

El parque zoo Santafé "parque de la conservación"

Ivan Dario Ruiz Bernal

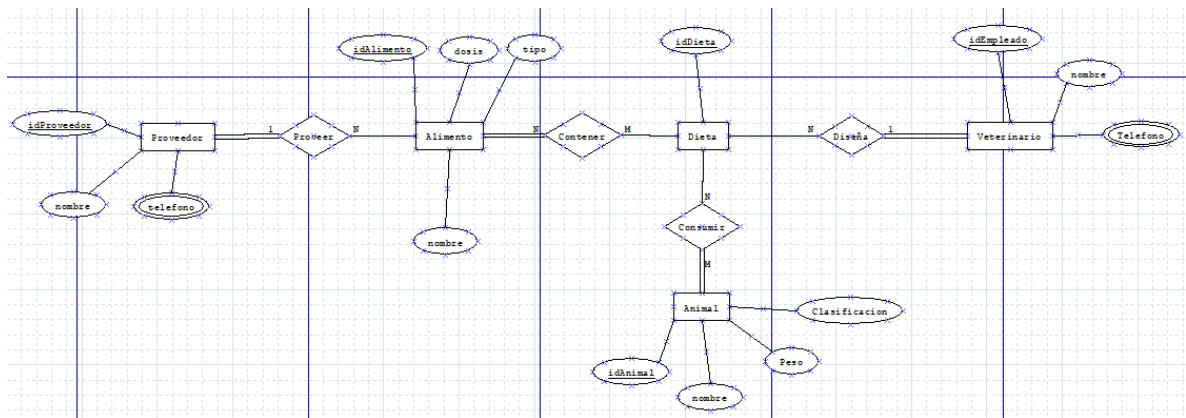
17/02/2023



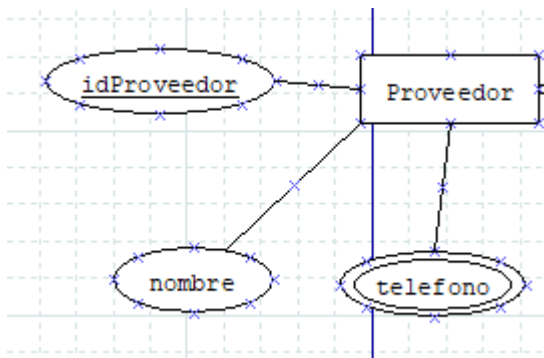
SOFKA U – QA

JUAN PINEDA

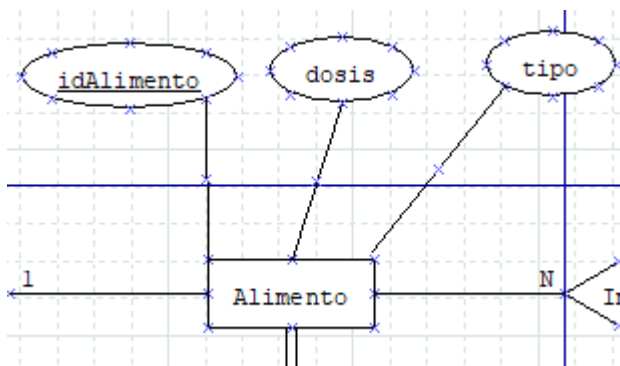
Modelo E-R



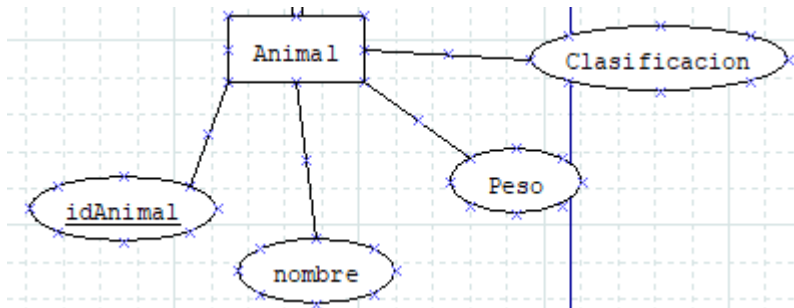
Entidades:



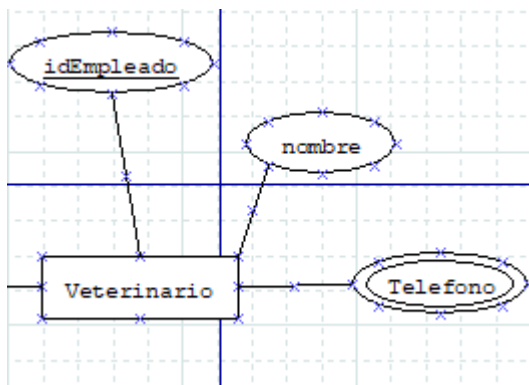
Proveedor, nos interesa guardar su información por si vemos que algún alimento llega a tener problemas podemos hacer un seguimiento preciso de dónde proviene.



Alimento, nos indica la dosis, el tipo y nombre de alimento, que nos sirve para saber los animales que comen, de que proveedor vienen y que veterinario los ha implementado en la dieta de los animales

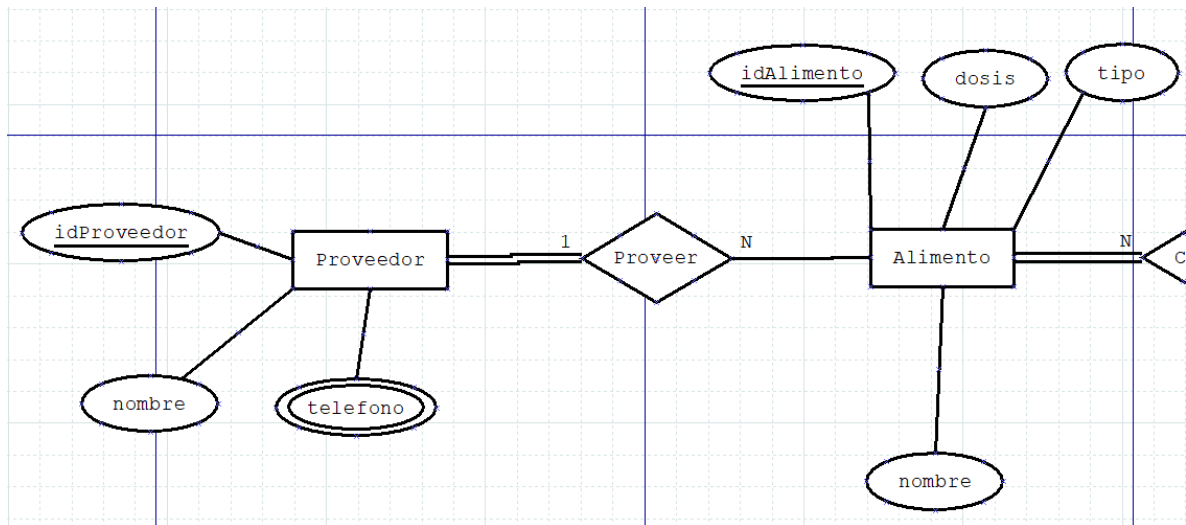


Animal, nos indica el nombre, peso y clasificación de cada animal.



