, se plantea de la siguiente manera:

Un entrenador genera un informe de peso que básicamente es pesar al animal y registrar una fecha para llevar un control y un detalle de este pesaje. El veterinario para crear las dietas de cada animal accederá a estos informes que crearon los entrenadores y, de esta manera generar una dieta para estos animales.

Esta entidad dieta se genera teniendo en cuenta el alimento y, con el informe que genero el entrenador tiene acceso a la información del animal a parte del peso de este mismo para así generar una dieta adecuada.

También se cuenta con un empleado logístico que genera ordines de compra indicando en el detalle que alimentos se requieren en el zoológico. Esta orden de compra es recibida por el proveedor y este provee los alimentos de la orden de compra que tenga disponibles y, genera una factura que al final de cada mes se pagara por el zoológico.

Al tener esta entidad factura es posible saber el consume de alimento cada mese n el zoológico, ya que estas facturas tienen un detalle donde se describe la cantidad de alimento que se facturo y de esta manera saber también que facturas fueron las más costosas.

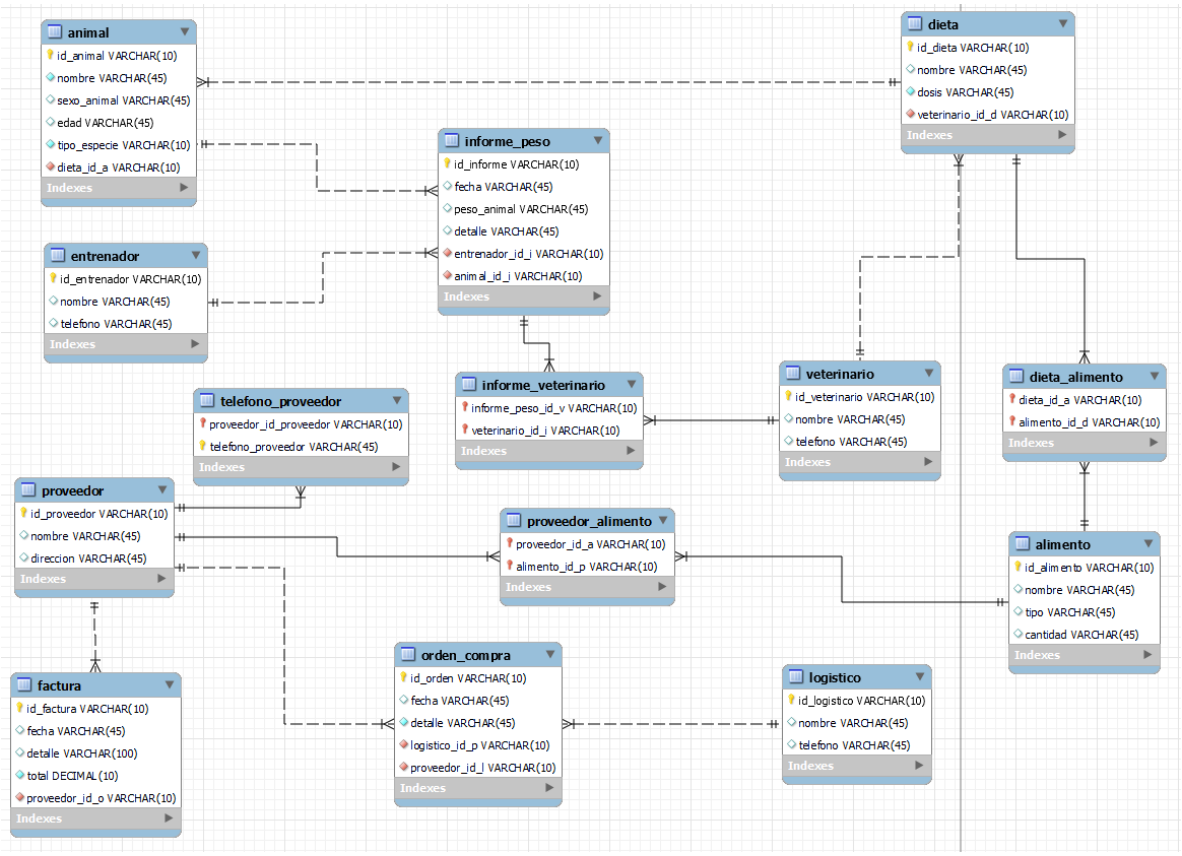
*__Lectura de Cardinalidades__:

- 1 entrenador genera 1 o muchos informes de peso y, 1 informe de peso es generado por 1 entrenador (1: N).
- 1 informe de peso contiene 1 animal y, 1 animal este contenido en 1 informe de peso (N:1).
- 1 veterinario consulta 1 o muchos informes de peso y, 1 informe de peso es consultado por 1 o muchos veterinarios (N:M).
- 1 veterinario diseña 1 o muchas dietas y, 1 dieta es generada por 1 veterinario (1: N).
- 1 dieta contiene 1 o muchos alimentos y, 1 alimento es contenido por 1 o muchas dietas (N:M).
- 1 dieta es requerida por 1 o muchos animales y, 1 animal requiere 1 dieta.
- 1 proveedor provee 1 o muchos alimentos y, 1 alimento es proveído por 1 o muchos proveedores (N:M).
- 1 logístico genera 1 o muchas ordines de compra y, 1 orden de compra es generada por 1 logístico (1:N).
- 1 proveedor genera 1 o muchas facturas y, 1 factura es generada por 1 proveedor (1: N).

*__Modelo Relacional__

Se crearon las distintas entidades definidas en el modelo ER y, teniendo en cuenta las relaciones que se habían definido allí, se definieron nuevos atributos en algunas tablas como llaves foráneas para cumplir con estas relaciones. también para las tablas de muchos a muchos se crearon nuevas tablas intermedias hacienda referencia a las claves primarias de las tablas que se relacionan.

Por último, para los atributos multivaluados que en este caso solo se tiene el teléfono del proveedor se creó una nueva tabla para que de esta manera un proveedor pueda tener varios teléfonos asociados.



***Normalización:**

- Se cumple con la primera forma normal ya que los atributos de las entidades son atómicos y, también se garantiza que no existirán registros repetidos.
- Se cumple con la segunda forma normal ya que se cumple con la primera y, los atributos dependen únicamente de su llave primaria.
- Se cumple con la tercera forma normal ya que se cumplen con las primeras 2 formas normales y , al cambiar un atributo de 1 tabla, este cambio no modifica el valor de otra columna de esa tabla.

