

Transporte de Animales Domésticos: Animal Trip

Presentado por:

Andrés Felipe Sánchez Arias

Corporación Universitaria Minuto de Dios
Ingeniería de Sistemas

Contenido

Introducción	3
Objetivos.....	4
Descripción del problema	5
Creación de base de datos	7
Stored Procedures.....	12
Triggers	13
Usuarios y logins	17
Diseño y desarrollo de la aplicación.....	24

Actualmente, las mascotas hacen parte importante en la vida de numerosas personas. Por lo que, su cuidado, comodidad y seguridad son factores que influyen en la decisión de sus dueños a la hora de contratar servicios de veterinaria, peluquería, entrenamiento, transporte, etc. No obstante, centrándose en el ámbito de transporte, la falta de opciones confiables y seguras para movilizar a las mascotas hace que sea difícil para los dueños planificar sus viajes de manera efectiva.

Objetivos

- Desarrollar una aplicación de escritorio fundamentada en Java o Python, que simule la reservación de viajes para animales domésticos teniendo en cuenta los estándares de los sistemas transaccionales.
- Crear una base de datos en el sistema de gestión Microsoft SQL Server, con el fin de almacenar, administrar y relacionar los datos recopilados en la aplicación de escritorio.
- Construir una interfaz gráfica de usuario intuitiva, amigable y simple para integrarla a la aplicación de escritorio.

Descripción del problema

Dentro del contexto descrito antes, es posible evidenciar la falta de un sistema de transporte para animales domésticos en trayectos largos. Es decir, si una persona desea transportarse con su mascota desde Bogotá hacia Medellín, no existe un servicio especial para animales que brinde confianza y seguridad a su dueño. Por lo que, algunas de las alternativas por las que optan los usuarios son: transportar a su mascota de forma aérea, así estas son ubicadas en bodegas o espacios inadecuados dentro de los aviones. O viajan de forma terrestre, sin las mínimas condiciones de seguridad, bienestar o comodidad. En los casos anteriores, las mascotas pueden sufrir diferentes enfermedades o lesiones como: dificultades respiratorias, ansiedad, deshidratación y hasta la muerte.

La solución propuesta para resolver la problemática es el desarrollo de una aplicación de escritorio para el transporte de mascotas. La aplicación permitirá a los clientes reservar y planificar el viaje de sus mascotas de una manera sencilla y eficiente.

Los clientes tendrán la opción de elegir entre dos medios de transporte, avión o autobús, y seleccionar el destino deseado. Las tarifas serán dinámicas y variarán de acuerdo con la temporada, el medio de transporte elegido y la distancia del viaje.

Además, durante los viajes, las mascotas serán acompañadas por profesionales veterinarios altamente capacitados, garantizando su seguridad y bienestar en todo momento.

En resumen, la aplicación de transporte de mascotas ofrecerá una solución completa y confiable para los dueños de mascotas que desean viajar con sus animales. Con la automatización de los procesos de reserva y planificación, los clientes podrán disfrutar de una experiencia de viaje sin estrés para ellos y sus mascotas.

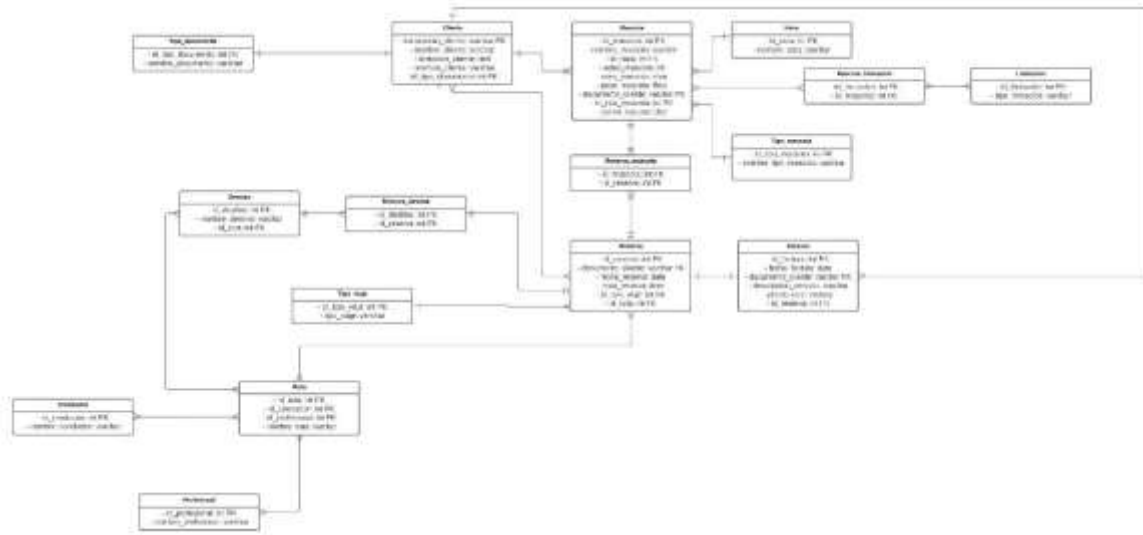
La arquitectura del sistema propuesto incluiría los siguientes componentes:

1. Interfaz de usuario: Se desarrollará una aplicación de escritorio fácil de usar, con una interfaz gráfica intuitiva que permita a los clientes reservar y planificar el viaje de sus mascotas.
2. Base de datos: Una base de datos segura almacenará la información de los clientes, las reservas, rutas, etc.

Para el desarrollo de esta arquitectura, se pueden utilizar varias tecnologías de software, tales como lenguajes de programación (Java o Python). Se gestionaría la base de datos relacional con Microsoft SQL Server para almacenar y gestionar información. Además de emplear como entorno de desarrollo NetBeans, en su versión 8.2.

En resumen, la arquitectura del sistema propuesto es segura y eficiente, además de permitir a los clientes planificar los viajes de sus mascotas de manera sencilla y confiable.

A continuación, se presentará el diagrama ER con sus respectivas entidades, atributos, llaves primarias, foráneas y cardinalidades:



Para llevar a cabo este proyecto, su puesta en marcha se dividirá en dos partes, la primera de ellas será todo lo relacionado con la base de datos, desde su creación hasta procedimientos almacenados. Luego, se desarrollará la aplicación de escritorio con las características antes mencionadas.

Para la creación de la base de datos se usó el sistema de gestión de bases de datos Microsoft SQL Server Management Studio. Entonces, se crearon las tablas anteriormente presentadas en el DER, con sus respectivos atributos, llaves primarias, llaves foráneas, tablas intermedias, etc.

A continuación, se presenta la documentación de cada tabla en el script correspondiente:

```
CREATE TABLE Cliente
(
    documento_cliente varchar(50) PRIMARY KEY NOT NULL,
    nombre_cliente varchar(25) NOT NULL,
    direccion_cliente text NOT NULL,
    telefono_cliente varchar(10) NOT NULL,
    tipo_documento int NOT NULL
)
GO
ALTER TABLE Cliente ADD foreign key(tipo_documento) references Tipo_documento(id_tipo_documento);
/*Asignación de llave foránea "tipo_documento" que referencia a la tabla "Tipo_documento".*/
```

/*Creación de la tabla "Cliente"*/

/*Declaración de columna "documento_cliente" como una variable de tipo varchar. Ya que, se trata de una cadena de caracteres y no realizarán operaciones numéricas. Además de la sentencia "NOT NULL" que no permite tener valores nulos dentro de esta columna. Se define como llave foránea*/

/*Declaración de columna "nombre_cliente" como una variable de tipo varchar. Ya que, se trata de una cadena de caracteres de tamaño variable. Además de la sentencia "NOT NULL" que no permite tener valores nulos dentro de esta columna.*/

/*Declaración de columna "direccion_cliente" como una variable de tipo text. Ya que, se trata de una cadena de caracteres. Además de la sentencia "NOT NULL" que no permite tener valores nulos dentro de esta columna.*/

/*Declaración de columna "telefono_cliente" como una variable de tipo varchar. Ya que, se trata de una cadena de caracteres y no realizarán operaciones numéricas. Además de la sentencia "NOT NULL" que no permite tener valores nulos dentro de esta columna.*/

/*Declaración de columna "tipo_documento" como una variable de tipo entero. Además de la sentencia "NOT NULL" que no permite tener valores nulos dentro de esta columna.*/

Script 1. Tabla "Cliente".

```
CREATE TABLE Conductor
(
    id_conductor int IDENTITY (1,1) PRIMARY KEY NOT NULL,
    nombre_conductor varchar(25) NOT NULL
)
GO
```

/*Creación de la tabla "Conductor"*/

/*Declaración de columna "id_conductor" como una variable de tipo entero. Aquí se usa "IDENTITY" para autoincremento de la misma. Además de la sentencia "NOT NULL" que no permite tener valores nulos dentro de esta columna.*/

/*Declaración de la columna "nombre_conductor" como una variable de tipo varchar, debido a que se trata de una cadena de caracteres variable. Además de la sentencia "NOT NULL" que no permite tener valores nulos dentro de esta columna.*/

Script 2. Tabla "Conductor".

```

CREATE TABLE Destino
/*Creación de la tabla "Destino"*/
(
  id_destino int PRIMARY KEY NOT NULL,
  /*Declaración de columna "id_destino" como una variable de tipo entero.
  Aquí se usa "IDENTITY" para autoincremento de la misma. Además de la sentencia
  "NOT NULL" que no permite tener valores nulos dentro de esta columna.*/

  nombre_destino varchar(25) NOT NULL,
  /*Declaración de la columna "nombre_destino" como una variable de tipo varchar,
  debido a que se trata de una cadena de caracteres variable. Además de la sentencia
  "NOT NULL" que no permite tener valores nulos dentro de esta columna.*/

  id_ruta int NOT NULL,
  /*Declaración de columna "id_ruta" como una variable de tipo entero. Además de la
  sentencia "NOT NULL" que no permite tener valores nulos dentro de esta columna.*/
)
GO

ALTER TABLE Destino ADD foreign key(id_ruta) references Ruta(id_ruta); /*Asignación de llave foránea "id_ruta" que
referencia la tabla "Ruta"*/

```

Script 3. Tabla "Destino".

```

CREATE TABLE Factura
/*Creación de la tabla "Factura"*/
(
  id_factura int IDENTITY (1,1) PRIMARY KEY NOT NULL,
  /*Declaración de columna "id_factura" como una variable de tipo entero.
  Aquí se usa "IDENTITY" para autoincremento de la misma. Además de la
  sentencia "NOT NULL" que no permite tener valores nulos dentro de esta
  columna.*/

  fecha_factura date NOT NULL,
  /*Declaración de la columna "fecha_factura" como una variable de tipo
  date, ya que se trata de una fecha. Además de la sentencia "NOT NULL"
  que no permite tener valores nulos dentro de esta columna.*/

  descripcion_servicio text NOT NULL,
  /*Declaración de columna "descripcion_servicio" como una variable de tipo
  Ya que, se trata de una cadena de caracteres. Además de la sentencia
  "NOT NULL" que no permite tener valores nulos dentro de esta columna.*/

  precio_total int NOT NULL,
  /*Declaración de columna "precio_total" como una variable de tipo entero.
  Además de la sentencia "NOT NULL" que no permite tener valores nulos
  dentro de esta columna.*/

  documento_cliente varchar(50) NOT NULL,
  /*Declaración de columna "documento_cliente" como una variable de tipo
  varchar. Ya que, se trata de una cadena de caracteres y no realizarán
  operaciones numéricas. Además de la sentencia "NOT NULL" que no
  permite tener valores nulos dentro de esta columna.*/

  id_reserva int NOT NULL,
  /*Declaración de columna "id_reserva" como una variable de tipo entero.
  Además de la sentencia "NOT NULL" que no permite tener valores nulos
  dentro de esta columna.*/
)
GO

ALTER TABLE Factura ADD foreign key(documento_cliente) references Cliente(documento_cliente); /*Asignación de llave foránea
"documento_cliente" que referencia a la
tabla "Cliente".*/

ALTER TABLE Factura ADD foreign key(id_reserva) references Reserva(id_reserva); /*Asignación de llave foránea
"id_reserva" que referencia a la tabla
"Reserva".