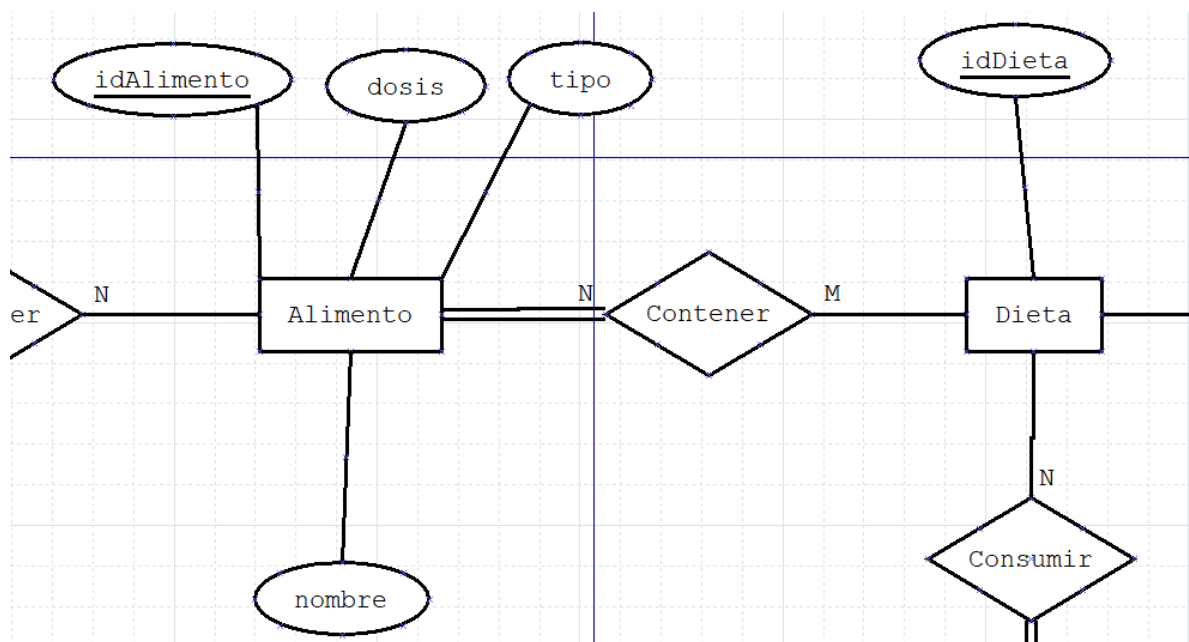
Veterinario, nos indica el nombre y los teléfonos del veterinario, además nos permite saber las dietas que ha diseñado el veterinario.

Relaciones:



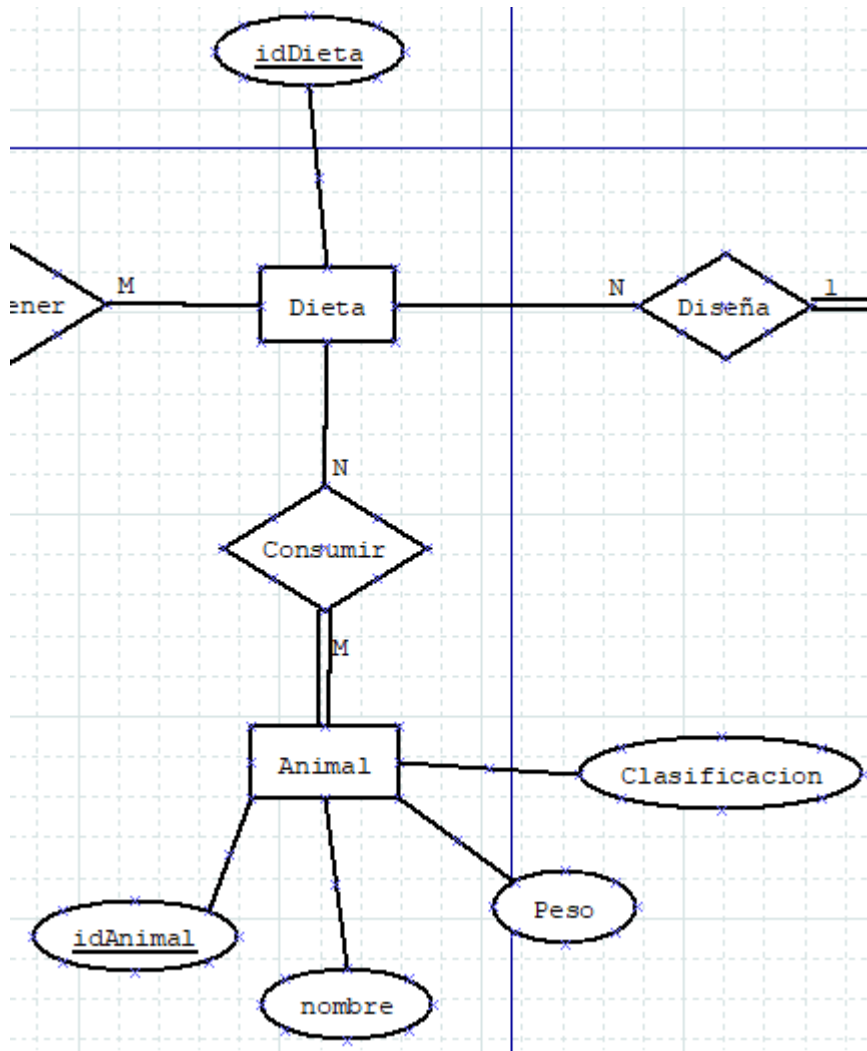
Primero haremos la lectura de la relación, un proveedor puede proveer uno o más alimentos, por otro lado, un alimento es proveído por un proveedor.