***Creación Base de datos con Sentencias:**

Ahora se mostrarán algunos pantallazos de la creación del esquema de la base de datos definiendo todas las tablas mostradas en el modelo relacional, y definiendo las llaves primarias y foráneas necesarias para el funcionamiento de esta. El script que genera toda la estructura de la base de datos tal como se visualiza en el modelo relacional se encuentra dentro del repositorio bajo el nombre de EsquemaZoo.sql

Al ejecutar este script ya se puede empezar a inyectar datos dentro de la base de datos para probar que las relaciones funcionan de manera correcta.

```sql
CREATE DATABASE zoologico;

use zoologico;

#Tabla Veterinario
CREATE TABLE veterinario (
  id_veterinario VARCHAR(10) NOT NULL,
  nombre VARCHAR(45) NULL,
  telefono VARCHAR(45) NULL,
  PRIMARY KEY (id_veterinario)
);

#Tabla dieta
CREATE TABLE dieta (
  id_dieta VARCHAR(10) NOT NULL,
  nombre VARCHAR(45) NULL,
  dosis VARCHAR(45) NOT NULL,
  veterinario_id_d VARCHAR(10) NOT NULL,
  PRIMARY KEY (id_dieta),
  FOREIGN KEY (veterinario_id_d) REFERENCES veterinario(id_veterinario)
  );

#Tabla animal
CREATE TABLE animal (
  id_animal VARCHAR(10) NOT NULL,
  nombre VARCHAR(45) NOT NULL,
  sexo_animal VARCHAR(45) NULL,
  edad VARCHAR(45) NULL,
  tipo_especie VARCHAR(25) NOT NULL,

#Tabla entrenador
CREATE TABLE entrenador (
  id_entrenador VARCHAR(10) NOT NULL,
  nombre VARCHAR(45) NULL,
  telefono VARCHAR(45) NULL,
  PRIMARY KEY (id_entrenador)
  );

#Tabla de Proveedor
CREATE TABLE proveedor (
  id_proveedor VARCHAR(10) NOT NULL,
  nombre VARCHAR(45) NULL,
  direccion VARCHAR(45) NULL,
  PRIMARY KEY (id_proveedor)
  );

#Tabla de alimento
CREATE TABLE alimento (
  id_alimento VARCHAR(10) NOT NULL,
  nombre VARCHAR(45) NULL,
  tipo VARCHAR(45) NULL,
  cantidad VARCHAR(45) NULL,
  PRIMARY KEY (id_alimento)
  );

CREATE TABLE factura (
  id_factura VARCHAR(10) NOT NULL,
  fecha VARCHAR(45) NULL,
  detalle VARCHAR(100),
  total DECIMAL(10) NOT NULL,
  proveedor_id_o VARCHAR(10) NOT NULL,
  PRIMARY KEY (id_factura),
  FOREIGN KEY (proveedor_id_o)REFERENCES proveedor(id_proveedor)
  );

#Tabla de logistico
CREATE TABLE logistico(
  id_logistico VARCHAR(10) NOT NULL,
  nombre VARCHAR(45) NULL,
  telefono VARCHAR(45) NULL,
  PRIMARY KEY (id_logistico)
  );

#Tabla de orden_compra
CREATE TABLE orden_compra(
  id_orden VARCHAR(10) NOT NULL,
  fecha VARCHAR(45) NULL,
  detalle VARCHAR(45) NOT NULL,
  logistico_id_p VARCHAR(10) NOT NULL,
  proveedor_id_l VARCHAR(10) NOT NULL,
  PRIMARY KEY (id_orden),
  FOREIGN KEY (logistico_id_p)REFERENCES logistico(id_logistico),
  FOREIGN KEY (proveedor_id_l)REFERENCES proveedor(id_proveedor)
  );

#Tabla dieta_alimento
CREATE TABLE dieta_alimento(
  dieta_id_a VARCHAR(10) NOT NULL,
  alimento_id_d VARCHAR(10) NOT NULL,
  PRIMARY KEY (dieta_id_a, alimento_id_d),
  FOREIGN KEY (dieta_id_a)REFERENCES dieta(id_dieta),
  FOREIGN KEY (alimento_id_d)REFERENCES alimento(id_alimento)
  );

#Tabla proveedor_alimento
CREATE TABLE proveedor_alimento(
  proveedor_id_a VARCHAR(10) NOT NULL,
  alimento_id_p VARCHAR(10) NOT NULL,
  PRIMARY KEY (proveedor_id_a,alimento_id_p),
  FOREIGN KEY (proveedor_id_a)REFERENCES proveedor(id_proveedor),
  FOREIGN KEY (alimento_id_p)REFERENCES alimento(id_alimento)
  );

#Tabla Informe Peso
CREATE TABLE informe_peso(
  id_informe VARCHAR(10) NOT NULL,
  fecha VARCHAR(45) NULL,
  peso_animal VARCHAR(45) NULL,
  detalle VARCHAR(45) NULL,
  entrenador_id_i VARCHAR(10) NOT NULL,
  animal_id_i VARCHAR(10) NOT NULL,
  PRIMARY KEY (id_informe),
  FOREIGN KEY(entrenador_id_i)REFERENCES entrenador(id_entrenador),
  FOREIGN KEY (animal_id_i)REFERENCES animal(id_animal)
  );

#Tabla informe_veterinario
CREATE TABLE informe_veterinario(
  informe_peso_id_v VARCHAR(10) NOT NULL,
  veterinario_id_i VARCHAR(10) NOT NULL,
  PRIMARY KEY (informe_peso_id_v,veterinario_id_i),
  FOREIGN KEY (informe_peso_id_v)REFERENCES informe_peso(id_informe),
  FOREIGN KEY (veterinario_id_i)REFERENCES veterinario(id_veterinario)
  );

#Tabla telefono_proveedor
CREATE TABLE telefono_proveedor(
  proveedor_id_proveedor VARCHAR(10) NOT NULL,
  telefono_proveedor VARCHAR(45) NOT NULL,
  PRIMARY KEY (proveedor_id_proveedor,telefono_proveedor),
  FOREIGN KEY (proveedor_id_proveedor)REFERENCES proveedor(id_proveedor)
  );
```

***Consultas SQL**

Algunas de las consultas que se crearon son consultas sencillas pensando en que posteriormente se pueden usar desde la conexión de java y, teniendo en cuenta que las consultas más interesantes se definen posteriormente en las vistas.

```
#1.Obtener todos los animales registrados en la base de datos:
SELECT * FROM animal;
```

Esta primera consulta nos devuelve todos los registros almacenados en animal.