```

Script 4. Tabla "Factura"

```

CREATE TABLE Limitacion
/*Creación de la tabla "Limitacion"*/
(
  id_limitacion int IDENTITY(1,1) PRIMARY KEY NOT NULL,
  /*Declaración de columna "id_limitacion" como una variable de tipo entero.
  Aquí se usa "IDENTITY" para autoincremento de la misma. Además de la sentencia
  "NOT NULL" que no permite tener valores nulos dentro de esta columna.*/

  tipo_limitacion varchar(25) NOT NULL,
  /*Declaración de la columna "tipo_limitacion" como una variable de tipo varchar,
  debido a que se trata de una cadena de caracteres variable. Además de la sentencia
  "NOT NULL" que no permite tener valores nulos dentro de esta columna.*/
)
GO

```

Script 5. Tabla "Limitación"


```

CREATE TABLE Mascota
(
  id_mascota int IDENTITY (1,1) PRIMARY KEY NOT NULL,

  nombre_mascota varchar(25) NOT NULL,

  edad_mascota int NOT NULL,

  sexo_mascota char(1) NOT NULL,

  peso_mascota float NOT NULL,

  carnet_vacuna char(1) NOT NULL,

  id_raza int NOT NULL,

  documento_cliente varchar(50) NOT NULL,

  id_tipo_mascota int NOT NULL,

)
GO

/*Creación de la tabla "Mascota"*/
/*Declaración de columna "id_mascota" como una variable de tipo entero.
Aquí se usa "IDENTITY" para autoincremento de la misma. Además de la sentencia
"NOT NULL" que no permite tener valores nulos dentro de esta columna.*/

/*Declaración de la columna "nombre_mascota" como una variable de tipo varchar,
debido a que se trata de una cadena de caracteres variable. Además de la sentencia
"NOT NULL" que no permite tener valores nulos dentro de esta columna.*/

/*Declaración de columna "edad_mascota" como una variable de tipo entero. Además
de la sentencia "NOT NULL" que no permite tener valores nulos dentro de esta
columna.*/

/*Declaración de columna "sexo_mascota" como una variable de tipo char, ya que
se trata de un carácter 'M' (Masculino) o 'F' (Femenino). Además de la sentencia
"NOT NULL" que no permite tener valores nulos dentro de esta columna.*/

/*Declaración de columna "peso_mascota" como una variable de tipo flotante, ya
que se trata de datos aproximados. Además de la sentencia "NOT NULL" que no
permite tener valores nulos dentro de esta columna.*/

/*Declaración de columna "carnet_vacuna" como una variable de tipo char, ya que
se trata de un carácter 'S' (Si) o 'N' (No). Además de la sentencia "NOT NULL"
que no permite tener valores nulos dentro de esta columna.*/

/*Declaración de columna "id_raza" como una variable de tipo entero. Además de
la sentencia "NOT NULL" que no permite tener valores nulos dentro de esta
columna.*/

/*Declaración de columna "documento_cliente" como una variable de tipo varchar.
Ya que, se trata de una cadena de caracteres y no realizarán operaciones
numéricas. Además de la sentencia "NOT NULL" que no permite tener valores nulos
dentro de esta columna.*/

/*Declaración de columna "id_tipo_mascota" como una variable de tipo entero.
Además de la sentencia "NOT NULL" que no permite tener valores nulos dentro
de esta columna.*/

ALTER TABLE Mascota ADD foreign key(id_raza) references Raza(id_raza); /*Asignación de llave foránea "id_raza" que referencia la tabla "Raza"*/
ALTER TABLE Mascota ADD foreign key(id_tipo_mascota) references Tipo_mascota(id_tipo_mascota); /*Asignación de la llave foránea "id_tipo_mascota"
que referencia la tabla "Tipo_mascota"*/
ALTER TABLE Mascota ADD foreign key(documento_cliente) references Cliente(documento_cliente); /*Asignación de la llave foránea "documento_cliente"
que referencia la tabla "Cliente"*/

```

Script 6. Tabla "Mascota".

```

CREATE TABLE Mascota_limitacion
(
  id_mascota int NOT NULL,

  id_limitacion int NOT NULL,

  primary key (id_mascota, id_limitacion));

/*Creación de la tabla intermedia "Mascota_limitacion"*/

/*Declaración de columna "id_mascota" como una variable de tipo entero. Además de la
sentencia "NOT NULL" que no permite tener valores nulos dentro de esta columna.*/

/*Declaración de columna "id_limitacion" como una variable de tipo entero.
Además de la sentencia "NOT NULL" que no permite tener valores nulos dentro
de esta columna.*/

/*Declaración de llave primaria compuesta*/

ALTER TABLE Mascota_limitacion ADD foreign key (id_mascota) references Mascota(id_mascota); /*Declaración de llave foránea "id_mascota" que
referencia la tabla "Mascota"*/

ALTER TABLE Mascota_limitacion ADD foreign key (id_limitacion) references Limitacion(id_limitacion); /*Declaración de llave foránea "id_limitacion"
que referencia la tabla "Limitacion"*/

GO

```

Script 7. Tabla intermedia "Mascota_limitacion".

```

CREATE TABLE Profesional
(
  id_profesional int IDENTITY (1,1) PRIMARY KEY NOT NULL,

  nombre_profesional varchar(25) NOT NULL,

)
GO

/*Creación de la tabla "Profesional"*/

/*Declaración de columna "id_profesional" como una variable de tipo entero.
Aquí se usa "IDENTITY" para autoincremento de la misma. Además de la
sentencia "NOT NULL" que no permite tener valores nulos dentro de esta
columna.*/

/*Declaración de la columna "nombre_profesional" como una variable de tipo varchar,
debido a que se trata de una cadena de caracteres variable. Además de la sentencia
"NOT NULL" que no permite tener valores nulos dentro de esta columna.*/

```

Script 8. Tabla "Profesional".

```

CREATE TABLE Raza /* Esta línea comienza la creación de una nueva tabla llamada "Raza".*/
(
  id_raza int identity(1,1) PRIMARY KEY NOT NULL, /* Esta línea crea una columna llamada "id_raza" con el tipo de datos "int". la cláusula IDENTITY
  indica que esta columna es una clave primaria que se generará automáticamente en orden ascendente
  a medida que se agreguen nuevas filas a la tabla. La cláusula NOT NULL indica que esta columna no
  puede contener valores NULL.*/
  nombre_raza varchar(25) NOT NULL, /* Esta línea crea una segunda columna llamada "nombre_raza" con el tipo de datos "varchar".
  La longitud máxima de esta columna es de 25 caracteres. la cláusula NOT NULL indica que
  esta columna no puede contener valores NULL.*/
)
GO

```

Script 9. Tabla "Raza".

```

CREATE TABLE Reserva /* Esta línea comienza la creación de una nueva tabla llamada "Reserva".*/
(
  id_reserva int identity(1,1) PRIMARY KEY NOT NULL, /* Una columna entera que se autoincrementa en 1 para cada nueva fila, y no puede ser nula. */
  fecha_reserva date NOT NULL, /* Una columna de fecha que no puede ser nula. */
  hora_reserva time NOT NULL, /* Una columna de tiempo que no puede ser nula.*/
)
GO
ALTER TABLE Reserva ADD foreign key(id_tipo_viaje) references Tipo_viaje(id_tipo_viaje); /* Esta línea agrega a la tabla Reserva una llave foránea de la tabla Tipo_viaje
referenciando al id_tipo_viaje */
ALTER TABLE Reserva ADD foreign key(documento_cliente) references Cliente(documento_cliente); /* Esta línea agrega a la tabla Reserva una llave foránea de la tabla Cliente
referenciando al documento_cliente */
ALTER TABLE Reserva ADD foreign key(id_ruta) references Ruta(id_ruta); /* Esta línea agrega a la tabla Reserva una llave foránea de la tabla Ruta
referenciando al id_ruta */

```

Script 10. Tabla "Reserva".

```

CREATE TABLE Reserva_destino /* Esta línea comienza la creación de una nueva tabla intermedia llamada "Reserva_destino".*/
(
  id_reserva int NOT NULL, /* Una columna entera que no puede ser nula. */
  id_destino int NOT NULL, /* Una columna entera que no puede ser nula. */
  primary key (id_reserva, id_destino); /* Se asignan los dos atributos como una llave primaria */
)
ALTER TABLE Reserva_destino ADD foreign key(id_reserva) references Reserva(id_reserva); /* Se referencia cada atributo de la tabla con su respectiva tabla principal, en este caso
Reserva*/
ALTER TABLE Reserva_destino ADD foreign key(id_destino) references Destino(id_destino); /* Se referencia cada atributo de la tabla con su respectiva tabla principal, en este caso
Destino*/
GO

```

Script 11. Tabla intermedia "Reserva_destino".

```

CREATE TABLE Reserva_mascota /* Esta línea comienza la creación de una nueva tabla llamada "Reserva_mascota".*/
(
  id_reserva int NOT NULL, /* Una columna entera que no puede ser nula*/
  id_mascota int NOT NULL, /* Una columna entera que no puede ser nula*/
  primary key(id_reserva, id_mascota); /* Se le asignan los dos atributos como una llave primaria*/
)
ALTER TABLE Reserva_mascota ADD foreign key(id_reserva) references Reserva(id_reserva); /* Se referencia cada atributo de la tabla con su respectiva tabla principal, en este caso
Reserva*/
ALTER TABLE Reserva_mascota ADD foreign key(id_mascota) references Mascota(id_mascota); /* Se referencia cada atributo de la tabla con su respectiva tabla principal, en este caso
Mascota*/
GO

```

Script 12. Tabla intermedia "Reserva_mascota".

```
--CREATE TABLE Ruta /* Esta línea comienza la creación de una nueva tabla llamada "Ruta".*/  
  
    (id_ruta int identity(1,1) NOT NULL, /* Una columna entera que funciona como clave primaria y es autoincremental (IDENTITY).*/  
    nombre_ruta varchar(25) NOT NULL, /* Una columna de texto de longitud 25 que no puede ser nula.*/  
    id_profesional int NOT NULL, /*Una columna entera que hace referencia a una llave primaria de otra tabla que puede ser nula.*/  
    id_conductor int NOT NULL, /*Una columna entera que hace referencia a una llave primaria de otra tabla que puede ser nula.*/  
  
    ALTER TABLE Ruta ADD foreign key(id_profesional) references Profesional(id_profesional); /* Se referencia cada atributo de la tabla con su respectiva tabla principal, en este caso  
    Profesional*/  
    ALTER TABLE Ruta ADD foreign key(id_conductor) references Conductor(id_conductor); /* Se referencia cada atributo de la tabla con su respectiva tabla principal, en este caso  
    Conductor*/  
  
GO
```

Script 13. Tabla "Ruta".

```
--CREATE TABLE Tipo_documento /* Esta línea comienza la creación de una nueva tabla llamada "Tipo_documento".*/  
  
    (id_tipo_documento int identity(1,1) NOT NULL, /* Una columna entera que funciona como clave primaria y es autoincremental (IDENTITY).*/  
    nombre_documento varchar(25) NOT NULL, /* Una columna de texto de longitud 25 que no puede ser nula.*/  
  
GO
```

Script 14. Tabla "Tipo_documento".

```
--CREATE TABLE Tipo_mascota /* Esta línea comienza la creación de una nueva tabla llamada "Tipo_mascota".*/  
  
    (id_tipo_mascota int identity(1,1) NOT NULL, /* Una columna entera que funciona como clave primaria y es autoincremental (IDENTITY).*/  
    tipo_mascota varchar(25) NOT NULL, /* Una columna de texto de longitud 25 que no puede ser nula.*/  
  
GO
```

Script 15. Tabla "Tipo_mascota".

```
--CREATE TABLE Tipo_viaje /* Esta línea comienza la creación de una nueva tabla llamada "Tipo_viaje".*/  
  
    (id_tipo_viaje int identity(1,1) NOT NULL, /* Una columna entera que funciona como clave primaria y es autoincremental (IDENTITY).*/  
    tipo_viaje varchar(25) NOT NULL, /* Una columna de texto de longitud 25 que no puede ser nula.*/  
  
GO
```

Script 16. Tabla "Tipo_viaje".

Luego, ya teniendo las tablas necesarias en la base de datos, el siguiente paso será la creación de procedimientos almacenados que permitan contener un conjunto de instrucciones predeterminadas con el fin de reutilizar código, abstracción en la base de datos, etc. Se han creado dos Stored Procedures, a continuación, se presenta la descripción de estos:

- Stored Procedure para la generación de un recibo.

Luego de que el cliente realice una reserva, puede generar el recibo de esta, este contendrá información como su nombre y el de la mascota, la ruta que seleccionó, la fecha de la reserva y el valor total. El script se presenta así:

```
--CREATE PROCEDURE sp_generar_recibo
--Crea el procedimiento almacenado "sp_generar_recibo"
@id_reserva int
--Definición del parámetro de entrada llamado "@id_reserva" de tipo "int"
AS
--Inicio del bloque de instrucciones.
BEGIN
    DECLARE @nombre_cliente varchar(25)
    -- Declara la variable "@nombre_cliente" de tipo "varchar(25)"
    DECLARE @nombre_mascota varchar(25)
    -- Declara la variable "@nombre_mascota" de tipo "varchar(25)"
    DECLARE @documento_cliente int
    -- Declara la variable "@documento_cliente" de tipo "int"
    DECLARE @telefono_cliente varchar(10)
    -- Declara la variable "@telefono_cliente" de tipo "varchar(10)"
    DECLARE @nombre_ruta varchar(25)
    -- Declara la variable "@nombre_ruta" de tipo "varchar(25)"
    DECLARE @fecha_reserva date
    -- Declara la variable "@fecha_reserva" de tipo "date"
    DECLARE @precio_total money
    -- Declara la variable "@precio_total" de tipo "money"
    SELECT @nombre_cliente = c.nombre_cliente,
           @documento_cliente = c.documento_cliente,
           @telefono_cliente = c.telefono_cliente,
           @nombre_mascota = MAX(m.nombre_mascota),
           @nombre_ruta = ru.nombre_ruta,
           @fecha_reserva = r.fecha_reserva,
           @precio_total = MAX(f.precio_total)
    FROM Reserva r
    INNER JOIN Cliente c ON c.documento_cliente = r.documento_cliente
    LEFT JOIN Mascota m ON m.id_mascota = r.id_mascota
    INNER JOIN Ruta ru ON ru.id_ruta = r.id_ruta
    INNER JOIN Factura f ON f.id_reserva = r.id_reserva
    WHERE r.id_reserva = @id_reserva
    -- Los datos se traen de la tabla "Reserva" con el alias "r"
    -- Une la tabla "Reserva" con la tabla "Cliente" mediante la columna "documento_cliente".
    -- Une la tabla "Reserva" con la tabla "Mascota" mediante la columna "id_mascota".
    -- Une la tabla "Reserva" con la tabla "Ruta" mediante la columna "id_ruta".
    -- Une la tabla "Reserva" con la tabla "Factura" mediante la columna "id_reserva".
    -- Agrupa los resultados por los campos especificados
    GROUP BY c.nombre_cliente, c.documento_cliente, c.telefono_cliente, ru.nombre_ruta, r.fecha_reserva;
    PRINT 'Recibo de reserva' -- Imprime el título del recibo
    PRINT '-----'
    PRINT 'Cliente: ' + @nombre_cliente -- Imprime el nombre del cliente
    PRINT 'Mascota: ' + @nombre_mascota -- Imprime el nombre de la mascota
    PRINT 'Documento cliente: ' + CONVERT(varchar(10), @documento_cliente) -- Imprime el documento del cliente
    PRINT 'Ruta: ' + @nombre_ruta -- Imprime el nombre de la ruta
    PRINT 'Fecha de reserva: ' + CONVERT(varchar(10), @fecha_reserva) -- Imprime la fecha de la reserva. Función CONVERT
    --para convertir la fecha en formato DATE en una cadena de caracteres de longitud máxima de 10 caracteres.
    PRINT 'Precio total: ' + CONVERT(varchar(20), @precio_total) -- Imprime el precio total de la reserva. Función convert
    --para convertir el valor del precio total en una cadena de caracteres de longitud máxima de 20 caracteres.
    -- Fin del procedimiento almacenado.
END
```

Script 1. Creación de sp_generar_recibo.