El alimento necesita la existencia de al menos un proveedor para existir por lo que usamos la participación total en la entidad que necesita par su existencia en la relación.



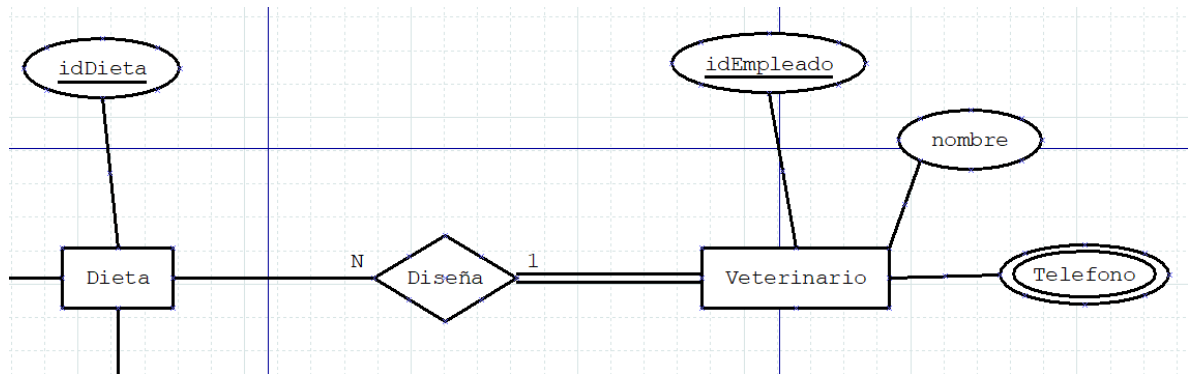
La lectura de la relación es un alimento puede estar contenido en una o más dietas, mientras una dieta puede contener uno o más alimentos.

Vemos que la dieta necesita la existencia de al menos un alimento por lo que ponemos una participación total desde la entidad necesaria hacia la relación.



Una dieta puede ser consumida por uno o más animales, además un animal consume una o más dietas

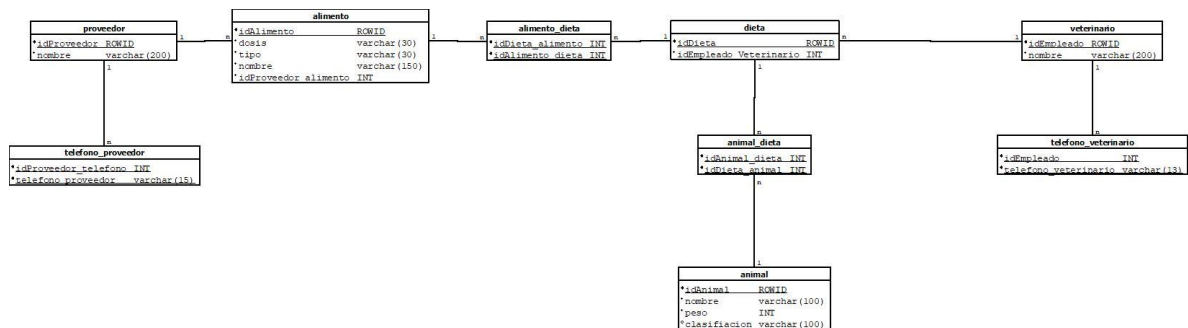
Una dieta necesita la existencia de un animal para poder existir por lo que ponemos la participación de animal a consumir



Una dieta es diseñada por un veterinario, por otro lado, un veterinario diseña una o más dietas

Una dieta necesita la existencia de un veterinario que la diseñe para existir por lo que usamos la participación total en su entidad necesaria.

Modelo Relacional



Hacemos las transformaciones necesarias del modelo Entidad – Relación al modelo relacional.

Usamos el tipo de dato ROWID que nos proporciona un número de id único para cada entidad así le encargamos el identificador único a la base de datos y aseguramos la integridad de cada registro.

Lo primero que vemos es la adición de nuevas tablas, hablaremos primero de las tablas que aparecieron gracias a las relaciones.

Animal_dieta:

animal_dieta	
*idAnimal_dieta	INT
*idDieta_animal	INT

Esta tabla sale de la relación de muchos a muchos entre animal y dieta, teniendo la concatenación de las llaves primarias de las tablas mencionadas anteriormente además de ser foráneas a su vez.

Animal_dieta:

alimento_dieta	
*idDieta_alimento	INT
*idAlimento_dieta	INT

Esta tabla sale de la relación (N:M) entre alimento y dieta, teniendo como llaves principales foráneas la concatenación de las llaves principales de alimento y dieta.

Luego de tener las relaciones (N:M) transformadas a las tablas, vemos que las relaciones de uno a muchos nos generan llaves foráneas:

dieta	
*idDieta	ROWID
idEmpleado Veterinario	INT

Vemos que tiene el id del veterinario para poder relacionarlo con las distintas dietas que pueda diseñar

alimento	
* <u>idAlimento</u>	BOWID
dosis	varchar(30)
tipo	varchar(30)
nombre	varchar(150)
idProveedor alimento	INT

Vemos que tiene el id del proveedor para poder tener una trazabilidad de la empresa que está proporcionando el alimento.

Por el momento no hemos iniciado la normalización, vemos la primera forma y vemos que cumple, ya que, tiene una clave definida y no tiene atributos multivalor ni repetidos, siendo así atributos atómicos, los atributos que veíamos multivalor se transformaron en tablas con la concatenación de su propia llave principal y la de la entidad que las contenía:

telefono_veterinario	
* <u>idEmpleado</u>	INT
telefono_veterinario	varchar(13)

telefono_proveedor	
* <u>idProveedor_telefono</u>	INT
telefono_proveedor	varchar(15)

Cumple la segunda forma normal ya que, tiene una clave única ya que las columnas dependen de la llave primaria de la tabla y es única.

Cumple la tercera forma normal debido a que, cada atributo que no está incluido en la clave primaria no depende transitivamente de la clave primaria.

Explicación de la estructura seleccionada

Elegí estas entidades y su manera de relacionarse ya que me interesa registrar el consumo de los alimentos por parte de los animales. Sabemos que existen personas que hacen el control del peso de los animales, por lo que los animales siempre tienen un peso, esto puede o no puede relacionarse con el alimento, por lo tanto, esta base de datos pretende ayudar a tener el control de esto.