id_animal	nombre	sexo_animal	edad	tipo_especie	dieta_id_a
A1	Hugo First	female	14	Werewolf	D1
A10	Barb E. Cue	female	8	Wham-a-Wham	D10
A11	Alex Blaine Layder	female	1	Grottore	D11
A12	Carrie A. Tune	female	8	Wyvern	D12
A13	Alex Blaine Layder	male	15	Botchling	D13
A14	Bill Loney	female	5	Umbra	D14
A15	Dusty Carr	male	1	Grottore	D15
A16	Lake Speed	female	4	Imp	D16
A17	Harry Pitts	female	13	The Caretaker	D17
A18	Mark A. Roni	male	14	Howler	D18

animal 1

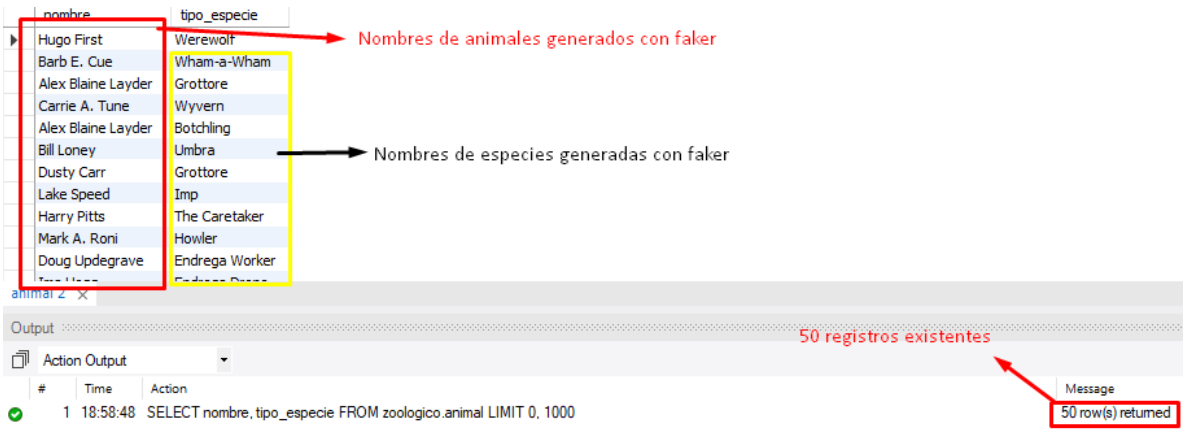
Output

Action Output

#	Time	Action	Message
178	18:33:02	SELECT * FROM zoologico.alimento LIMIT 0, 1000	50 row(s) returned
179	18:34:15	SELECT * FROM zoologico.proveedor_alimento LIMIT 0, 1000	50 row(s) returned
180	18:35:29	SELECT * FROM zoologico.telefono_proveedor LIMIT 0, 1000	0 row(s) returned
181	18:38:08	SELECT * FROM zoologico.telefono_proveedor LIMIT 0, 1000	50 row(s) returned
182	18:51:14	SELECT * FROM animal LIMIT 0, 1000	50 row(s) returned

```
#2.Obtener el nombre y el tipo de especie de los animales registrados
SELECT nombre, tipo_especie FROM animal;
```

Esta segunda consulta solo nos devuelve el nombre del animal “alias” y la especie con la que se identifica.



nombre	tipo_especie
Hugo First	Werewolf
Barb E. Cue	Wham-a-Wham
Alex Blaine Layder	Grottore
Carrie A. Tune	Wyvern
Alex Blaine Layder	Botchling
Bill Loney	Umbra
Dusty Carr	Grottore
Lake Speed	Imp
Harry Pitts	The Caretaker
Mark A. Roni	Howler
Doug Updegrave	Endrega Worker

animal 2

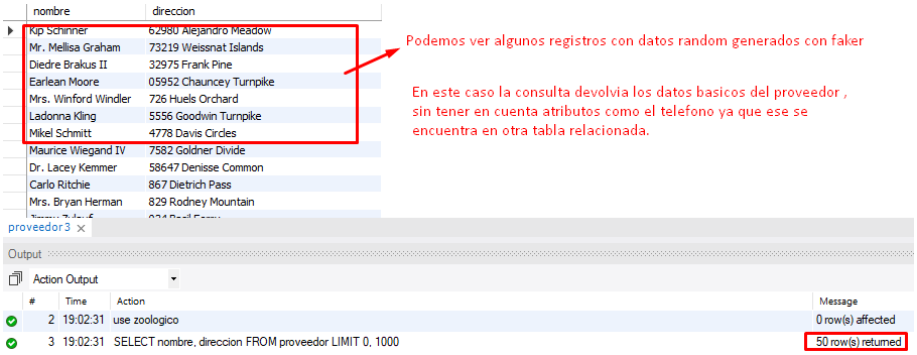
Output

Action Output

#	Time	Action	Message
1	18:58:48	SELECT nombre, tipo_especie FROM zoologico.animal LIMIT 0, 1000	50 row(s) returned

```sql
#3.Obtener el nombre y la dirección de todos los proveedores registrados en la base de datos:
SELECT nombre, direccion FROM proveedor;
```

Esta consulta nos sirve para obtener una información básica de los proveedores.



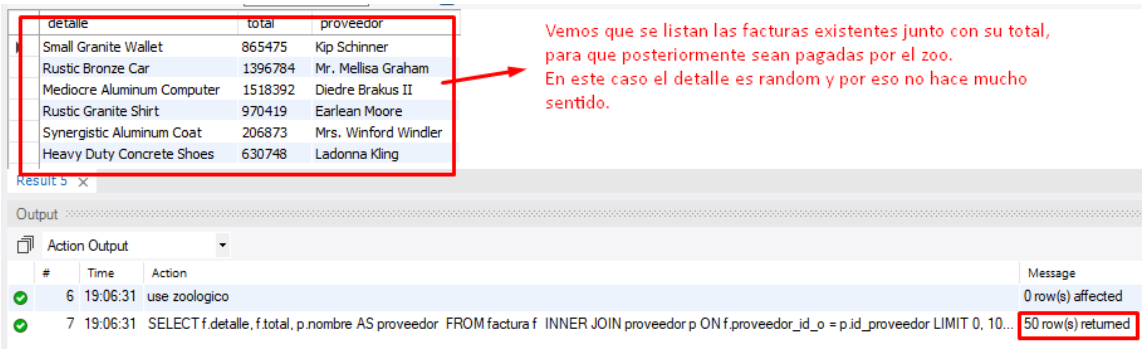

```sql
#4.Obtener los detalles de la dieta que un animal en particular está siguiendo:
SELECT d.nombre, d.dosis, v.nombre AS nombre_veterinario, v.telefono AS telefono_veterinario
FROM dieta d
INNER JOIN veterinario v ON d.veterinario_id_d = v.id_veterinario
INNER JOIN animal a ON d.id_dieta = a.dieta_id_a
WHERE a.nombre = 'Mufasa';
```