- Stored Procedure para cancelar una reserva.

Dentro de este procedimiento almacenado se quiere automatizar el proceso de eliminar una reserva, para esto se debe tener presente el id de la misma, ya que a partir de este dato se hace la eliminación de esta. Este es su script:

```
--CREATE PROCEDURE sp_cancelar_reserva
--Creación de un procedimiento almacenado llamado "sp_cancelar_reserva".
@id_reserva int
--Parámetro de entrada "@id_reserva" de tipo entero
AS
--Inicio bloque de código
BEGIN
    SET NOCOUNT ON;
    -- Clausula NOCOUNT para no contar el número de filas afectadas por las operaciones

    IF NOT EXISTS (SELECT * FROM Reserva WHERE id_reserva = @id_reserva) -- Validar si la reserva existe en la tabla "Reserva" mediante la
    --consulta SELECT y la cláusula WHERE
    BEGIN
        PRINT 'La reserva no existe.';
        RETURN;
    END
    -- Fin ejecución del procedimiento almacenado

    DELETE FROM Reserva WHERE id_reserva = @id_reserva; -- Eliminar la reserva de la tabla "Reserva" mediante la sentencia DELETE y la
    --cláusula WHERE

    PRINT 'La reserva ha sido cancelada exitosamente.'; -- Mensaje de eliminación exitosa
    --Fin bloque de código
END
```

Script 2. Creación de sp_cancelar_reserva.

Adicionalmente, se crearon triggers, que son objetos dentro la base de datos que se encargan de responder automáticamente a un evento específico. El primero de ellos se describe así:

1. Trigger para validar con la edad de la mascota si puede o no hacer una reserva.
 Este trigger se ejecuta en la tabla “Reserva”, dentro de la que luego de hacer una inserción valida si de acuerdo con la edad de la mascota esta puede realizar un viaje. Para lo anterior, simplemente se emplea una estructura de control en la que si la edad de la mascota es menor a 2 años se ejecuta un raiserror y se hace un rollback a la transacción.

Tabla “Reserva” antes de la ejecución del trigger.

	id_reserva	fecha_reserva	hora_reserva	id_tipo_viaje	documento_cliente	id_ruta	id_mascota	id_raza
1	1	2023-04-04	18:00:00.0000000	1	1003654298	1	1	1
2	2	2023-04-05	15:48:00.0000000	1	1004563875	4	2	4
3	3	2023-01-05	17:22:15.0000000	2	1013272772	5	3	15
4	4	2023-01-01	12:25:00.0000000	1	1023475698	2	4	12
5	5	2022-04-05	14:45:00.0000000	1	1024475998	1	5	7
6	6	2022-08-01	09:54:54.0000000	1	1025648795	3	7	16
7	7	2023-01-09	17:20:00.0000000	2	1032796861	1	8	10
8	1015	2023-04-21	12:45:31.0000000	2	1234567894	3	20	3
9	1016	2023-02-23	10:00:00.0000000	1	1014856773	2	1015	4

Tabla 1 Reserva antes de ejecutar trigger reserva_insert.

```

--CREATE TRIGGER tr_reserva_insert
ON Reserva
AFTER INSERT
AS
--BEGIN
--DECLARE @edad_mascota INT;
--SELECT @edad_mascota = edad_mascota FROM Mascota WHERE id_mascota = (SELECT id_mascota FROM inserted);
--IF @edad_mascota < 2
--BEGIN
--    RAISERROR ('No se puede reservar una mascota menor a 2 años', 16, 1);
--    ROLLBACK TRANSACTION;
--END
--END;
--Creación del trigger
--Tabla en la que se va aplicar el trigger
--El trigger se ejecuta luego de una inserción
--Inicio
--Declaración de una variable llamada "@edad_mascota"
--La variable se iguala a la columna
--"edad_mascota" de la tabla "Mascota" para la mascota que se acaba de
--insertar en la tabla "Reserva" usando el "id_mascota".
--La variable de la edad de mascota es validada, si es menor a 2, inicia
--una instrucción
--Mensaje de error
--Revertir la transacción
--fin del trigger
  
```

Messages
 Command completed successfully.
 Completion time: 2023-04-21T10:49:49.397607-05:00

Script 3. Trigger reserva_insert.

Para comprobar el funcionamiento del trigger, primero se realiza una inserción en la tabla “Mascota”, empleando datos como: nombre de la mascota Tomy y edad de 1 año.

```

--INSERT INTO Mascota (nombre_mascota, edad_mascota, sexo_mascota, peso_mascota, carnet_vacuna, id_raza, documento_cliente, id_tipo_mascota)
VALUES ('Tomy', 1, 'H', 54.3, 'S', 7, '1735937373', 1);
  
```

Messages
 (1 row affected)
 Completion time: 2023-04-21T10:49:19.767926-05:00

Script 4. Inserción en la tabla “Mascota”.

El registro se realiza exitosamente en la tabla “Mascota”:

	id_mascota	nombre_mascota	edad_mascota	sexo_mascota	peso_mascota	camet_vacuna	id_ruta	documento_cliente	id_tipo_mascota
1	1	Toby	5	M	45,2	S	1	1003654298	1
2	2	Hachiko	1	F	32	S	4	1004563875	1
3	3	Michi	2	F	18,4	S	15	1013272772	2
4	4	Milo	1	M	13	S	12	1023475698	2
5	5	Rambo	7	M	34,9	S	7	1024475998	1
6	7	Romeo	2	F	10,3	S	16	1025648795	2
7	8	Anwen	1	M	35,2	S	10	1032796861	1
8	9	Jerry	6	F	12	S	15	1042536987	2
9	10	Joe	2	F	11,5	S	13	1045076254	2
10	20	España	4	F	12	N	3	1234567894	1
11	1015	Tomas	3	M	27,4	N	4	1014856773	1
12	1016	Lucas	1	M	17	S	10	1056773092	1
13	1017	Lupe	5	F	27,9	S	5	2749225364	1
14	1018	Tomy	1	M	54,3	S	7	1739937373	1

Tabla 2 Mascota.

Posteriormente, se hace una inserción en la tabla “Reserva” con los datos de la mascota que se acaba de insertar (Tomy). Pero, como la edad de Tomy es de 1 año, se muestra un mensaje de error: “No se puede reservar una mascota menor a 2 años”.

```

INSERT INTO Reserva (fecha_reserva, hora_reserva, id_tipo_viaje, documento_cliente, id_ruta, id_mascota, id_raza)
VALUES ('2023-04-21', '17:59:00', 2, '1739937373', 2, 1018, 1);
  
```

0 %

Messages

(0 rows affected)

Msg 50000, Level 16, State 1, Procedure tr_reserva_insert, Line 10 [Batch Start Line 0]
 No se puede reservar una mascota menor a 2 años
 Msg 3609, Level 16, State 1, Line 1
 The transaction ended in the trigger. The batch has been aborted.

Completion time: 2023-04-21T10:51:06.4797851-05:00

Script 5. Inserción en la tabla “Reserva”.

Así que, en la tabla “Reserva” no se afecta ninguna fila, debido a que se hace un rollback.

	id_reserva	fecha_reserva	hora_reserva	id_tipo_viaje	documento_cliente	id_ruta	id_mascota	id_raza
1	1	2023-04-04	18:00:00.0000000	1	1003654298	1	1	1
2	2	2023-04-05	15:48:00.0000000	1	1004563875	4	2	4
3	3	2023-01-05	17:22:15.0000000	2	1013272772	5	3	15
4	4	2023-01-01	12:25:00.0000000	1	1023475698	2	4	12
5	5	2022-04-05	14:45:00.0000000	1	1024475998	1	5	7
6	6	2022-08-01	09:54:54.0000000	1	1025648795	3	7	16
7	7	2023-01-09	17:20:00.0000000	2	1032796861	1	8	10
8	1015	2023-04-21	12:45:31.0000000	2	1234567894	3	20	3
9	1016	2023-02-23	10:00:00.0000000	1	1014856773	2	1015	4

Tabla 3 Reserva.

2. Trigger para cambiar el estado de una ruta si se elimina una reserva.
 Este trigger se basa en que, si una reserva se elimina, se actualice el estado de la ruta que esta cubría. Es decir, inicialmente, todas las rutas tienen un estado de ocupado. Pero si una reserva se elimina, la ruta que tenía la reserva eliminada libera asientos o lugares.

Tabla "Ruta" antes del trigger.

	id_ruta	nombre_ruta	id_profesional	id_conductor	estado
1	1	Bogotá - Medellín	1	2	Ocupado
2	2	Bogotá - Pasto	2	1	Ocupado
3	3	Bogotá - Cali	3	7	Ocupado
4	4	Bogotá - Manizales	4	8	Ocupado
5	5	Bogotá - Barranquilla	5	10	Ocupado

Tabla 4 "Ruta".

Tabla "Reserva" antes del trigger.

	id_reserva	fecha_reserva	hora_reserva	id_tipo_viaje	documento_cliente	id_ruta	id_mascota	id_raza
1	1	2023-04-04	18:00:00.0000000	1	1003654298	1	1	1
2	2	2023-04-05	15:48:00.0000000	1	1004563875	4	2	4
3	3	2023-01-05	17:22:15.0000000	2	1013272772	5	3	15
4	4	2023-01-01	12:25:00.0000000	1	1023475698	2	4	12
5	5	2022-04-05	14:45:00.0000000	1	1024475998	1	5	7
6	6	2022-08-01	09:54:54.0000000	1	1025648795	3	7	16
7	7	2023-01-09	17:20:00.0000000	2	1032796861	1	8	10
8	1015	2023-04-21	12:45:31.0000000	2	1234567894	3	20	3
9	1016	2023-02-23	10:00:00.0000000	1	1014856773	2	1015	4

Tabla 5 "Reserva".

```

--Creación del trigger
--Tabla en la que se va a ejecutar el trigger.
--Después de eliminar un registro se ejecuta el trigger
CREATE TRIGGER actualizar_estado_ruta
ON Reserva
AFTER DELETE
AS
BEGIN
    UPDATE Ruta
    SET estado = 'Asientos disponibles'
    FROM DELETED d
    WHERE Ruta.id_ruta = d.id_ruta;
END
--Inicio de instrucciones
--Actualizar la tabla "Ruta"
--Poner en la columna de estado 'Asientos disponibles'
--Se especifica la tabla deleted, donde están los registros eliminados
--Condición para actualizar los registros de la tabla "Ruta" que corresponden a los registros
--eliminados de la tabla "Reserva"
--Fin
  
```

Messages

Command completed successfully.

Completion time: 2023-04-21T11:10:06.815600Z-05:00

Script 6. Trigger actualizar_estado_ruta.

Para implementar este trigger, se elimina la reserva con el identificador 4, esta tenía como identificador de ruta 2 (ver tabla 8):

id_reserva	fecha_reserva	hora_reserva	id_tipo_viaje	documento_cl...	id_ruta	id_mascota	id_raza
1	2023-04-04	18:00:00	1	1003654298	1	1	1
2	2023-04-05	15:48:00	1	1004563875	4	2	4
3	2023-01-05	17:22:15	2	1013272772	5	3	15
5	2022-04-05	14:45:00	1	1024475998	1	5	7
6	2022-08-01	09:54:54	1	1025648795	3	7	16
7	2023-01-09	17:20:00	2	1032796861	1	8	10
1015	2023-04-21	12:45:31	2	1234567894	3	20	3
1016	2023-02-23	10:00:00	1	1014856773	2	1015	4

Tabla 6 "Reserva".