De este modo vemos a esa persona como un usuario que modifica el peso actual del animal en la base de datos, al momento de hacer esto se disparará el gatillo que guarda el peso anterior y la dieta que se le proporcionaba en un historial de pesos, así podemos tener la trazabilidad de los animales por dieta, peso, y especie. Si en dado momento el veterinario cambia la dieta también se guardará lo anterior y lo nuevo para poder tener una trazabilidad de cada especie.

Esto nos permite poder hacer un manejo de datos y un análisis exhaustivo de la mano del veterinario para ver la relación entre la comida y los pesos de los animales, por otro lado, si llega a haber una complicación con los animales y se presume que el alimento tuvo que ver con la enfermedad, se puede hacer una trazabilidad de donde viene el alimento, lo que se le daba, su dieta, su proveedor y su dosis. Esto nos permite tener un control total de la cadena de alimentación hacia los animales

No incluimos las facturas, ni las ordenes hacia el proveedor, debido a que para la funcionalidad que se le pretende dar a la base de datos no hace mucha relevancia. La factura de los alimentos proveídos no me da mucha información con respecto a que le sucede a un animal con respecto a lo que come, esta información nos serviría más si pretendiera crear una persistencia del proceso total de la gestión de los alimentos.

Como esta base se enfoca hacia el manejo del veterinario y sus tomas de decisiones brindamos toda la información de interés, evitando crear la persistencia de archivos que pueden ocupar espacio y no ser usados por el empleado, mientras se puede crear una base de datos de pagos que guarde las nóminas y las facturas, ya que estas se pagan a final de mes, por lo que tiene más sentido tener esta información en una base de datos de pagos que se consulte con esta regularidad.

Creación Base de datos en SQL

```
1 • ○ create table veterinario(  
2     idEmpleado int auto_increment primary key,  
3     nombre varchar(60)  
4 );  
5  
6 • ○ create table telefono_veterinario(  
7     idEmpleado_telefono int,  
8     telefono_veterinario varchar(15),  
9     primary key (idEmpleado_telefono,telefono_veterinario),  
10    foreign key(idEmpleado_telefono) references veterinario(idEmpleado)  
11 );
```

Iniciamos las sentencias con las palabras reservadas “create table” las cuales nos permiten crear tablas con el nombre que indiquemos después de esta sentencia y las columnas que le pasemos dentro de los paréntesis “()”; vemos que en las columnas hay una estructura diferente, primero se indica el nombre que tendrá la columna, luego el tipo de dato que usará, en el primer caso hacemos que la base de datos asigne los valores del id por lo que ponemos la palabra reservada “auto_increment” que hace que se cree un valor único para cada registro, y finalmente le decimos que es la llave primaria. Con el nombre solo debemos especificar su tipo.

En la segunda tabla indicamos las dos columnas solo con su tipo de dato, luego hacemos la concatenación de las columnas anteriores con la palabra reservada “primary key” y luego le indicamos que columna será una llave foránea, por lo que va a tomar datos de otra tabla,

esto lo hacemos con la palabra reservada “foreign key(nombre de la tabla)”, luego le indicamos de donde se va a tomar “references” seguido del nombre de la tabla y entre paréntesis el nombre de la columna de la tabla, quedando nombre_de_la_tabla(nombreColumna).

```
1 • ○ create table dieta(  
2     idDieta int auto_increment primary key,  
3     idEmpleado_veterinario int,  
4     foreign key(idEmpleado_veterinario) references veterinario(idEmpleado)  
5 );  
6  
7 • ○ create table animal(  
8     idAnimal int auto_increment primary key,  
9     nombre varchar(60),  
10    peso varchar(4),  
11    clasificacion varchar(50)  
12 );  
13  
14 • ○ create table animal_dieta(  
15     idAnimal_dieta int,  
16     idDieta_animal int,  
17     primary key(idAnimal_dieta,idDieta_animal),  
18     foreign key(idAnimal_dieta) references animal(idAnimal),  
19     foreign key(idDieta_animal) references dieta(idDieta)  
20 );
```

```

1 • ⊖ create table proveedor(
2     idProveedor int auto_increment primary key,
3     nombre varchar(60)
4 );
5
6 • ⊖ create table telefono_proveedor(
7     idProveedor_telefono int,
8     telefono_proveedor varchar(15),
9     primary key(idProveedor_telefono,telefono_proveedor),
10    foreign key(idProveedor_telefono) references proveedor(idProveedor)
11 );
12
13 • ⊖ create table alimento(
14     idAlimento int auto_increment primary key,
15     dosis varchar(30),
16     tipo varchar(30),
17     nombre varchar(60),
18     idProveedor_alimento int,
19     foreign key(idProveedor_alimento) references proveedor(idProveedor)
20 );

```

```

1 • ⊖ create table alimento_dieta(
2     idAlimento_dieta int,
3     idDieta_alimento int,
4     primary key(idAlimento_dieta,idDieta_alimento),
5     foreign key(idAlimento_dieta) references alimento(idAlimento),
6     foreign key(idDieta_alimento) references dieta(idDieta)
7 );

```

Seguimos los mismos principios explicados con la primera tabla.

Creación de triggers y las tablas que guardan los cambios

```

create table control_cambio_peso(
    idCambio int auto_increment primary key,
    usuario varchar(60),
    accion varchar(17),
    fecha date,
    idAnimal int,
    nombreAnimal varchar(60),
    clasificaion varchar(50),
    peso int
);

```

Creamos la tabla que guardará los cambios en el peso del animal

```

create table control_cambio_dieta(
    idCambio int auto_increment primary key,
    usuario varchar(60),
    accion varchar(18),
    fecha date,
    idDieta int
);

```

Creamos la tabla que guardará los cambios en la dieta del animal

```

DELIMITER //
create trigger cambiar_peso after update on animal
for each row
begin
    insert into control_cambio_peso(usuario,accion,fecha,idAnimal,nombre,clasificacion,peso)
    values (user(),"cambio en el peso",now(),old.idAnimal,old.nombre,old.clasificacion,old.peso);
end; //
DELIMITER ;

```

Declaramos el trigger en que tabla estará atento

Hacemos la inserción de los cambios realizados en animal

Creamos el primer trigger para guardar las actualizaciones en el peso de los animales

```

delimiter //
create trigger nuevo_animal after insert on animal
for each row
begin
    insert into control_cambio_peso(usuario,accion,fecha,idAnimal,nombre,clasificacion,peso)
    values(user(),"insercion",now(),new.idAnimal,new.nombre,new.clasificacion,new.peso);
end; //
delimiter ;