Esta consulta nos sirve para averiguar la dieta que esta siguiendo un animal en específico, por lo que en la condición del where se le pasa el nombre “alias” del cual se quiere obtener esta información. También nos devuelve la información del veterinario que diseño esta dieta.

nombre_dieta	dosis	nombre_veterinario	telefono_veterinario
Dieta Herbivora	56g/dia	Tobias	1-568-472-1628

```sql
#5. Consulta para obtener el detalle y el costo total de todas las facturas, junto con el nombre del proveedor que las emitió:
SELECT f.detalle, f.total, p.nombre AS proveedor
FROM factura f
INNER JOIN proveedor p ON f.proveedor_id_o = p.id_proveedor;
```

Esta consulta nos brinda información importante de la factura y también nos proporciona el nombre del proveedor que genero esta factura. En este caso se listan todas las facturas que ha generado cada proveedor y, esto es de mucha ayuda para saber los costos que esta generando el zoológico en compra de alimento.



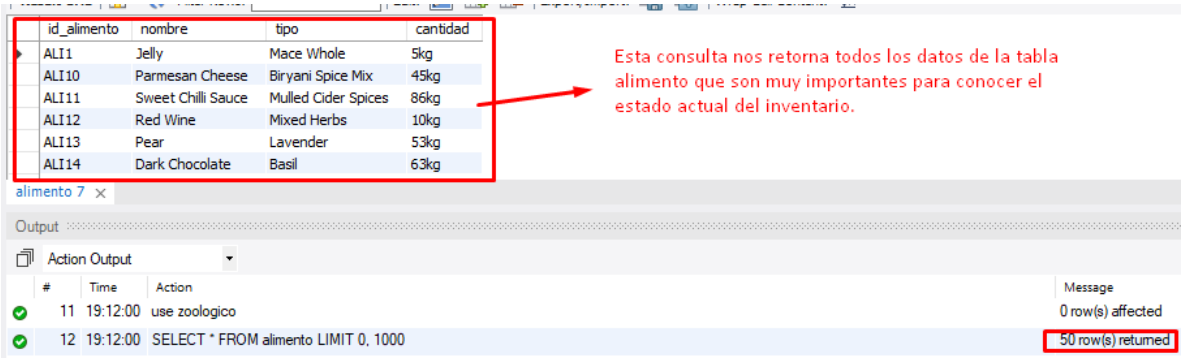
```sql
#6. Consulta para obtener todos los animales de una misma especie
SELECT * FROM animal WHERE animal.tipo_especie = "Werewolf";
```

Esta consulta nos muestra los animales que pertenecen a una especie en específico, en este caso los animales que son de la especie Werewolf que solo es 1.

id_animal	nombre	sexo_animal	edad	tipo_especie	dieta_id_a
A1	Hugo First	female	14	Werewolf	D1

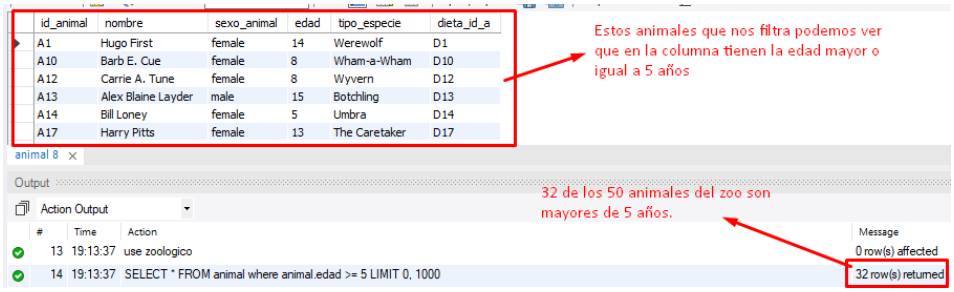
```sql
#7. Consulta para conocer el estado actual del inventario de alimentos
SELECT * FROM alimento;
```

Esta es una consulta simple que nos muestra el stock de alimentos.



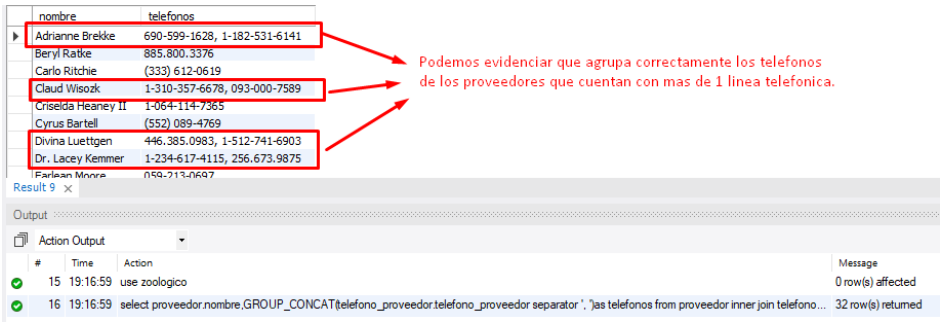

```sql
#8. Consulta para saber que animales son mayores de 5 años de edad
SELECT * FROM animal where animal.edad >= 5;
```

Esta consulta nos sirve para filtrar los animales que son mayores de 5 años.



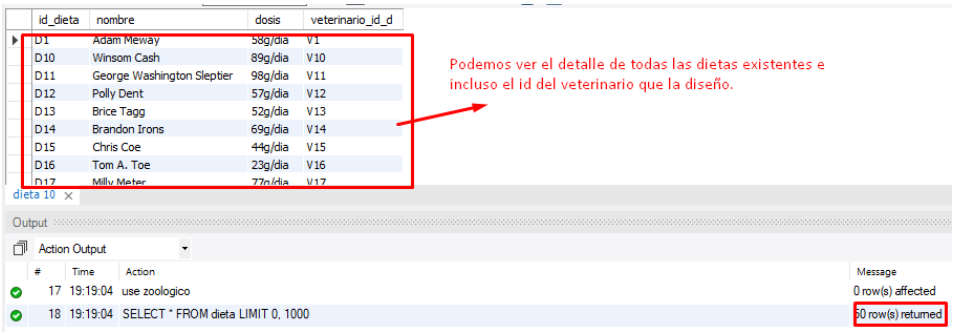

```sql
#9. Consulta para agrupar todos los telefonos de los proveedores
select proveedor.nombre,GROUP_CONCAT(telefono_proveedor.telefono_proveedor separator ', ')as telefonos
from proveedor inner join telefono_proveedor on id_proveedor = telefono_proveedor.proveedor_id_proveedor
group by proveedor.nombre;
```

Esta consulta nos permite agrupar los teléfonos de los proveedores y, entonces de esta forma Podemos tener un proveedor con varios teléfonos asociados que se muestran en la misma columna separados por coma.




```sql
#10.Listado de todas las dietas existentes
SELECT * FROM dieta;
```

La ultima consulta nos permite listar de manera sencilla todas las dietas que existen.