Por tanto, al haber eliminado la reserva con el identificador de ruta 2, este debería cambiar su estado de ocupado a “Asientos disponibles”. A continuación, se muestra la actualización de la tabla “Ruta”:

id_ruta	nombre_ruta	id_profesional	id_conductor	estado
1	Bogotá - Medel...	1	2	Ocupado
2	Bogotá - Pasto	2	1	Asientos dispo...
3	Bogotá - Cali	3	7	Ocupado
4	Bogotá - Maniz...	4	8	Ocupado
5	Bogotá - Barran...	5	10	Ocupado

Tabla 7 Ruta.

Finalmente, es indispensable asignar roles dentro de la base de datos para asegurar la integridad y seguridad de estos. Para lo anterior, se realizó la creación de diferentes usuarios y logins, cada uno de ellos con ciertos permisos, tal como se describe a continuación:

Primero se crea un login con credenciales:

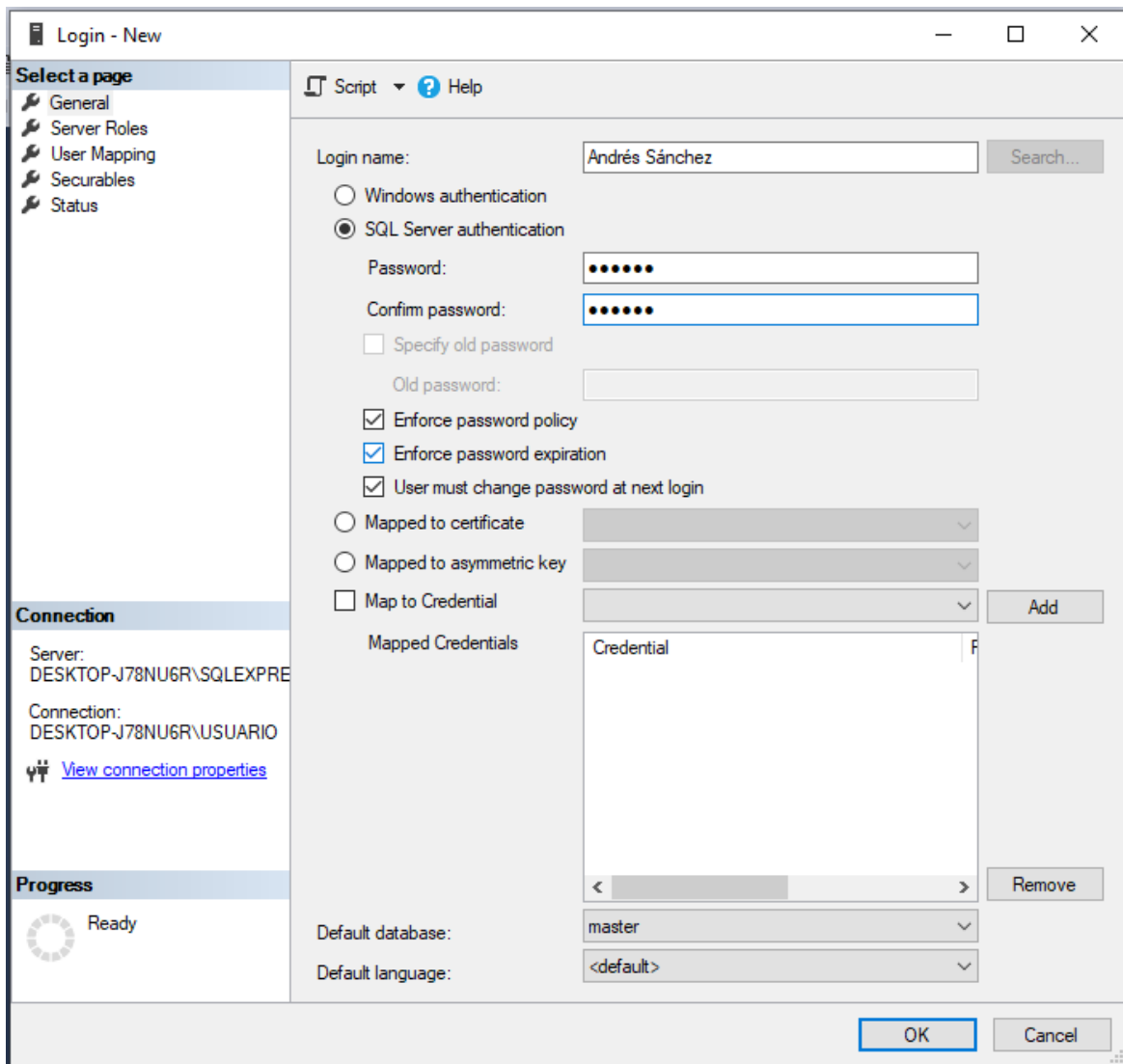


Ilustración 1.

Luego, se crea el usuario correspondiente al login:

Database User - New

Select a page

- General
- Owned Schemas
- Membership
- Securables
- Extended Properties

Script ? Help

User type:
SQL user with login

User name:
Andrés Sánchez

Login name:
Andrés Sánchez

Default schema:
dbo

Connection

Server:
DESKTOP-J78NU6R\SQLEXPRESS

Connection:
DESKTOP-J78NU6R\USUARIO

[View connection properties](#)

Progress

Ready

OK Cancel

Ilustración 2.

Posteriormente se realiza la concesión de permisos a dicho usuario, entre ellos están: inserción, actualización o eliminación de registros dentro de tablas determinadas, tal como se muestra a continuación:

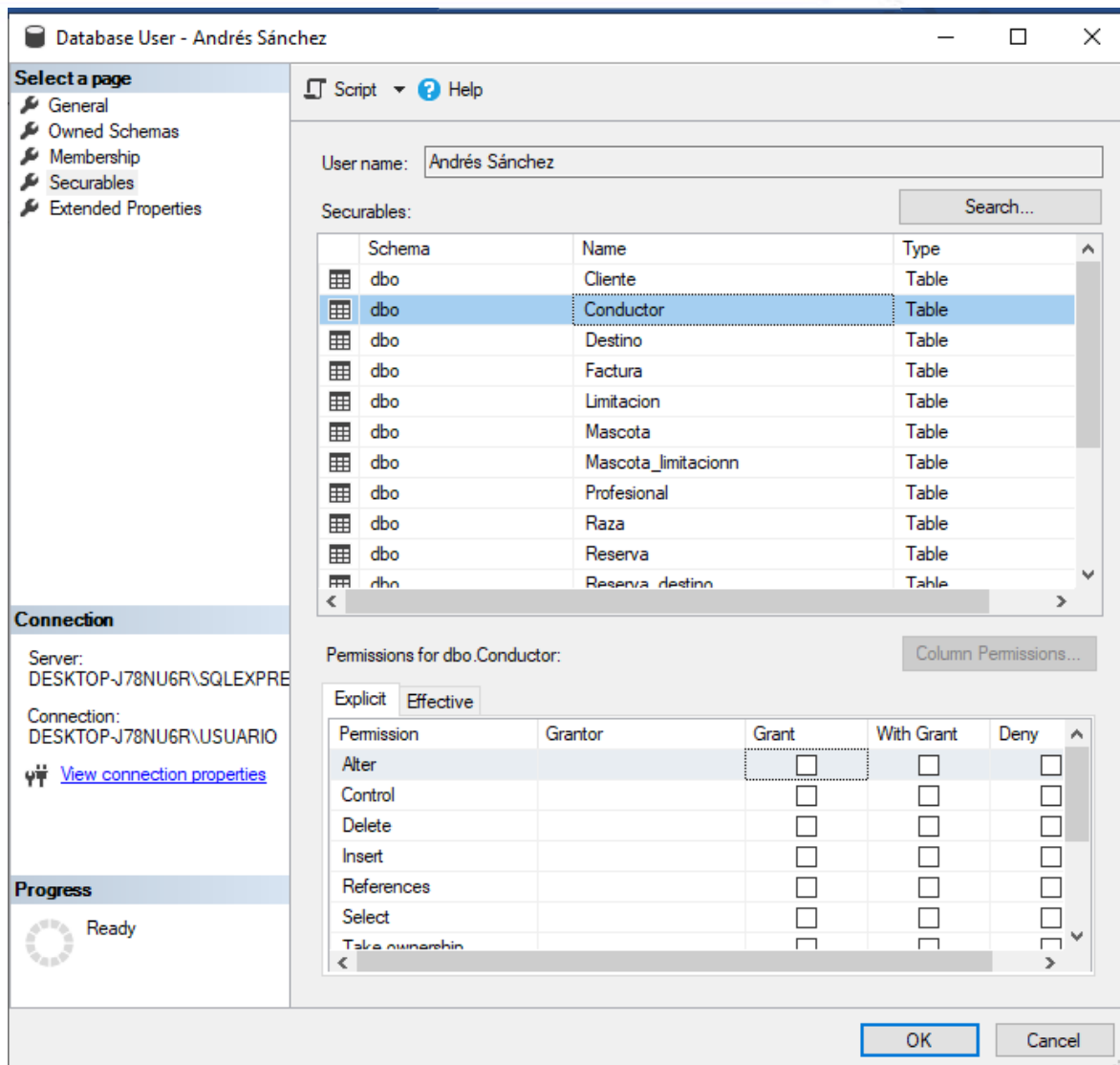


Ilustración 3.

Cabe aclarar, que los pasos correspondientes a login, usuarios y permisos se realizan con varios usuarios, pero se muestra uno de ejemplo. Ahora, se inicia sesión con el usuario creado:

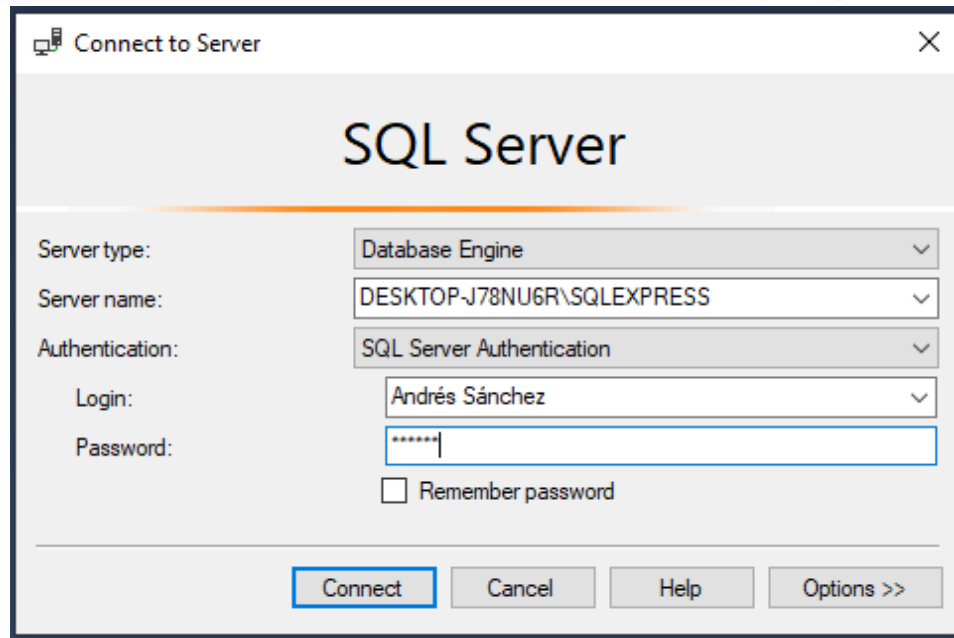


Ilustración 4.

Se inicia, y se verifica que se conectó con el usuario Andrés Sánchez.

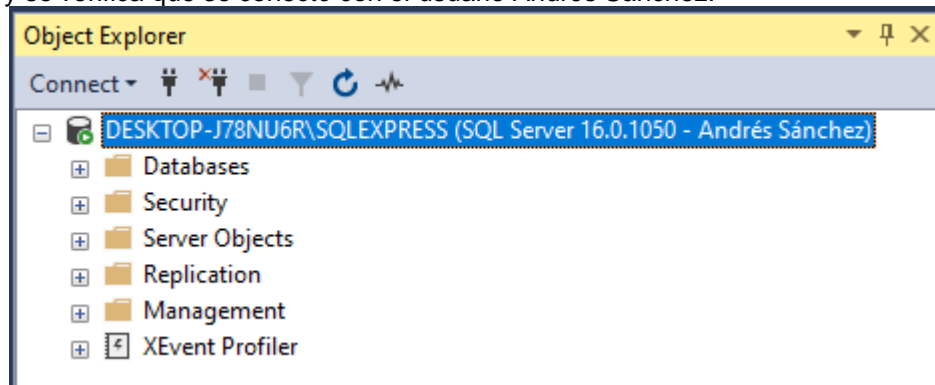
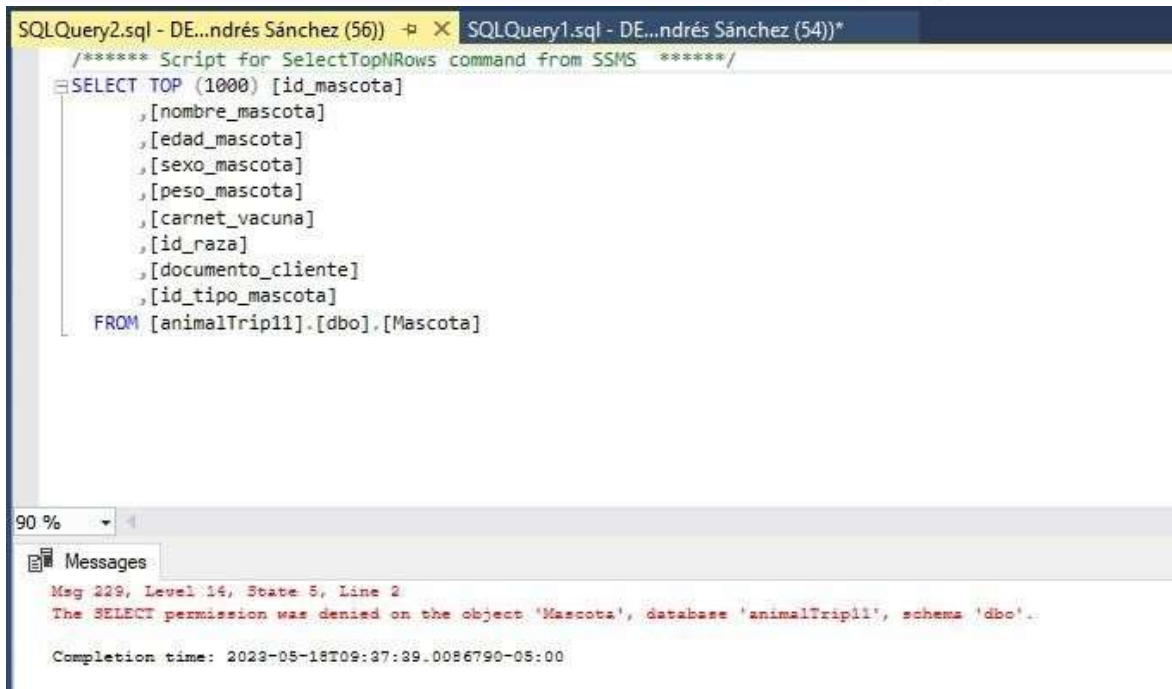


Ilustración 5.