```

Declaramos que se ejecute cuando añadimos un animal

Hacemos que se inserte por primera vez el animal para llevar un control desde el principio

Creamos el segundo trigger para guardar los nuevos animales

```

DELIMITER //
create trigger cambiar_dieta after update on alimento_dieta
for each row
begin
    insert into control_cambio_dieta(usuario,accion,fecha,idDieta)
    values (user(),"cambio en la dieta",now(),old.idDieta_alimento);
end; //
DELIMITER ;

```

Se declara que guarde las actualizaciones de la tabla alimento_dieta

Se añade el cambio para tener un control de las dietas a lo largo del tiempo

Creamos el tercer trigger para guardar las actualizaciones en la tabla alimento_dieta

```

delimiter //
create trigger nueva_dieta after insert on alimento_dieta
for each row
begin
    insert into control_cambio_dieta(usuario,accion,fecha,idDieta)
    values(user(),"insercion",now(),new.idDieta_alimento);
end; //
delimiter ;

```

Se crea el trigger para que observe cuando se inserta una nueva dieta y se dispare

Al dispararse crea el primer registro de esta dieta en la tabla de control así podemos tener un control de la dieta desde su creación

Creamos el cuarto trigger para guardar las dietas en el control desde que se crean

Creación de las consultas

1. Consultar todos los alimentos:

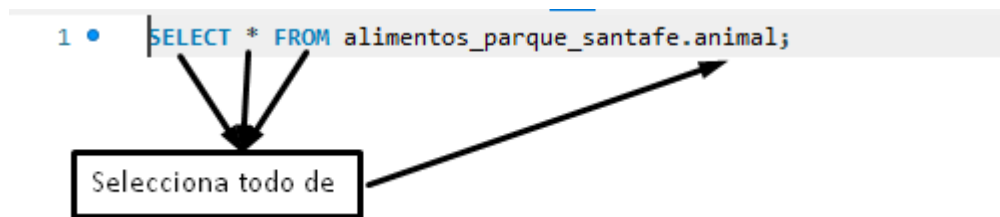
```

1 • SELECT * FROM alimentos_parque_santafe.alimento;

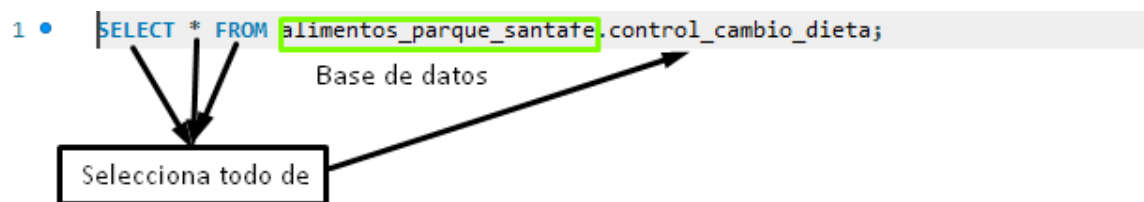
```

Selecciona todo de

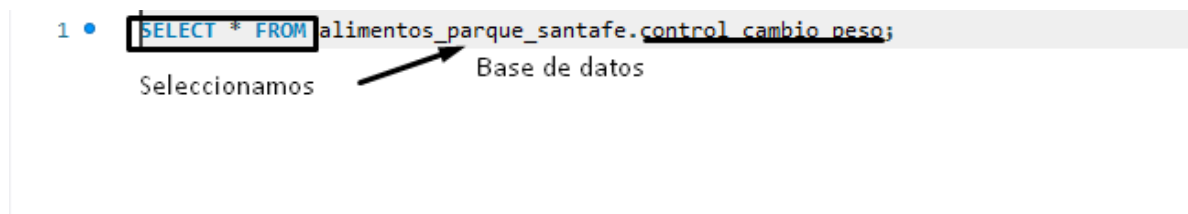
2. Consultar todos los animales:



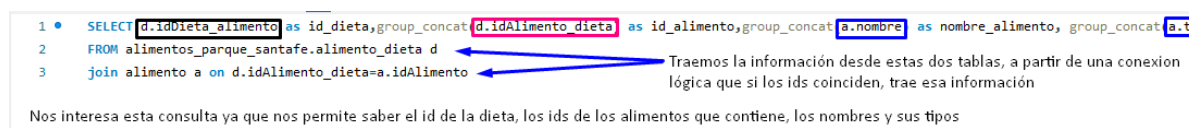
3. Consultar todos los cambios e inserciones a una o más dietas:



4. Consultar todos los cambios e inserciones a uno o más pesos de los animales:



5. Consultar la dieta de los animales:



6. Consultar proveedores:

```

1 • SELECT p.idProveedor,p.nombre as nombre_proveedor,group_concat(a.idAlimento) as id_alimento,group_concat(a.nombre) as
2 FROM alimentos_parque_santafe.proveedor p
3 join alimento a on p.idProveedor=a.idProveedor alimento
4 group by p.idProveedor;

```

Tablas que brindan
la información

Se agrupará esta información con respecto a el id del proveedor

Agrupación para un mejor análisis de la
información

7. Consultar teléfonos de un proveedor:

```

1 • select p.idProveedor,p.nombre,group_concat(tp.telefono_proveedor) as telefonos_proveedor
2 from proveedor p
3 join telefono_proveedor tp on p.idProveedor=tp.idProveedor_telefono
4 group by p.idProveedor;

```

Podemos consultar todos los números de telefono de un proveedor por medio de un registro

8. Consultar Veterinario:

```

1 • select v.idEmpleado, v.nombre,group_concat(ad.idDieta_alimento) as dietas_diseñadas,group_concat(a.nombre) as alimento
2 from veterinario v
3 join dieta d on d.idEmpleado_veterinario=v.idEmpleado
4 join alimento_dieta ad on ad.idDieta_alimento=d.idDieta
5 join alimento a on a.idAlimento=ad.idAlimento_dieta
6 group by v.idEmpleado;

```

Con esta consulta conectamos varias tablas para poder traer información del veterinario, trayendo las dietas diseñadas,
los alimentos que usa

9. Consultar los teléfonos de los veterinarios:

```
1 • select v.idEmpleado as id_veterinario, group_concat(tv.telefono_veterinario) as telefonos_veterinario
2 from veterinario v
3 join telefono_veterinario tv on v.idEmpleado=tv.idEmpleado_telefono
4 group by v.idEmpleado
```