***Vistas**

El script para generar estas vistas se encuentra en el repositorio bajo el nombre de VistasZoo.sql.

```sql
#1. Vista para ver todos los detalles concatenados de las facturas
CREATE VIEW factura_detalles_concatenados AS
SELECT GROUP_CONCAT(detalle SEPARATOR ', ') AS detalles_concatenados
FROM factura;
```

Esto sirve para conocer que es lo que se esta cobrando en las facturas. En este caso al tratarse de un detalle generado de manera aleatoria no se entiende muy bien pero, de esta manera podrían concatenarse detalles de las facturas que expliquen la cantidad del alimento que se facturo .

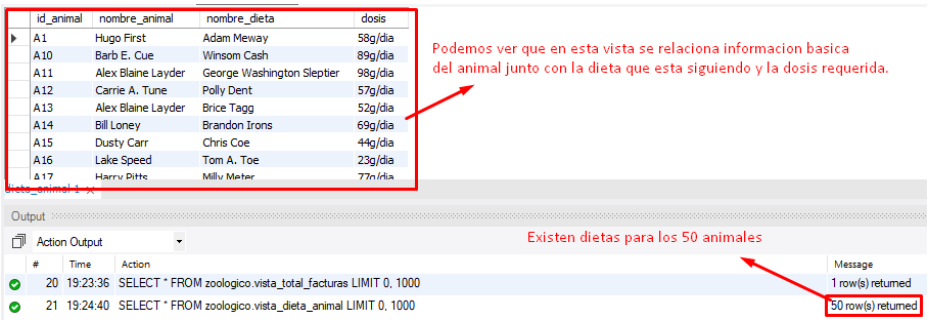
```sql
#2. Vista que me permite saber el total de todas las facturas sumado
CREATE VIEW vista_total_facturas AS
    SELECT
        SUM(total) AS total_facturas
    FROM
        factura;
    SELECT total_facturas
        FROM vista_total_facturas;
```

Esta vista me devuelve únicamente la suma del valor total a pagar de todas las facturas que ha generado el proveedor. En este caso se tenían 50 facturas creadas y, con esta vista se calcula el valor total a pagar de esas 50 facturas.

total_facturas
50366492

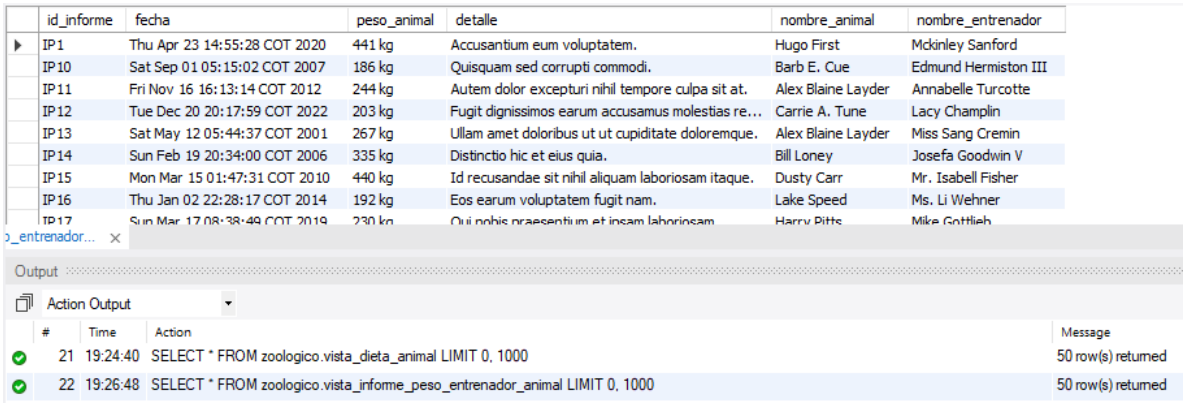
```sql
#3. Vista para conocer que dieta esta llevando cada animal y la dosis correspondiente
CREATE VIEW vista_dieta_animal AS
SELECT a.id_animal, a.nombre AS nombre_animal, d.nombre AS nombre_dieta, d.dosis
FROM animal a
JOIN dieta d ON a.dieta_id_a = d.id_dieta;
```

Esta es una de las vistas que me pareció mas importante ya que puede ser útil para saber que cantidad de alimento esta consumiendo cada animal del zoológico.




```sql
#4. Vista para conocer los detalles del informe de peso y,el animal, el entrenador que registro el informe.
CREATE VIEW vista_informe_peso_entrenador_animal AS
SELECT i.id_informe, i.fecha, i.peso_animal, i.detalle, a.nombre AS nombre_animal, e.nombre AS nombre_entrenador
FROM informe_peso AS i
JOIN animal AS a ON i.animal_id_i = a.id_animal
JOIN entrenador AS e ON i.entrenador_id_i = e.id_entrenador;
```

Esta vista permite llevar un control de los informes que han realizado con los pesajes de los animales y mostrar el entrenador que realizo este pesaje.



id_informe	fecha	peso_animal	detalle	nombre_animal	nombre_entrenador
IP1	Thu Apr 23 14:55:28 COT 2020	441 kg	Accusantium eum voluptatem.	Hugo First	Mckinley Sanford
IP10	Sat Sep 01 05:15:02 COT 2007	186 kg	Quisquam sed corrupti commodi.	Barb E. Cue	Edmund Hermiston III
IP11	Fri Nov 16 16:13:14 COT 2012	244 kg	Autem dolor excepturi nihil tempore culpa sit at.	Alex Blaine Layder	Annabelle Turcotte
IP12	Tue Dec 20 20:17:59 COT 2022	203 kg	Fugit dignissimos earum accusamus molestias re...	Carrie A. Tune	Lacy Champlin
IP13	Sat May 12 05:44:37 COT 2001	267 kg	Ullam amet doloribus ut ut cupiditate doloremque.	Alex Blaine Layder	Miss Sang Cremin
IP14	Sun Feb 19 20:34:00 COT 2006	335 kg	Distinctio hic et eius quia.	Bill Loney	Josefa Goodwin V
IP15	Mon Mar 15 01:47:31 COT 2010	440 kg	Id recusandae sit nihil aliquam laboriosam itaque.	Dusty Carr	Mr. Isabell Fisher
IP16	Thu Jan 02 22:28:17 COT 2014	192 kg	Eos earum voluptatem fugit nam.	Lake Speed	Ms. Li Wehner
IP17	Sun Mar 17 08:38:49 COT 2019	230 kg	Qui nobis praesentium et ipsam laboriosam	Harry Pitts	Mike Gottlieb

_entrenador...