Finalmente, se realiza la comprobación de los permisos otorgados anteriormente. En este caso el usuario Andrés Sánchez sólo puede acceder a las tablas de Conductor y Profesional. Entonces, se intenta ingresar a visualizar la tabla Mascota:



```

SQLQuery2.sql - DE...ndrés Sánchez (56)  SQLQuery1.sql - DE...ndrés Sánchez (54))*
/***** Script for SelectTopNRows command from SSMS *****/
SELECT TOP (1000) [id_mascota]
,[nombre_mascota]
,[edad_mascota]
,[sexo_mascota]
,[peso_mascota]
,[carnet_vacuna]
,[id_raza]
,[documento_cliente]
,[id_tipo_mascota]
FROM [animalTrip11].[dbo].[Mascota]
  
```

90 %

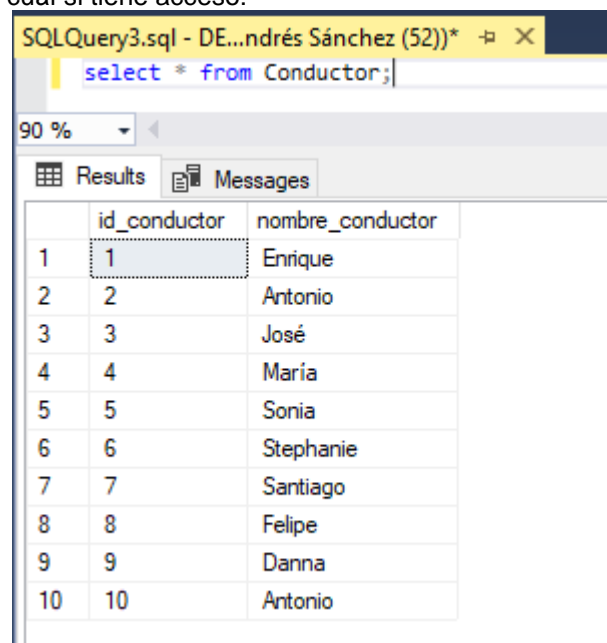
Messages

Msg 229, Level 14, State 5, Line 2
The SELECT permission was denied on the object 'Mascota', database 'animalTrip11', schema 'dbo'.

Completion time: 2023-05-18T09:27:29.0086790-05:00

Ilustración 6.

Como era de esperarse, se muestra un mensaje de acceso denegado. Ahora, se intenta visualizar la tabla Conductor, a la cuál si tiene acceso:



SQLQuery3.sql - DE...ndrés Sánchez (52))*

```
select * from Conductor;
```

90 %

Results Messages

	id_conductor	nombre_conductor
1	1	Enrique
2	2	Antonio
3	3	José
4	4	María
5	5	Sonia
6	6	Stephanie
7	7	Santiago
8	8	Felipe
9	9	Danna
10	10	Antonio

Ilustración 7.

Vemos que esta acción se puede realizar sin ningún problema. Como se mencionó anteriormente, se crearon diferentes usuarios, a continuación, se muestran sus credenciales y los permisos que tiene cada uno.

Andrés Sánchez: andres - Puede manipular las tablas Conductor y Profesional.
 Laura Hernández: laura - Puede manipular la tabla Destino.
 Javier Rojas: javier - Puede manipular la tabla Cliente.

María Torres: maria - Puede manipular las tablas Mascota, Limitación y Raza.
Jhonathan Palacios: jhonathan - Puede manipular las tablas Destino y Ruta.
Héctor Cárdenas: hector - Es el administrador de la BD y tiene acceso a todas las tablas.

Para terminar, estos son los logins registrados en la base de datos:

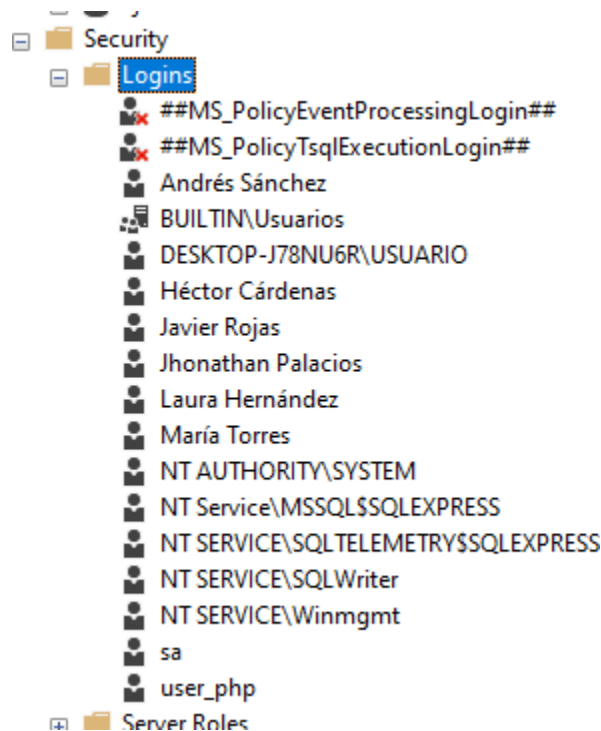


Ilustración 8.

Los usuarios:

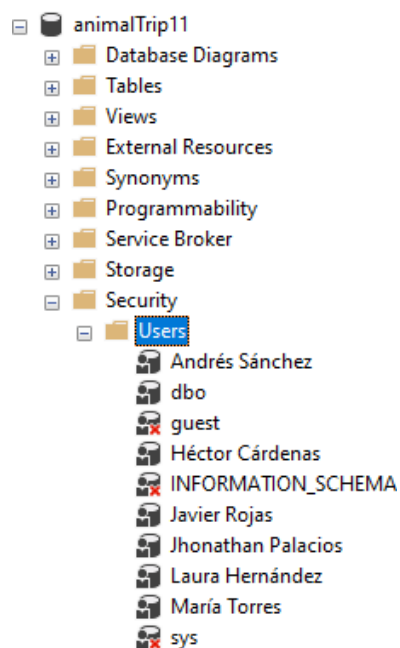


Ilustración 9.

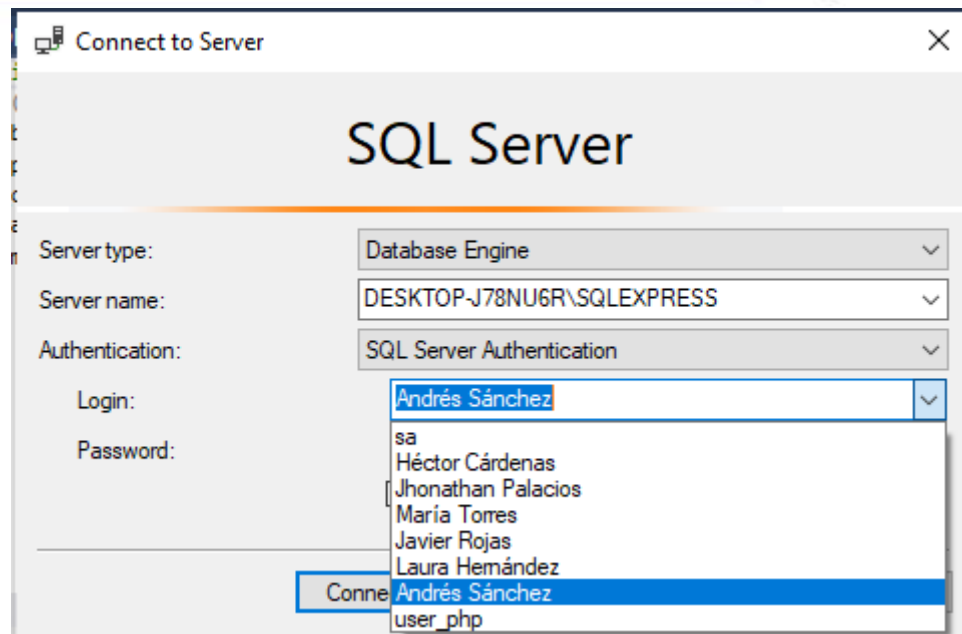


Ilustración 10.

Como segundo y último apartado, se presenta todo lo concerniente al desarrollo de la aplicación de escritorio construida desde NeatBeans. Lo primero que se debe hacer es la configuración del SQL Server para su correcta conexión con el entorno de desarrollo.

1. Tendremos que buscar el índice de SQL Server Network Configuration, en el apartado del SQL desplegamos el Protocols for MSSQLSERVER, y buscamos en el menú de despliegue el TCP/IP que deberá estar Disable, ya está habilitado.

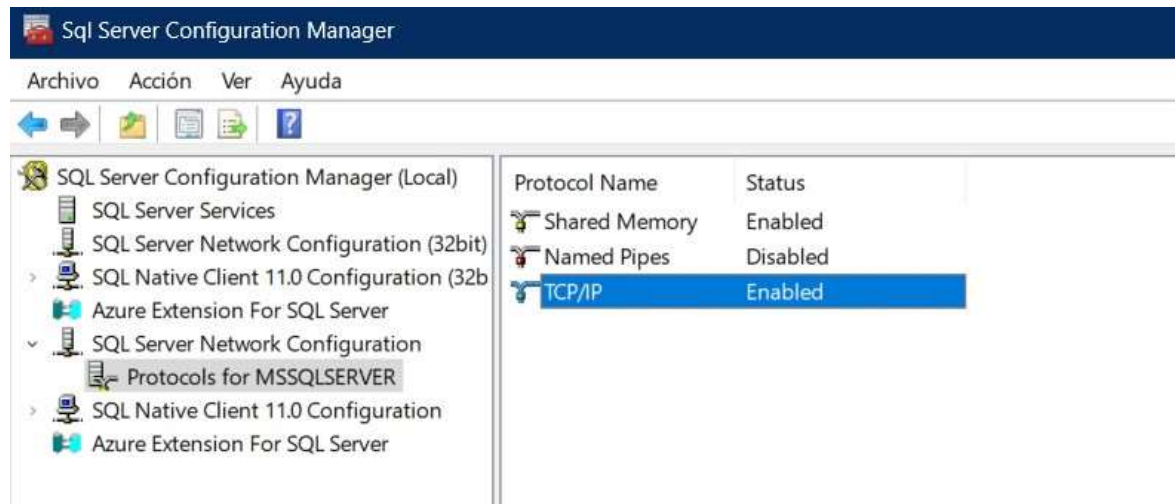


Ilustración 11.

2. Se configura el puerto de escucha de nuestro server SQL, en este caso será el 1433:

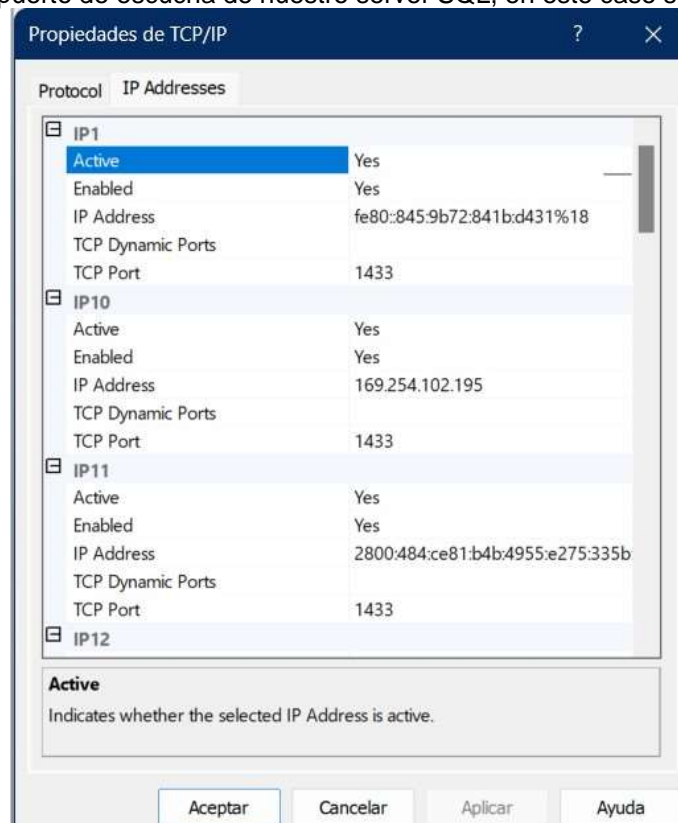


Ilustración 12.