10. Consultar todos los cambios realizados por un usuario:

```
1 • select cd.usuario as usuarios, group_concat(cd.accion, cp.accion) as acciones, group_concat(cd.fecha, cp.fecha) as fechas_de_c
2 from control_cambio_dieta cd
3 join control_cambio_peso cp on cd.usuario=cp.usuario
4 group by cd.usuario
```

Esta consulta nos permite saber todos los cambios realizados por un usuario con fechas y acciones

Creación de las vistas

1.

```

1 • create view animales as
2 select a.idAnimal, a.nombre, a.peso, group_concat(cp.pesoAnimal) as historico_pesos,
3 ad.idDieta_animal, al.tipo, al.dosis, group_concat(al.idProveedor_alimento) as id_proveedor
4 ,group_concat(p.nombre) as nombre_proveedor, group_concat(pv.telefono_proveedor)
5 as telefonos_proveedor
6 from animal a
7 join control_cambio_peso cp on cp.idAnimal_cambio=a.idAnimal
8 join animal_dieta ad on a.idAnimal=ad.idAnimal_dieta
9 join alimento_dieta ald on ald.idDieta_alimento=ad.idDieta_animal
10 join alimento al on al.idAlimento=ald.idAlimento_dieta
11 join proveedor p on p.idProveedor=al.idProveedor_alimento
12 join telefono_proveedor pv on pv.idProveedor_telefono=p.idProveedor
13 group by a.idAnimal, ad.idDieta_animal, al.tipo, al.dosis;

```

hacemos uso de la conexión entre alimento y dieta

Tomamos los telefonos del proveedor

tomamos información de la tabla animal

tomamos información de los cambios en el peso del animal

tomamos información de la dieta del animal

tomamos la información de los alimentos de las dietas

tomamos la información del proveedor

2.

```

1 • create view dietas as
2 SELECT d.idDieta_alimento as id_dieta, group_concat(d.idAlimento_dieta) as id_alimento,
3 group_concat(a.nombre) as nombre_alimento, group_concat(a.tipo) as tipo_alimento
4 FROM alimentos_parque_santafe.alimento_dieta d
5 join alimento a on d.idAlimento_dieta=a.idAlimento
6 group by d.idDieta_alimento;

```

Tomamos la información de las dietas, sus alimentos, sus dosis y sus tipos

3.

```

1 • create view veterinarios as
2 select v.idEmpleado, v.nombre, group_concat(tv.telefono_veterinario) as telefonos_veterinario
3 from dieta d
4 join veterinario v on d.idEmpleado_veterinario=v.idEmpleado
5 join telefono_veterinario tv on tv.idEmpleado_telefono=v.idEmpleado
6 group by v.idEmpleado;

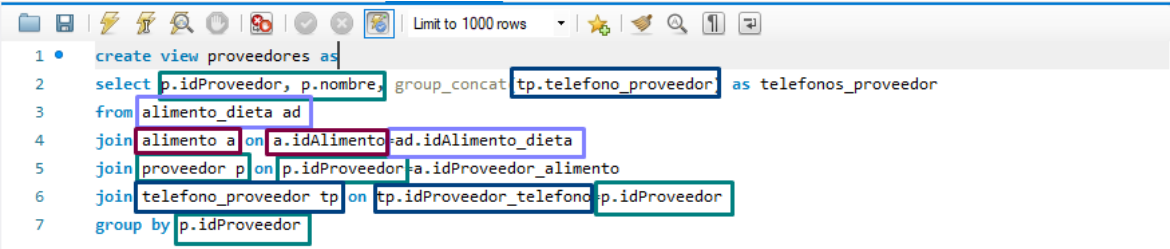
```

Nos restringe que solo se muestren veterinarios que hayan diseñado una dieta

Nos trae la información del veterinario

Nos trae la información de contacto de los veterinarios

4.



```
1 • create view proveedores as
2 select p.idProveedor, p.nombre, group_concat(tp.telefono_proveedor) as telefonos_proveedor
3 from alimento_dieta ad
4 join alimento a on a.idAlimento=ad.idAlimento_dieta
5 join proveedor p on p.idProveedor=a.idProveedor_alimento
6 join telefono_proveedor tp on tp.idProveedor_telefono=p.idProveedor
7 group by p.idProveedor
```

Se trae la información de los proveedores pero solo de los que al menos uno de sus alimentos se estén usando en una dieta

Procedimientos

1.



```
1 delimiter //
2 • create procedure insertar_animal
3 in nombre varchar(60),
4 in peso varchar(4),
5 in clasificacion varchar(50)
6 )
7 begin
8 insert into animal(nombre,peso,clasificacion)
9 values(nombre,peso,clasificacion);
10 end; //
11 delimiter ;
```

2.

```

1 delimiter //
2 • create procedure insertar_veterinario
3   in nombre varchar(60)
4 )
5 begin
6   insert into veterinario nombre
7   values nombre;
8 end; //
9 delimiter ;

```

3.

```

1 delimiter //
2 • create procedure insertar_proveedor
3   in nombre varchar(60)
4 )
5 begin
6   insert into proveedor nombre
7   values nombre;
8 end; //
9 delimiter ;