Output

Action Output

#	Time	Action	Message
21	19:24:40	SELECT * FROM zoologico.vista_dieta_animal LIMIT 0, 1000	50 row(s) returned
22	19:26:48	SELECT * FROM zoologico.vista_informe_peso_entrenador_animal LIMIT 0, 1000	50 row(s) returned

***Procedimientos**

El script para crear los procedimientos siguientes se encuentra en el repositorio bajo el nombre de ProcedimientosZoo.sql.

Estos procedimientos no tienen mayor complejidad y, se crean pensando en la lógica de que se estuviera implementando un servicio, entonces se trata de ser lo mas simple posible para mantener las buenas practicas a la hora de luego llamarlos desde Java.

```
#1. Procedimiento para agregar un nuevo animal
DELIMITER //
CREATE PROCEDURE insertar_animal(IN id VARCHAR(10),IN name_animal VARCHAR(45),IN animal_sexo VARCHAR(45),
  IN edad_a VARCHAR(45),
  IN tipo VARCHAR(10),
  IN dieta_id VARCHAR(10)
)
BEGIN
  INSERT INTO animal VALUES(id, name_animal, animal_sexo, edad_a, tipo, dieta_id);
END
//
DELIMITER ;
call insertar_animal('A51','Grogu','macho','2','Mono','D2');
```

Se crea el procedimiento para agregar un nuevo animal que recibe los parámetros correspondientes a las columnas definidas en nuestra tabla animal; Además luego se define la sentencia insert into que tomara los valores que le ingresaron como parámetro. Para llamarlo se usa call seguido del nombre del procedimiento y los valores de los parámetros.

```
#2. Procedimiento para actualizar la cantidad de un alimento
DELIMITER //
CREATE PROCEDURE actualizar_cantidad_alimento(
  IN id VARCHAR(10),
  IN cant VARCHAR(45)
)
BEGIN
  UPDATE alimento
  SET cantidad = cant
  WHERE id_alimento = id;
END
//
DELIMITER ;
call actualizar_cantidad_alimento('ALI1','300kg');
```

El segundo procedimiento recibe 2 parámetros, un id y una cantidad que posteriormente se actualizara teniendo en cuenta el id que ingreso como parámetro que hace referencia al id de un alimento ya creado.

```
#3. Cambiar la dieta que tiene asignada un animal
DELIMITER //
CREATE PROCEDURE actualizar_dieta_animal(IN id VARCHAR(10),IN dieta_id VARCHAR(10))
BEGIN
  UPDATE animal
  SET dieta_id_a = dieta_id
  WHERE id_animal = id;
END
//
DELIMITER ;
call actualizar_dieta_animal('A51','D10');
```

Este procedimiento permite mediante el id de un animal que ya tiene una dieta establecida cambiarle esta dieta si es necesario. En este caso le actualizamos la dieta al animal que habíamos creado con el primer procedimiento.

```
#4. Procedimiento para eliminar un animal
DELIMITER //
CREATE PROCEDURE eliminar_animal(IN id VARCHAR(10))
BEGIN
  DELETE FROM animal WHERE id_animal = id;
END
//
DELIMITER ;
call eliminar_animal('A51');
```

Por último este cuarto procedimiento permite eliminar el registro de un animal mediante el id que se le ingre como parámetro.

***Triggers**

El script para crear los siguientes triggers se encuentra en el repositorio bajo el nombre de Triggers.sql.

Lo primero que se hizo para que los triggers en este caso funcionen fue crear una tabla llamada control_de_cambios_zoologico .

```
#Creacion tabla para control de cambios
create table control_de_cambios_zoologico(
usuario varchar(45),
accion varchar(45),
fecha datetime default current_timestamp
);
```

Esta tabla ayudara a llevar un control de los cambios que se lleven a cabo en algunas de las tablas que tenemos creadas en nuestra base de datos y, de esta manera tener conocimiento de quien altera nuestros registros o, quien agrega nuevos registros.

```
#1. Trigger para registrar quien agrega un animal
DELIMITER //
create trigger inserto_animal after insert on animal
for each row
begin
    insert into control_de_cambios_zoologico
    values (user(),"AGREGO ANIMAL",now());
end;
//
DELIMITER ;

insert into animal values('A10','Grogu','macho','2','Mono','D002');
```

Para probar este trigger se agrega un nuevo animal con identificador A10 .

	usuario	accion	fecha
	root@localhost	AGREGO ANIMAL	2023-02-17 00:12:36

```
#2. Trigger para registrar quien elimina un medico
DELIMITER //
create trigger elimino_animal after delete on animal
for each row
begin
    insert into control_de_cambios_zoologico
    values (user(),"ELIMINO ANIMAL",now());
end;
//
DELIMITER ;

delete from animal where id_animal = 'A10';
```

El segundo trigger controla quien elimina registros de la tabla animal y, en este caso eliminamos el registro que se había creado para probar el primer trigger.

	usuario	accion	fecha
	root@localhost	AGREGO ANIMAL	2023-02-17 00:12:36
	root@localhost	ELIMINO ANIMAL	2023-02-17 00:16:17

```
#3. Trigger para cuando ingresen nueva dieta
DELIMITER //
create trigger inserto_dieta after insert on dieta
for each row
begin
    insert into control_de_cambios_zoologico
    values (user(),"AGREGO DIETA",now());
end;
//
DELIMITER ;
insert into dieta values('D004','Dieta Acuatica','100g/dia','VET001');
```

El tercer trigger controla quien agrega un nuevo registro en la tabla dieta.

	usuario	accion	fecha
	root@localhost	AGREGO ANIMAL	2023-02-17 00:12:36
	root@localhost	ELIMINO ANIMAL	2023-02-17 00:16:17
	root@localhost	AGREGO DIETA	2023-02-17 00:22:45

```sql
#4. Trigger para cuando eliminen dieta
DELIMITER //
create trigger elimino_dieta after delete on dieta
for each row
begin
    insert into control_de_cambios_zoologico
    values (user(),"ELIMINO DIETA",now());
end;
//
DELIMITER ;
delete from dieta where id_dieta = 'D004';
```