3. Para poder realizar el CRUD de la base de datos de AnimalTrip tendremos que descargar la versión de NetBeans 8.2 la cual no nos presentara problemas de permisos para la conexión. Al igual que su driver desde la página de Microsoft en este caso utilizaremos el “mssql-jdbc-8.2.2.jre8” para la versión de nuestro Server 2022.

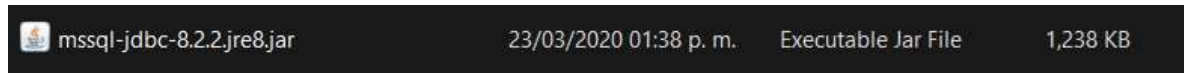


Ilustración 13.

4. Teniendo ya instalado nuestro entorno de desarrollo, procedemos a crear un nuevo proyecto, en este caso se llamará “AnimalTripOficial”:

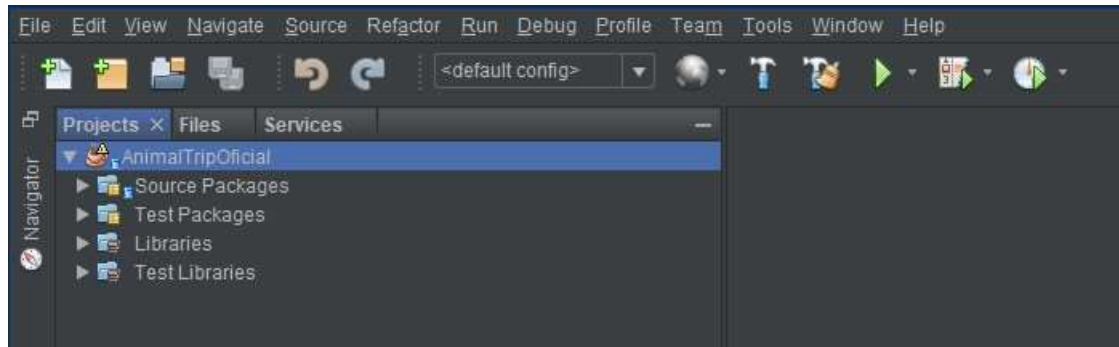


Ilustración 14.

5. Luego, se añade, en el apartado de librerías el archivo del driver que se descargó anteriormente.

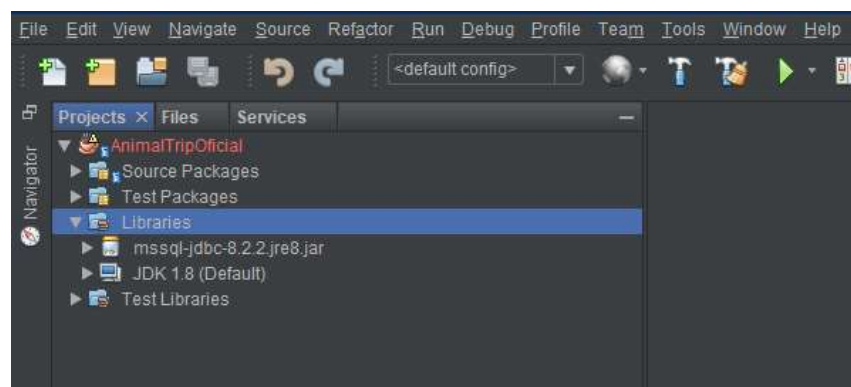


Ilustración 15.

Dentro de la aplicación de java, se pretende desarrollar un sistema de inicio de sesión y gestión tanto de usuarios como de empleados. Cada actor involucrado va a tener diferentes opciones o acciones a realizar dentro de la base de datos. Por ejemplo, un cliente puede registrarse, hacer una reservación, cancelarla, ver las mascotas que tiene adscritas, etc. Mientras que el grupo de empleados se divide en varios de ellos, y adicionalmente, según su rol tienen diferentes permisos. En esta primera parte se abordará la interfaz construida para los clientes.

1. Se usará el modelo MVC como patrón de desarrollo de nuestro CRUD, empezamos por crear un Package llamado Modelo, otro llamado Vista y uno llamado Imágenes. Este último abarcará las imágenes que incorporaremos a las vistas de los JFrame.

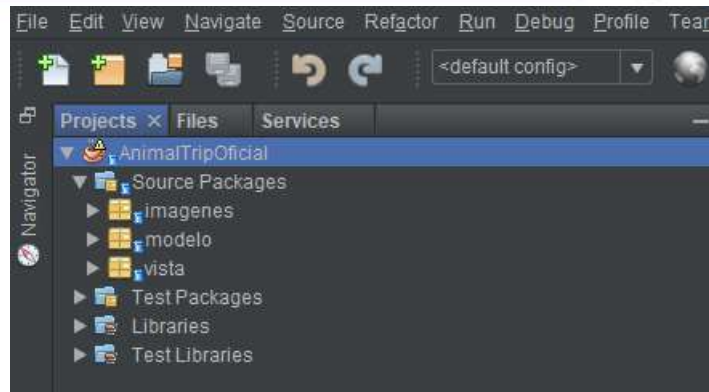


Ilustración 16.

2. En el Package Modelo se crea la clase Conexión (); que será la encargada de realizar el enlace de nuestro IDE con el motor BD Server. En esta clase define el usuario de conexión, la contraseña y la BD a la cual queremos acceder. Además de la información del puerto por el cual se estará comunicando nuestro motor SQL Server.

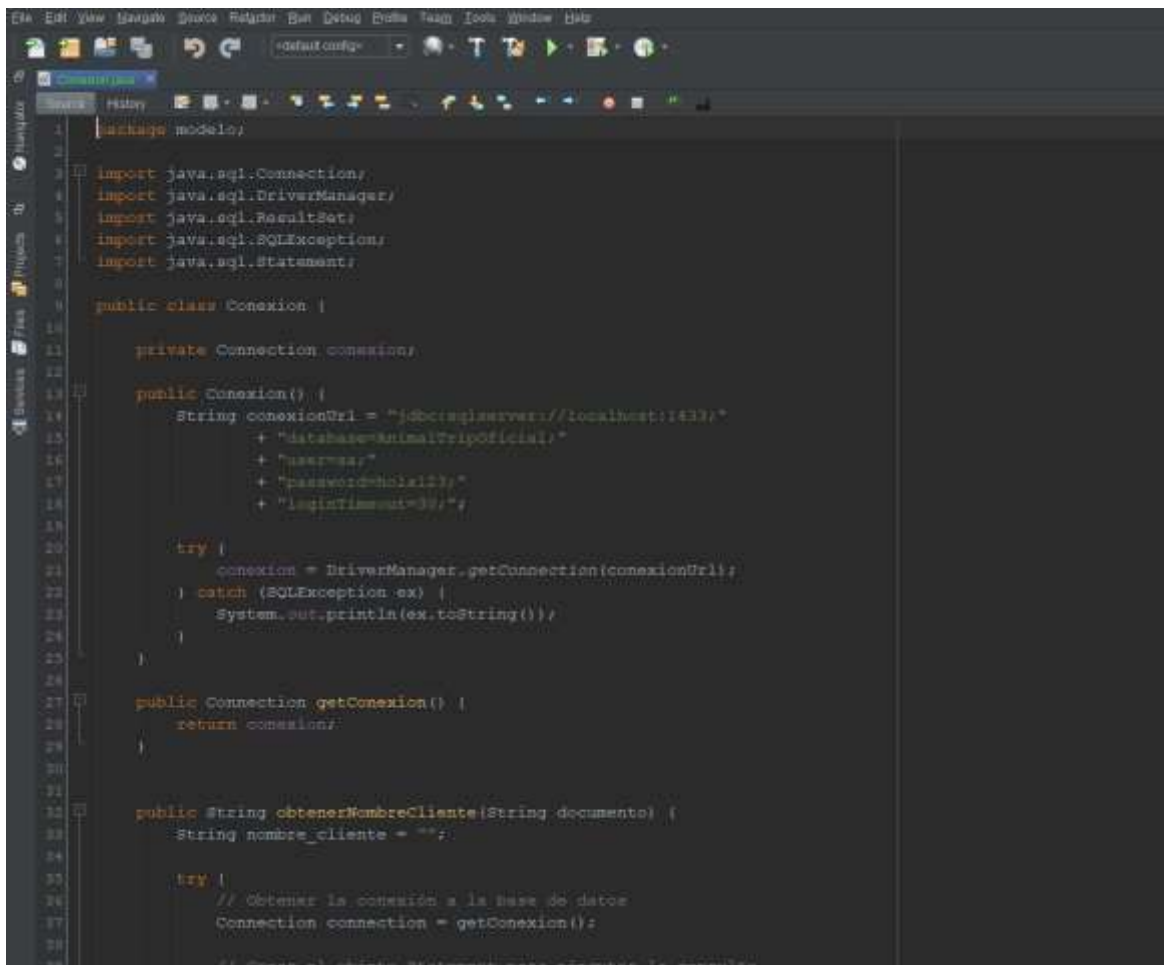


Ilustración 17.

```

40 // Crear el objeto Statement para ejecutar la consulta
41 Statement statement = connection.createStatement();
42
43 // Ejecutar la consulta para obtener el nombre del cliente
44 String query = "SELECT nombre_cliente FROM db_cliente WHERE documento_cliente = '" + documento + "'";
45 ResultSet resultSet = statement.executeQuery(query);
46
47 // Verificar si hay un resultado y obtener el nombre del cliente
48 if (resultSet.next()) {
49     nombre_cliente = resultSet.getString("nombre_cliente");
50 }
51
52 // Cerrar los recursos
53 resultSet.close();
54 statement.close();
55 } catch (SQLException e) {
56     e.printStackTrace();
57 }
58
59 return nombre_cliente;
60 }
61
62 public String obtenerDocumentoCliente(String contraseña) {
63     String documento_cliente = "";
64
65     try {
66         // Obtener la conexión a la base de datos
67         Connection connection = getConexion();
68
69         // Crear el objeto Statement para ejecutar la consulta
70         Statement statement = connection.createStatement();
71
72         // Ejecutar la consulta para obtener el documento del cliente
73         String query = "SELECT documento_cliente FROM db_cliente WHERE contraseña_cliente = '" + contraseña + "'";
74         ResultSet resultSet = statement.executeQuery(query);
75
76         // Verificar si hay un resultado y obtener el nombre del cliente
77         if (resultSet.next()) {
78             documento_cliente = resultSet.getString("documento_cliente");
79         }
80
81         // Cerrar los recursos
82         resultSet.close();
83         statement.close();
84     } catch (SQLException e) {
85         e.printStackTrace();
86     }
87
88     return documento_cliente;
89 }
90 }

```

Ilustración 18.

```

74     if (resultSet.next()) {
75         documento_cliente = resultSet.getString("documento_cliente");
76     }
77
78     // Cerrar los recursos
79     resultSet.close();
80     statement.close();
81 } catch (SQLException e) {
82     e.printStackTrace();
83 }
84
85 return documento_cliente;
86 }
87 }

```

Ilustración 19.

3. En ese mismo Package se crea la clase Login_Validator.java, el cual se empleará más adelante para validar las credenciales de los empleados al momento de iniciar sesión.

```
import com.microsoft.sqlserver.jdbc.SQLServerDataSource;
import com.microsoft.sqlserver.jdbc.SQLServerException;
import java.sql.Connection;
import java.sql.SQLException;

public class Login_Validator {
    private Conexion databaseConnector;

    public Login_Validator(Conexion connector) {
        this.databaseConnector = connector;
    }

    public boolean validateCredentials(String username, String password) throws SQLException {
        // Crea el objeto SQLServerDataSource
        SQLServerDataSource dataSource = new SQLServerDataSource();
        dataSource.setServerName("DESKTOP-J78NU6R\\SQLEXPRESS");
        dataSource.setDatabaseName("animalTrip11");
        dataSource.setUser(username);
        dataSource.setPassword(password);

        try {
            // Establece la conexión
            Connection connection = dataSource.getConnection();
            connection.close();

            return true; // Las credenciales son válidas
        } catch (SQLServerException e) {
            e.printStackTrace();
            return false; // Las credenciales son inválidas
        }
    }
}
```

Ilustración 20.

Luego, esas serían las clases que integran el Package modelo. A continuación, se presentan los frames que integran el Package vista. Como se mencionó antes, se describirán primero los relacionados con el cliente.