```

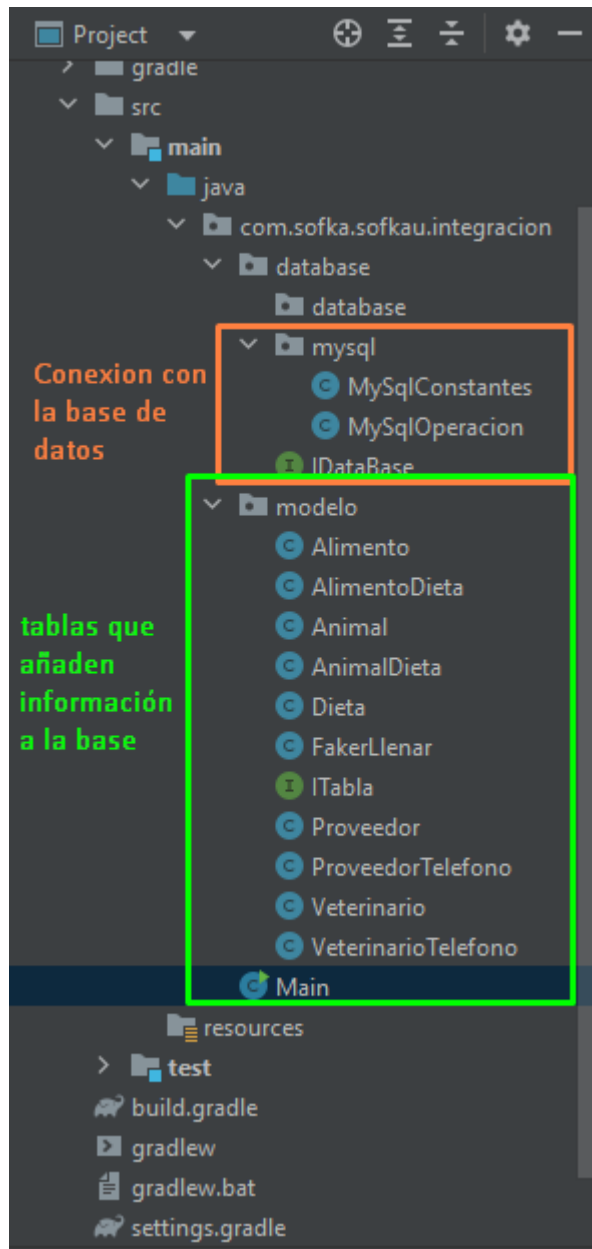
4.

```

1 delimiter //
2 • create procedure insertar_alimento
3   in dosis varchar(30),
4   in tipo varchar(30),
5   in nombre varchar(60),
6   in idProveedor_alimento int
7 )
8 begin
9   insert into alimento dosis tipo nombre idProveedor_alimento
10  values dosis tipo nombre idProveedor_alimento;
11 end; //
12 delimiter ;

```

Código en Java



```

1 usage
private static final String SERVIDOR="localhost";
3 usages
private static final String NOMBRE_BASE_DATOS="alimentos_parque_santafe";
1 usage
private static final String USUARIO="root";
1 usage
private static final String CONTRASENA="IVAN.dario00";
1 usage
private static final String INSERTAR_ANIMAL=("insert into %s.%s(%s) values (%s);");
no usages
private static final String SELECT_ALL_FROM_ANIMALES=String.format("select * from %s.animal",NOMBRE_BASE_DATOS);
7 usages
private static final MySQLOperacion mySqlOperacion= new MySQLOperacion();

```

Conexion con la base de datos (blue text) points to the database connection constants.

Instrucciones a la base de datos (pink text) points to the database operation instructions.

```

1 usage  Ivan
private static void insertIntoAnimal(String tabla,String atributos,String valores) {
    mySqlOperacion.setInstruccionSql(String.format(INSERTAR_ANIMAL,NOMBRE_BASE_DATOS,
        tabla,
        atributos,
        valores));
    mySqlOperacion.ejecutarInstruccionSqlVoid();
}

1 usage  Ivan
public static void abrirConexion() {
    mySqlOperacion.setServidor(SERVIDOR);
    mySqlOperacion.setNombreBaseDatos(NOMBRE_BASE_DATOS);
    mySqlOperacion.setUsuario(USUARIO);
    mySqlOperacion.setContraseña(CONTRASENA);
}

1 usage  Ivan
public static void closeConexion() { mySqlOperacion.cerrar(); }

```

Nos permite insertar animales en la base de datos

Nos permite usar un puente entre la aplicación y la base de datos

Nos permite cerrar la conexión para evitar errores y daños en la base de datos, cuidando así la integridad de los mismos

```

ITabla tabla=new Veterinario();
int contador=0;
for(int i=0;i<51;i++){
    abrirConexion();
    insertIntoAnimal(tabla.getTabla(),tabla.getAtributos(),tabla.getValores());
    closeConexion();
    if(tabla instanceof AlimentoDieta&&i==49){
        break;
    }
    if(i==50){
        i=0;
        contador++;
    }
    switch (contador){
        case 1: tabla=new VeterinarioTelefono();
            break;
        case 2: tabla=new Dieta();
            break;
        case 3: tabla=new Animal();
            break;
        case 4: tabla=new AnimalDieta();
            break;
        case 5: tabla=new Proveedor();
            break;
        case 6: tabla=new ProveedorTelefono();
            break;
        case 7: tabla=new Alimento();
            break;
        case 8: tabla=new AlimentoDieta();
            break;
    }
}

```

Interface que nos permite usar diferentes estrategias para ingresar datos a la base siguiendo el patrón de desarrollo Strategy

acción que abre la conexión con la base de datos, inserta los datos y luego cierra la conexión para evitar daños en la integridad de los datos

Condición que nos permite terminar las iteraciones

Condición que nos permite reiniciar la variable de iteración al llenar las tablas con los registros deseados además de ayudarnos en la elección de la estrategia de llenado a seguir

Evidencia de que se llenaron las tablas

alimento

1 • `SELECT * FROM alimentos_parque_santafe.alimento`

Limit to 1000 rows

Result Grid

	idAlimento	dosis	tipo	nombre	idProveedor_alimento
1	359	gr	seca	comida	12
2	38	gr	mixta	comida	39
3	458	gr	mixta	comida	11
4	237	gr	mixta	comida	24
5	132	gr	seca	comida	5
6	244	gr	mixta	comida	34
7	28	gr	seca	comida	14
8	285	gr	mixta	comida	14
9	484	gr	seca	comida	12

alimento 1 x

Output

Action Output

#	Time	Action
1	18:59:44	<code>SELECT * FROM alimentos_parque_santafe.alimento LIMIT 0, 1000</code>

Message

50 row(s) returned

alimento_dieta

1 • `SELECT * FROM alimentos_parque_santafe.alimento_dieta`

Limit to 1000 rows

Result Grid

	idAlimento_dieta	idDieta_alimento
14	1	
13	3	
21	3	
2	4	
26	4	
47	9	
26	10	
20	12	
28	12	

alimento_dieta 3 x

Output

Action Output

#	Time	Action
1	19:04:07	<code>SELECT * FROM alimentos_parque_santafe.alimento_dieta LIMIT 0, 1000</code>

Message

50 row(s) returned

animal

1 • `SELECT * FROM alimentos_parque_santafe animal`

Result Grid

	idAnimal	nombre	peso	clasificacion
1	1	donkey	142	reptiles
2	2	grasshopper	920	anfibios
3	3	cod	594	anfibios
4	4	crab	427	aves
5	5	tiger	742	peces
6	6	pug	826	peces
7	7	termite	534	reptiles
8	8	lizard	341	reptiles
9	9	swan	309	aves