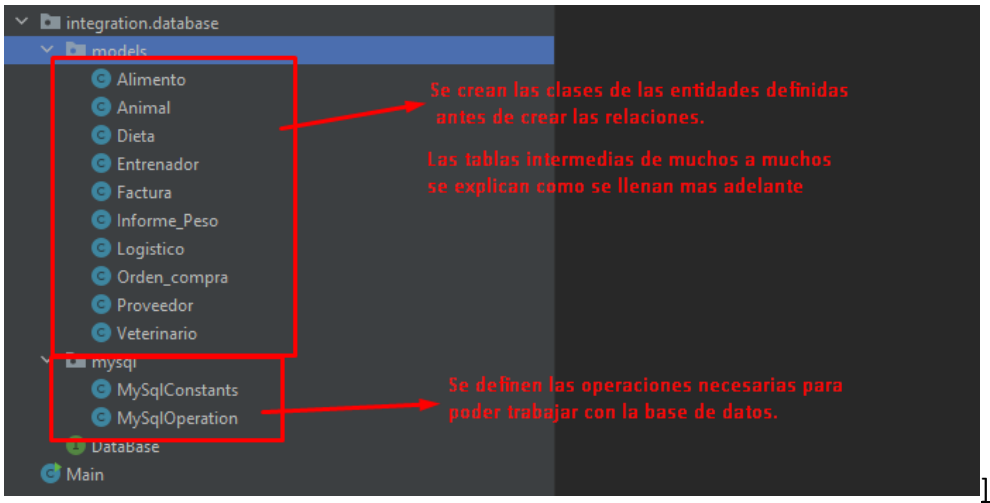
El cuarto trigger lleva un control de quien elimina un registro de la tabla dieta.

usuario	accion	fecha
root@localhost	AGREGO ANIMAL	2023-02-17 00:12:36
root@localhost	ELIMINO ANIMAL	2023-02-17 00:16:17
root@localhost	AGREGO DIETA	2023-02-17 00:22:45
root@localhost	ELIMINO DIETA	2023-02-17 00:24:08

***Poblado de la Base de Datos**

Para el poblado de la base de datos se creó la estructura explicada en la masterclass para que de esta manera todo funcionara adecamente. Posteriormente se modelaron las entidades de la base datos. En este caso en particular algunas tablas no se modelan ya que no tienen atributos nuevos , sino que son tablas intermedias que me relacionan los atributos de otras tablas por lo que posteriormente se explicara como se llenan estas tablas mediante los métodos en java.

Los modelos se pueden visualizar a continuación.



Se definen las variables para crear la conexión haciendo uso de una Instancia de MySqlOperation

```
private static final String SERVER="localhost";
1 usage
private static final String DATA_BASE_NAME="zoologico";
1 usage
private static final String USER="root";
1 usage
private static final String PASSWORD="admin1234";
```

Se definen los métodos de abrir y cerrar la conexión y se setean los atributos necesarios para la conexión asignándole los definidos anteriormente.

```
public static void openConnection(){
    mySqlOperation.setServer(SERVER);
    mySqlOperation.setDataBaseName(DATA_BASE_NAME);
    mySqlOperation.setUser(USER);
    mySqlOperation.setPassword(PASSWORD);
}

1 usage
public static void closeConnection(){
    mySqlOperation.close();

}
```

*Estructura básica de la clase para posteriormente poblar la base con registros.

```
2 usages
public class Animal {

    4 usages
    String id_animal;
    4 usages
    String nombre;
    4 usages
    String sexo_animal;
    4 usages
    String edad;
    4 usages
    String tipo_especie;
    4 usages
    String dieta_id;

    no usages
    public Animal(String id_animal, String nombre, String sexo_animal, String edad, String tipo_especie, String dieta_id) {
        this.id_animal = id_animal;
        this.nombre = nombre;
        this.sexo_animal = sexo_animal;
        this.edad = edad;
        this.tipo_especie = tipo_especie;
        this.dieta_id = dieta_id;
    }
```

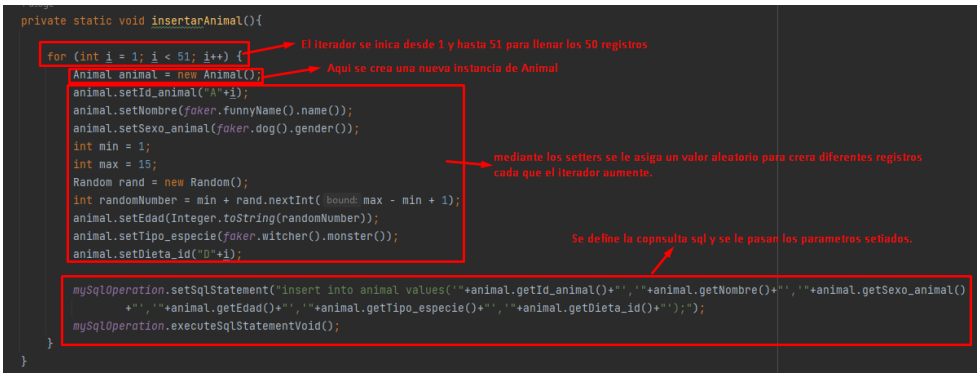
Se nombra la clase con el mismo nombre que tiene la tabla en la base de datos.

Las clases se crean de una forma basica y se definen los atributos que hacen referencia a las columnas que componen la tabla en la base de datos.

Todos se definen como String ya que estos atributos se pasaran a la query de esta manera.

Se define también el constructor con los atributos y también el constructor vacío para manipular los atributos mediante los setters y getters.