1. En el package Vista se crea un JFrame llamado MainMenu.java en el cual se implementará el registro de clientes y botones de inicio de sesión tanto de clientes como de empleados. Tal como se muestra a continuación:

```
btnIniciarSesionCliente.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Conexion databaseConnector = new Conexion();
        Login_Client iniciar_cliente = new Login_Client(databaseConnector);
        iniciar_cliente.setVisible(true);
        dispose(); // Cierra la ventana actual
    }
});

btnIniciarSesionEmpleado.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Login_Employee iniciar_empleado = new Login_Employee();
        iniciar_empleado.setVisible(true);
        dispose(); // Cierra la ventana actual
    }
});

btnRegistrarCliente.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        NewClient registrar = new NewClient();
        registrar.setVisible(true);
        dispose(); // Cierra la ventana actual
    }
});

btnSalir.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});
```

Ilustración 21.

2. Teniendo un frame para el menú de inicio así:



Ilustración 22.

3. Adicionalmente, se crea un frame Login_Client.java, en el cuál se reciben y validan las credenciales del cliente. Para esto se pide el número de documento registrado y la contraseña que el cliente selecciona al momento de registrarse. La interfaz es:

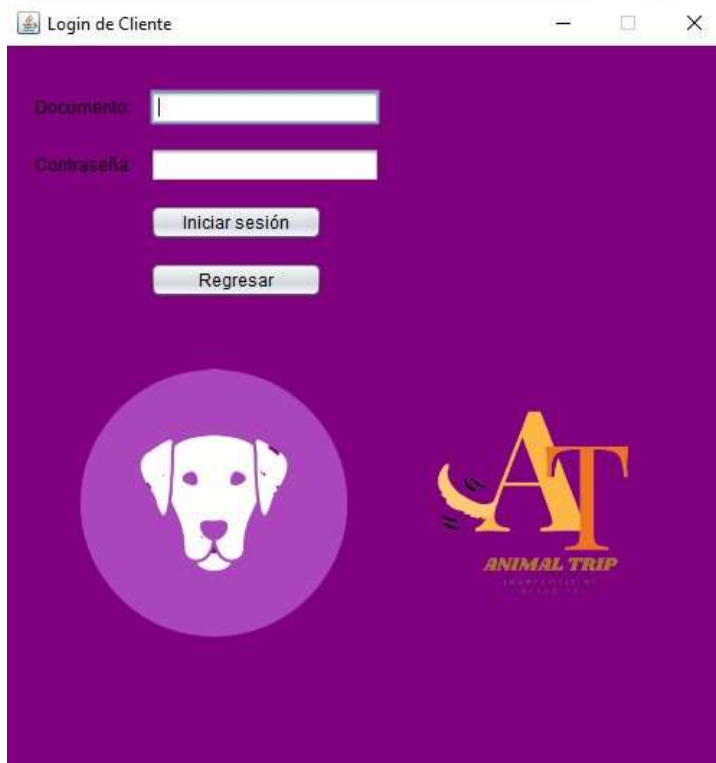


Ilustración 23.

4. Dentro de este frame, los datos se validan a través del listener del botón “Iniciar sesión” y una instancia de la conexión a la BD, tal como se muestra:

```
btnLogin.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        String documento = txtDocumento.getText();
        String contraseña = new String(txtContraseña.getPassword());
        String nombre_cliente = databaseConnector.obtenerNombreCliente(documento);
        String documento_cliente = databaseConnector.obtenerDocumentoCliente(contraseña);

        // Validar las credenciales del cliente
        boolean isValidCredentials = validateCredentials(documento, contraseña);

        if (isValidCredentials) {
            // Credenciales válidas, mostrar mensaje de éxito y realizar acciones adicionales
            // JOptionPane.showMessageDialog(null, "Inicio de sesión exitoso");
            String nombreCliente = databaseConnector.obtenerNombreCliente(documento);
            String documentoCliente = databaseConnector.obtenerDocumentoCliente(contraseña);

            // Crear una instancia de ClientMenu y pasar el nombre del cliente como parámetro
            ClientMenu clientMenu = new ClientMenu(nombre_cliente, documento_cliente);
            clientMenu.setVisible(true);
            // Cerrar el frame actual (LoginCliente)
            dispose();
        } else {
            // Credenciales inválidas, mostrar mensaje de error
            JOptionPane.showMessageDialog(null, "Credenciales inválidas");
        }
    }
});
```

Ilustración 24.

5. Luego de realizar este proceso, entra en juego otro frame ClientMenu.java, en el que se muestran las operaciones que el cliente puede realizar, además de su nombre e identificación:



Ilustración 25.

6. Cada uno de los botones tiene un listener que se encargará de realizar acciones específicas:

```
btnHacerReserva.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Cerrar el JFrame actual
        dispose();
        CreateReservation reservacion = new CreateReservation(nombre_cliente, documento_cliente);
        reservacion.setVisible(true);
    }
});

btnCancelarReserva.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Cerrar el JFrame actual
        dispose();
        DeleteReservation cancelarReservacion = new DeleteReservation(nombre_cliente, documento_cliente);
        cancelarReservacion.setVisible(true);
    }
});

btnGenerarRecibo.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Cerrar el JFrame actual
        dispose();
        GenerarRecibo recibo = new GenerarRecibo(nombre_cliente, documento_cliente);
        recibo.setVisible(true);
    }
});
```

Ilustración 26.

```

btnConsultarMascotas.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Cerrar el JFrame actual
        dispose();
        ReadMascotas consultarMascotas = new ReadMascotas(nombre_cliente, documento_cliente);
        consultarMascotas.setVisible(true);
    }
});

btnRegistrarMascota.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Cerrar el JFrame actual
        dispose();
        NewPet registro_mascota = new NewPet(nombre_cliente, documento_cliente);
        registro_mascota.setVisible(true);
    }
});

btnCerrarSesion.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Volver al menú de inicio de sesión
        MainMenu menu = new MainMenu();
        menu.setVisible(true);
        dispose(); // Cierra la ventana actual
    }
});

```

Ilustración 27.

7. El primer botón, “Hacer reserva” se encarga de redirigir al usuario al frame CreateReservation.java, en el que se presenta primero una tabla para manejar cierto orden.

```

// Crear un modelo de tabla
DefaultTableModel model = new DefaultTableModel() {
    @Override
    public boolean isCellEditable(int row, int column) {
        return false; // Hacer que todas las celdas sean no editables
    }
};

model.addColumn("id_reserva");
model.addColumn("fecha_reserva");
model.addColumn("hora_reserva");
model.addColumn("id_tipo_mascota");
model.addColumn("id_cliente");
model.addColumn("id_mascota");

// Crear el JTable con el modelo de tabla
JTable table = new JTable(model);

// Agregar el JTable a un JScrollPane para permitir desplazamiento
JScrollPane scrollPane = new JScrollPane(table);

panel.add(scrollPane, BorderLayout.CENTER);

Conexion databaseConnector = new Conexion();
Connection connection = databaseConnector.getConnection();

```

Ilustración 28.

8. Se presenta también la consulta para mostrar los registros de la tabla de reserva donde aparezca el documento de cliente relacionado:


```
// Realizar la consulta SQL 1
try {
    String query = "SELECT * FROM dba.reserva WHERE documento_cliente = ?";
    PreparedStatement statement = connection.prepareStatement(query);
    statement.setString(1, documento_cliente);
    ResultSet resultSet = statement.executeQuery();

    // Recorrer los registros obtenidos
    while (resultSet.next()) {
        // Obtener los valores de cada columna en el registro actual
        String id_reserva = resultSet.getString("id_reserva");
        String fecha_reserva = resultSet.getString("fecha_reserva");
        String hora_reserva = resultSet.getString("hora_reserva");
        String id_tipo_viaje = resultSet.getString("id_tipo_viaje");
        String id_ruta = resultSet.getString("id_ruta");
        String id_mascota = resultSet.getString("id_mascota");

        // Crear un vector para almacenar los datos de cada fila
        Vector<String> row = new Vector<>();
        row.add(id_reserva);
        row.add(fecha_reserva);
        row.add(hora_reserva);
        row.add(id_tipo_viaje);
        row.add(id_ruta);
        row.add(id_mascota);

        // Agregar la fila al modelo de tabla
        model.addRow(row);
    }

    // Cerrar recursos de la consulta 1
    resultSet.close();
    statement.close();
} catch (SQLException e) {
    e.printStackTrace();
}
```

Ilustración 29.

9. Luego, se crean e inicializan los combobox con cada uno de los valores que se inserten en la base de datos:

```
// Etiquetas
JLabel lblFechaReserva = new JLabel("Fecha reserva (dd/mm/aa)");

try {
    MaskFormatter dateMask = new MaskFormatter("##/##/####");
    dateMask.setPlaceholderCharacter('_');
    txtFechaReserva = new JFormattedTextField(dateMask);
    txtFechaReserva.setColumns(10);
} catch (ParseException e) {
    e.printStackTrace();
}

JLabel lblHoraReserva = new JLabel("Hora reserva (hh:mm formato 24 horas)");

try {
    MaskFormatter timeMask = new MaskFormatter("##:##");
    timeMask.setPlaceholderCharacter('_');
    txtHoraReserva = new JFormattedTextField(timeMask);
    txtHoraReserva.setColumns(5);
} catch (ParseException e) {
    e.printStackTrace();
}

JLabel lblTipoViaje = new JLabel("Tipo de viaje");
String[] tiposViaje = {"1", "2"};
cmbTipoViaje = new JComboBox<>(tiposViaje);

JLabel lblRuta = new JLabel("Tipo de ruta");
String[] tiposRuta = {"1", "2", "3", "4", "5"};
cmbRuta = new JComboBox<>(tiposRuta);

JLabel lblMascota = new JLabel("ID de mascota");
cmbMascota = new JComboBox<>();
```

Ilustración 30.

10. En el segundo query, se seleccón el id_mascota de la tabla Mascota, para insertarlo posteriormente en el combobox. Esto con el fin de que el cliente solo pueda insertar valores que si existan y estén relacionados con él.

```
// Realizar la consulta SQL
try {
    String query = "SELECT id_mascota FROM dbo.Mascota WHERE documento_cliente = ?";
    PreparedStatement statement = connection.prepareStatement(query);
    statement.setString(1, documento_cliente);
    ResultSet resultSet = statement.executeQuery();

    // Recorrer los registros obtenidos
    while (resultSet.next()) {
        // Obtener el valor de id_mascota en el registro actual
        String id_mascota = resultSet.getString("id_mascota");

        // Agregar el valor al JComboBox
        cmbMascota.addItem(id_mascota);
    }

    // Cerrar recursos de la consulta
    resultSet.close();
    statement.close();
} catch (SQLException e) {
    e.printStackTrace();
}

// Agregar el JComboBox y el JLabel al contenedor adecuado (por ejemplo, un JPanel)
panel.add(cmbMascota);
panel.add(lblMascota);
```

Ilustración 31.

```
// Agregar un ActionListener al JComboBox
cmbMascota.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Obtener la selección actual del JComboBox
        String selectedMascota = (String) cmbMascota.getSelectedItem();

        // Actualizar el texto del JLabel con la selección actual
        lblMascota.setText("ID de mascota: " + selectedMascota);
    }
});
```

Ilustración 32.

11. La interfaz presentada con lo anteriormente descrito es:

Tus reservaciones

USUARIO: Sofia ID: 1234567894

id_reserva	fecha_reserva	hora_reserva	id_tipo_viaje	id_ruta	id_mascota

Fecha reserva (mm/dd/aa)

__/__/__

Hora reserva (hh:mm formato 24 horas)

__:__

Tipo de viaje

1

Tipo de ruta

1

ID de mascota

20

Reservar Regresar

Ilustración 33.

12. En el listener del botón de “Reservar”, se inicializan los string de cada columna de la tabla y se convierten a texto, se verifica si algún campo está vacío antes de enviar el formulario y se establece la conexión con la base de datos.

```
btnReservacion.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // Obtener los valores de los campos de texto
        String fecha_reserva = txtFechaReserva.getText();
        String hora_reserva = txtHoraReserva.getText();
        String id_tipo_viaje = cmbTipoViaje.getSelectedItem().toString();
        String id_ruta = cmbRuta.getSelectedItem().toString();
        String id_mascota = cmbMascota.getSelectedItem().toString();

        // Verificar si algún campo está vacío
        if (fecha_reserva.isEmpty() || hora_reserva.isEmpty() || id_tipo_viaje.isEmpty()
            || id_ruta.isEmpty() || id_mascota.isEmpty()) {
            JOptionPane.showMessageDialog(null, "Por favor, complete todos los campos");
            return; // Salir del método sin ejecutar la consulta SQL
        }

        // Crear la conexión a la base de datos
        String conexionUrl = "jdbc:sqlserver://localhost:1433;"
            + "database=AnimalTripOficial;"
            + "user=sa;"
            + "password=hola123;"
            + "loginTimeout=30;";
```

Ilustración 34.