animal 1

Output

Action Output

#	Time	Action	Message
1	19:06:54	<code>SELECT * FROM alimentos_parque_santafe animal LIMIT 0, 1000</code>	50 row(s) returned

animal_dieta

1 • `SELECT * FROM alimentos_parque_santafe animal_dieta`

Result Grid

	idAnimal_dieta	idDieta_animal
2	2	1
1	1	2
41	41	2
22	22	3
20	20	4
3	3	5
19	19	7
25	25	8
37	37	8

animal_dieta 1

Output

Action Output

#	Time	Action	Message
1	19:08:51	<code>SELECT * FROM alimentos_parque_santafe animal_dieta LIMIT 0, 1000</code>	50 row(s) returned

control_cambio_dieta

1 • `SELECT * FROM alimentos_parque_santafe.control_cambio_dieta`

Result Grid

	idCambio	usuario	accion	fecha	idDieta
1	1	root@localhost	insercion	2023-02-18	36
2	2	root@localhost	insercion	2023-02-18	9
3	3	root@localhost	insercion	2023-02-18	13
4	4	root@localhost	insercion	2023-02-18	26
5	5	root@localhost	insercion	2023-02-18	27
6	6	root@localhost	insercion	2023-02-18	15
7	7	root@localhost	insercion	2023-02-18	36
8	8	root@localhost	insercion	2023-02-18	3
9	9	root@localhost	insercion	2023-02-18	4

Output

Action Output

#	Time	Action	Message
1	19:10:40	<code>SELECT * FROM alimentos_parque_santafe.control_cambio_dieta LIMIT 0, 1000</code>	50 row(s) returned

control_cambio_peso

1 • `SELECT * FROM alimentos_parque_santafe.control_cambio_peso;`

Result Grid

	idCambio	usuario	accion	fecha	idAnimal_cambio	nombreAnimal	clasificacionAnimal	pesoAnimal
1	1	root@localhost	insercion	2023-02-18	1	donkey	reptÃ-les	142
2	2	root@localhost	insercion	2023-02-18	2	grasshopper	anfibios	920
3	3	root@localhost	insercion	2023-02-18	3	cod	anfibios	594
4	4	root@localhost	insercion	2023-02-18	4	crab	aves	427
5	5	root@localhost	insercion	2023-02-18	5	tiger	peces	742
6	6	root@localhost	insercion	2023-02-18	6	pug	peces	826
7	7	root@localhost	insercion	2023-02-18	7	termite	reptÃ-les	534
8	8	root@localhost	insercion	2023-02-18	8	lizard	reptÃ-les	341
9	9	root@localhost	insercion	2023-02-18	9	swan	aves	309

Output

Action Output

#	Time	Action	Message
1	19:15:13	<code>SELECT * FROM alimentos_parque_santafe.control_cambio_peso LIMIT 0, 1000</code>	50 row(s) returned

dieta

Limit to 1000 rows

1 • `SELECT * FROM alimentos_parque_santafe.dieta;`

Result Grid

idDieta	idEmpleado_veterinario
34	2
31	4
46	6
28	8
1	10
43	10
9	11
26	12
48	17

dieta 1 x

Output

Action Output

#	Time	Action	Message
1	19:19:38	<code>SELECT * FROM alimentos_parque_santafe.dieta LIMIT 0, 1000</code>	50 row(s) returned

proveedor

Limit to 1000 rows

1 • `SELECT * FROM alimentos_parque_santafe.proveedor;`

Result Grid

idProveedor	nombre
1	Corinna Schimmel
2	Dr. Tinisha Cremin
3	Wade Okuneva
4	Mrs. Hanna Feest
5	Donald Keeling
6	Raymond Kirin
7	Lavonne Borer Sr.
8	Glen Ryan
9	Miss Deanna Halvorsen

proveedor 2 x

Output

Action Output

#	Time	Action	Message
1	19:22:23	<code>SELECT * FROM alimentos_parque_santafe.proveedor LIMIT 0, 1000</code>	50 row(s) returned

telefono_proveedor

1 • SELECT * FROM alimentos_parque_santafe.telefono_proveedor;

Result Grid

	idProveedor_telefono	telefono_proveedor
1	651.156.5741	
2	(657) 221-6859	
2	1-449-763-0723	
2	150.641.7884	
2	798.652.7845	
4	1-211-572-4997	
5	466-569-4019	
6	1-336-678-2031	
6	729-136-5160	

telefono_proveedor 2 x

Output

Action Output

#	Time	Action	Message
1	19:26:51	SELECT * FROM alimentos_parque_santafe.telefono_proveedor LIMIT 0, 1000	50 row(s) returned

telefono_veterinario

1 • SELECT * FROM alimentos_parque_santafe.telefono_veterinario

Result Grid

	idEmpleado_telefono	telefono_veterinario
1	1-163-572-1785	
1	638.032.0591	
7	(058) 821-4621	
7	305.709.8897	
9	011.082.2821	
9	1-342-394-5385	
9	1-813-479-7003	
12	(007) 930-4961	
12	078-751-4911	

telefono_veterinario 1 x

Output

Action Output

#	Time	Action	Message
1	19:28:14	SELECT * FROM alimentos_parque_santafe.telefono_veterinario LIMIT 0, 1000	50 row(s) returned

The screenshot shows a database query interface. At the top, a query editor displays the SQL statement: `SELECT * FROM alimentos_parque_santafe.veterinario`. Below the query editor, a 'Result Grid' shows the results of the query. The grid has two columns: 'idEmpleado' and 'nombre'. The results are as follows:

idEmpleado	nombre
1	Merle Kerluke IV
2	Stan Stracke
3	Barrie Robel
4	Dr. Paz Larson
5	Tad Anderson
6	Hermelinda Walsh
7	Jarrett Donnelly
8	Lavern Carter
9	Moises Luettnen

At the bottom of the interface, an 'Action Output' section shows the execution details: `SELECT * FROM alimentos_parque_santafe.veterinario LIMIT 0, 1000`. A message box indicates '51 row(s) returned'.

Respuesta a la pregunta

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

Sabemos que la diferencia entre las bases relacionales y no relacionales es su estructura, las bases relacionales siguen la estructura de tablas, llaves foráneas, llaves primarias, etc. Por otro lado, las no relacionales no tienen una estructura definida, por lo que nos permite escalar datos semi estructurados o directamente no estructurados a gran escala.

Por lo que en el contexto del proyecto no tiene sentido manejar algo no estructurado ya que necesitamos la relación entre los datos para poder tener un control de lo que los animales usan como alimento.