En la clase main se definen los métodos que agregaran datos a cada tabla




```
private static void insertarAnimal(){

    for (int i = 1; i < 51; i++) {
        Animal animal = new Animal();
        animal.setId_animal("A"+i);
        animal.setNombre(faker.funnyName().name());
        animal.setSexo_animal(faker.dog().gender());
        int min = 1;
        int max = 15;
        Random rand = new Random();
        int randomNumber = min + rand.nextInt( bound: max - min + 1);
        animal.setEdad(Integer.toString(randomNumber));
        animal.setTipo_especie(faker.witcher().monster());
        animal.setDieta_id("D"+i);

        mySqlOperation.setSqlStatement("insert into animal values('"+animal.getId_animal()+"','"+animal.getNombre()+"','"+animal.getSexo_animal()
                +"','"+animal.getEdad()+"','"+animal.getTipo_especie()+"','"+animal.getDieta_id()+"');");
        mySqlOperation.executeSqlStatementVoid();
    }
}
```

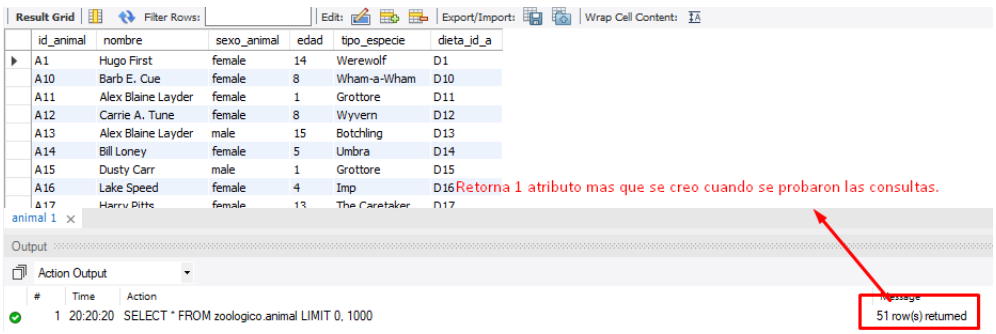
Los demás métodos se definen de la misma manera pero cambian los valores dependiendo los atributos que se definieron desde un principio para la base de datos. No veo necesario agregar pantallazos de todos los métodos definidos ya que todos siguen la misma lógica y se pueden visualizar en el repositorio que contiene todo el proyecto en la carpeta RetoZoo .

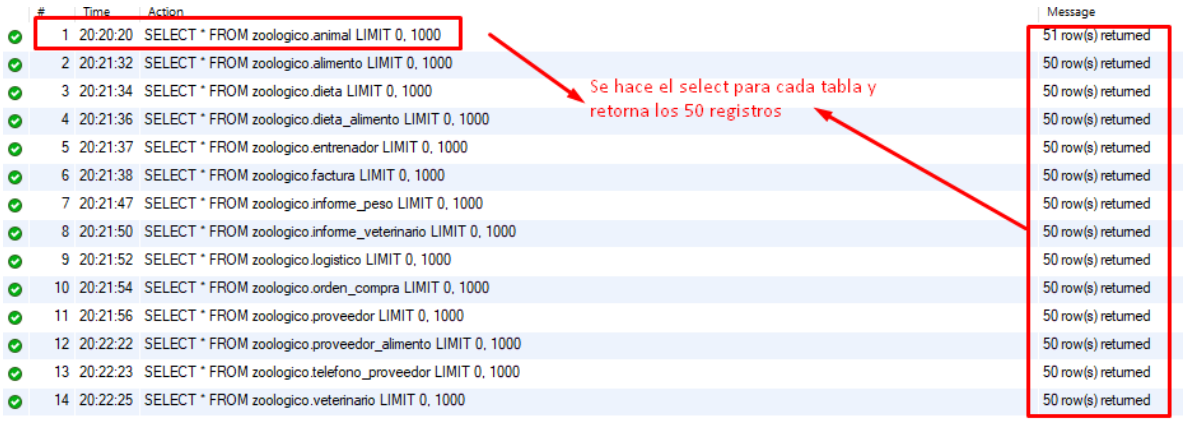
Para las tablas intermedias que no contienen atributos diferentes no vi la necesidad de crear nuevas clases por lo que se definen los métodos siguiendo la siguiente lógica.

```
private static void insertarProveedor_Telefono() {
    for (int i = 1; i < 51; i++) {
        String proveedor_id = "P"+faker.number().numberBetween(1, 51);
        String telefono = (faker.phoneNumber().cellPhone().toString());
        mySqlOperation.setSqlStatement("INSERT INTO telefono_proveedor VALUES ('" +proveedor_id+ "', '" +telefono+"');");
        mySqlOperation.executeSqlStatementVoid();
    }
}
```

Esta es la tabla del multivalor que es el teléfono del proveedor. Entonces con la lógica que genere los ids de los proveedores yo ya se que empiezan por P y van desde el 1 hasta el 50. Por eso defino la variable proveedor_id de tipo String que va a concatenar la “P” con un numero random generado por faker en este rango. Esto quiere decir que existen posibilidades de que para el mismo proveedor asigne distintos teléfonos.

Ahora se muestra evidencia de que se poblaron todas las tablas con 50 registros por tabla.





#	Time	Action	Message
1	20:20:20	SELECT * FROM zoologico.animal LIMIT 0, 1000	51 row(s) returned
2	20:21:32	SELECT * FROM zoologico.alimento LIMIT 0, 1000	50 row(s) returned
3	20:21:34	SELECT * FROM zoologico.dieta LIMIT 0, 1000	50 row(s) returned
4	20:21:36	SELECT * FROM zoologico.dieta_alimento LIMIT 0, 1000	50 row(s) returned
5	20:21:37	SELECT * FROM zoologico.entrenador LIMIT 0, 1000	50 row(s) returned
6	20:21:38	SELECT * FROM zoologico.factura LIMIT 0, 1000	50 row(s) returned
7	20:21:47	SELECT * FROM zoologico.informe_peso LIMIT 0, 1000	50 row(s) returned
8	20:21:50	SELECT * FROM zoologico.informe_veterinario LIMIT 0, 1000	50 row(s) returned
9	20:21:52	SELECT * FROM zoologico.logistico LIMIT 0, 1000	50 row(s) returned
10	20:21:54	SELECT * FROM zoologico.orden_compra LIMIT 0, 1000	50 row(s) returned
11	20:21:56	SELECT * FROM zoologico.proveedor LIMIT 0, 1000	50 row(s) returned
12	20:22:22	SELECT * FROM zoologico.proveedor_alimento LIMIT 0, 1000	50 row(s) returned
13	20:22:23	SELECT * FROM zoologico.telefono_proveedor LIMIT 0, 1000	50 row(s) returned
14	20:22:25	SELECT * FROM zoologico.veterinario LIMIT 0, 1000	50 row(s) returned

***Pregunta Final**

¿ Está conforme con el resultado obtenido según el contexto o cree que hubiera obtenido un mejor resultado con una base de datos no relacional?

Creo que el resultado obtenido para este contexto en especifico es muy bueno y, trabajándolo mediante una base de datos relacional pienso que se estructura de mejor manera la base de datos, además generar las relaciones entre tablas nos da un mejor entendimiento de como funciona el negocio incluso sin estar metidos de lleno en el.