13. En el siguiente bloque de código se prepara la inserción de los datos que el cliente ingresó a la tabla Reserva, añadiéndole a esto, se implementa el mensaje de error de un trigger descrito antes que no permite la inserción de una reserva si la mascota tiene menos de 2 años.

```
try {
    Connection conn = DriverManager.getConnection(conexionURL);

    // Preparar la sentencia SQL con los valores ingresados en el formulario
    String sql = "INSERT INTO dba.Reserva (fecha_reserva, hora_reserva, id_tipo_viaje, documento_cliente, id_casa, id_mascota) VALUES (?, ?, ?, ?, ?, ?)";
    PreparedStatement statement = conn.prepareStatement(sql);
    statement.setString(1, fecha_reserva);
    statement.setString(2, hora_reserva);
    statement.setString(3, id_tipo_viaje);
    statement.setString(4, documento_cliente);
    statement.setString(5, id_casa);
    statement.setString(6, id_mascota);
    // Ejecutar la sentencia SQL
    statement.executeUpdate();

    // Cerrar la conexión
    statement.close();
    conn.close();

    // Mostrar mensaje de éxito solo si la reserva se ha registrado correctamente
    JOptionPane.showMessageDialog(null, "Reserva registrada exitosamente (Sus datos, calge y viaje a internet para mas información de internet)");

    // Limpiar los campos de texto después de validarlos
    txtFechaReserva.setText("");
    txtHoraReserva.setText("");

} catch (SQLException ex) {
    if (ex.getErrorCode() == 10000) {
        // Si el código de error es 10000, significa que se el error generado por el trigger en el trigger
        String mensajeError = ex.getMessage();
        JOptionPane.showMessageDialog(null, mensajeError);
        txtFechaReserva.setText("");
        txtHoraReserva.setText("");
    } else {
        JOptionPane.showMessageDialog(null, "Error al registrar la reserva");
        txtFechaReserva.setText("");
        txtHoraReserva.setText("");
    }
}
```

Ilustración 35.


14. Luego de hacer la reserva, se presenta el mensaje de éxito:

Tus reservaciones

USUARIO: Sofia ID: 1234567894

id_reserva	fecha_reserva	hora_reserva	id_tipo_viaje	id_ruta	id_mascota
1032	2023-09-18	15:30:00.0000000	1	1	20

Mensaje

 Reserva registrada exitosamente (Por favor, salga y vuelva a ingresar para ver reflejada su reserva)

[Aceptar](#)

Fecha reserva (mm/dd/aa)

06/03/2023

Hora reserva (hh:mm formato 24 horas)

13:15

Tipo de viaje

2

Tipo de ruta

1

ID de mascota: 20

20

[Reservar](#)
[Regresar](#)

Activar Wi
Ve a Configurar

Ilustración 36.

15. Finalmente, se verifica que la reserva si se refleje en la interfaz del cliente:

Tus reservaciones

USUARIO: Sofia ID: 1234567894

id_reserva	fecha_reserva	hora_reserva	id_tipo_viaje	id_ruta	id_mascota
1032	2023-09-18	15:30:00.0000000	1	1	20
1033	2023-06-03	13:15:00.0000000	2	1	20

Ilustración 37.

En el siguiente frame DeleteReservation.java se implementa toda la lógica necesaria para poder poner en funcionamiento la cancelación de una reservación. Esto con ayuda del procedimiento almacenado presentado antes para este fin.

1. En principio, se crea de nuevo el modelo de la tabla que aparecerá al momento de ejecutar el frame, recordemos que estas clases necesitan el nombre y documento del cliente logueado por lo cual los pasaremos como parámetros al inicio.

```
// Crear un modelo de tabla
DefaultTableModel model = new DefaultTableModel() {
    @Override
    public boolean isCellEditable(int row, int column) {
        return false; // Hacer que todas las celdas sean no editables
    }
};

model.addColumn("id_reserva");
model.addColumn("fecha_reserva");
model.addColumn("hora_reserva");
model.addColumn("id_tipo_viaje");
model.addColumn("id_ruta");
model.addColumn("id_mascota");

// Crear el JTable con el modelo de tabla
table = new JTable(model);

// Agregar el JTable a un JScrollPane para permitir desplazamiento
JScrollPane scrollPane = new JScrollPane(table);

panel.add(scrollPane, BorderLayout.CENTER);

Conexion databaseConnector = new Conexion();
Connection connection = databaseConnector.getConnection();
```

Ilustración 38.

- Se tiene de igual manera la consulta para poder ver todos los registros de la tabla de reserva:

```
// Realizar la consulta SQL 1
try {
    String query = "SELECT id_reserva, fecha_reserva, hora_reserva, id_tipo_viaje, id_ruta, id_mascota FROM reservas WHERE id_cliente = ?";
    PreparedStatement statement = connection.prepareStatement(query);
    statement.setString(1, documento_cliente);
    ResultSet resultSet = statement.executeQuery();

    // Recorrer los registros obtenidos
    while (resultSet.next()) {
        // Obtener los valores de cada columna en el registro actual
        String id_reserva = resultSet.getString("id_reserva");
        String fecha_reserva = resultSet.getString("fecha_reserva");
        String hora_reserva = resultSet.getString("hora_reserva");
        String id_tipo_viaje = resultSet.getString("id_tipo_viaje");
        String id_ruta = resultSet.getString("id_ruta");
        String id_mascota = resultSet.getString("id_mascota");

        // Crear un vector para almacenar los datos de cada fila
        Vector<String> row = new Vector<>();
        row.add(id_reserva);
        row.add(fecha_reserva);
        row.add(hora_reserva);
        row.add(id_tipo_viaje);
        row.add(id_ruta);
        row.add(id_mascota);

        // Agregar la fila al modelo de tabla
        model.addRow(row);
    }

    // Cerrar recursos de la consulta 1
    resultSet.close();
    statement.close();
} catch (SQLException e) {
    e.printStackTrace();
}
```

Ilustración 39.

- Se hace nuevamente una consulta para obtener el valor de id_reserva y agregarlo al combobox para que el usuario pueda utilizar las reservas que son de él:

```
// Realizar la consulta SQL
try {
    String query = "SELECT id reserva FROM dbo.Reserva WHERE documento_cliente = ?";
    PreparedStatement statement = connection.prepareStatement(query);
    statement.setString(1, documento_cliente);
    ResultSet resultSet = statement.executeQuery();

    // Recorrer los registros obtenidos
    while (resultSet.next()) {
        // Obtener el valor de id mascota en el registro actual
        String id_reserva = resultSet.getString("id_reserva");

        // Agregar el valor al JComboBox
        cmbIDReserva.addItem(id_reserva);
    }

    // Cerrar recursos de la consulta
    resultSet.close();
    statement.close();
} catch (SQLException e) {
    e.printStackTrace();
}
```

Ilustración 40.

- El botón de cancelar reserva tiene la siguiente lógica, se crea la conexión a la base de datos y se implementa la sentencia de la eliminación de la reserva de cliente logueado:

```
@Override
public void actionPerformed(ActionEvent e) {
    // Verificar si se ha seleccionado un registro en el JComboBox
    if (cmbIDReserva.getSelectedIndex() == null) {
        JOptionPane.showMessageDialog(null, "No hay registros seleccionados");
        return;
    }

    String id_reserva = cmbIDReserva.getSelectedIndex().toString();
    // Crear la conexión a la base de datos
    String conexionUrl = "jdbc:mysql://localhost:3306/" +
        " + database=" + dbName +
        " + user=" +
        " + password=" +
        " + loginTimeout=" +

    try {
        Connection conn = DriverManager.getConnection(conexionUrl);


        // Ejecutar la sentencia de eliminación de la reserva
        String deleteSql = "DELETE FROM dbo.Reserva WHERE documento_cliente = ? AND id_reserva = ?";
        PreparedStatement deleteStatement = conn.prepareStatement(deleteSql);
        deleteStatement.setString(1, documento_cliente);
        deleteStatement.setString(2, id_reserva);
        deleteStatement.executeUpdate();
        deleteStatement.close();

        // Ejecutar el trigger después de la eliminación
        String triggerSql = "EXEC actualizar_estado_ruta";
        Statement triggerStatement = conn.createStatement();
        triggerStatement.executeUpdate(triggerSql);
        triggerStatement.close();

        conn.close();
        JOptionPane.showMessageDialog(null, "Reserva cancelada exitosamente");
        cmbIDReserva.setSelectedIndex(0);
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Reserva cancelada exitosamente");
    }
}
```

Ilustración 41.

- Además de esto, se implementa el trigger llamado actualizar_estado_ruta, descrito anteriormente. La ejecución se ve así:

 Tus reservaciones
 —
□
×

USUARIO: Nicolás
ID: 1004563875

id_reserva	fecha_reserva	hora_reserva	id_tipo_viaje	id_ruta	id_mascota
1024	2023-12-20	12:00:00.000...	1	1	2
1034	2023-06-07	12:00:00.000...	2	1	2

ID de la reserva:

1024

Cancelar reserva

Regresar

Ilustración 42.

Como se puede observar, el usuario escoge que reserva quiere borrar, en el caso que tenga más de una. De igual forma, estas son visibles en la tabla superior. Cuando se borra una reserva, aparece un mensaje de éxito y adicionalmente, esta se elimina también de las reservas del cliente, tal como se evidencia a continuación:



Tus reservaciones




USUARIO: Nicolás

ID: 1004563875

id_reserva	fecha_reserva	hora_reserva	id_tipo_viaje	id_ruta	id_mascota
1024	2023-12-20	12:00:00.000...	1	1	2
1034	2023-06-07	12:00:00.000...	2	1	2

Mensaje



Reserva cancelada exitosamente

Aceptar

ID de la reserva:

1024

Cancelar reserva

Regresar

Ilustración 43.

Tus reservaciones

USUARIO: Nicolás

ID: 1004563875

id_reserva	fecha_reserva	hora_reserva	id_tipo_viaje	id_ruta	id_mascota
1034	2023-06-07	12:00:00.000...	2	1	2

ID de la reserva:

1034

Cancelar reserva

Regresar

Ilustración 44.

El siguiente botón del menú del cliente es “Generar recibo” para llevar a cabo su funcionamiento se crea el frame GenerarRecibo.java. En el cual se contiene un modelo de tabla para mostrar la información adscrita al cliente en la tabla Factura, luego de hacer una reserva.

```

Connection connection = databaseConnector.getConnection();

// Realizar la consulta SQL
try {
    String query = "SELECT * FROM dbo.Factura WHERE documento_cliente = ?";
    PreparedStatement statement = connection.prepareStatement(query);
    statement.setString(1, documento_cliente);
    ResultSet resultSet = statement.executeQuery();

    // Recorrer los registros obtenidos
    while (resultSet.next()) {
        // Obtener los valores de cada columna en el registro actual
        String id_factura = resultSet.getString("id_factura");
        String fecha_factura = resultSet.getString("fecha_factura");
        String descripcion_servicio = resultSet.getString("descripcion_servicio");
        String precio_total = resultSet.getString("precio_total");
        String id_reserva = resultSet.getString("id_reserva");

        // Crear un vector para almacenar los datos de cada fila
        Vector<String> row = new Vector<>();
        row.add(id_factura);
        row.add(fecha_factura);
        row.add(descripcion_servicio);
        row.add(precio_total);
        row.add(id_reserva);

        // Agregar la fila al modelo de tabla
        model.addRow(row);
    }

    // Cerrar recursos
    resultSet.close();
    statement.close();
} catch (SQLException e) {
    e.printStackTrace();
}

```

Ilustración 45.

1. Se selecciona el id_reserva para así poder crear la factura con los datos del cliente y añadirle el id_reserva además de ingresar los valores en el combobox para que el usuario pueda elegir a que reserva quiere generarle una factura.

```

// Realizar la consulta SQL
try {
    String query = "SELECT id_reserva FROM dbo.Reserva WHERE documento_cliente = ?";
    PreparedStatement statement = connection.prepareStatement(query);
    statement.setString(1, documento_cliente);
    ResultSet resultSet = statement.executeQuery();

    // Recorrer los registros obtenidos
    while (resultSet.next()) {
        // Obtener el valor de id_mascota en el registro actual
        String id_reserva = resultSet.getString("id_reserva");

        // Agregar el valor al JComboBox
        cmbIDReserva.addItem(id_reserva);
    }

    // Cerrar recursos de la consulta
    resultSet.close();
    statement.close();
} catch (SQLException e) {
    e.printStackTrace();
}

```

Ilustración 46.

2. En el botón de pagar se implementa la conexión con la base de datos normalmente y empieza con las verificaciones, primero se verifica si ya existe una factura para la reserva indicada y luego se insertan los valores asignados en el registro.

```

@Override
public void actionPerformed(ActionEvent e) {
    // Verificar si se ha seleccionado un registro en el JTable
    if (tblReserva.getSelectedRow() == null) {
        JOptionPane.showMessageDialog(this, "No has seleccionado un registro.");
        return;
    }

    String id_reserva = tblReserva.getValueAt(selectedRow, 0).toString();

    // Generar los valores para el nuevo registro de factura
    String fecha_factura = "2023-06-07";
    String descripcion_servicio = "Servicio de reserva";
    String precio_total = "10000";

    // Crear la conexión a la base de datos
    String connectionString = "jdbc:mysql://localhost:3306/uniminuto?useSSL=false";
    Properties props = new Properties();
    props.put("user", "root");
    props.put("password", "root");

    try {
        Connection connection = DriverManager.getConnection(connectionString, props);
        // Verificar si se seleccionó una factura para el id de reserva
        String selectQuery = "SELECT * FROM db_facturas WHERE id_reserva = ?";
        PreparedStatement preparedStatement = connection.prepareStatement(selectQuery);
        preparedStatement.setString(1, id_reserva);
        ResultSet resultSet = preparedStatement.executeQuery();
        resultSet.next();
        int monto = resultSet.getInt(1);
        resultSet.close();
        preparedStatement.close();

        // Insertar en la tabla de facturas
        String insertQuery = "INSERT INTO facturas (id_reserva, fecha_factura, descripcion_servicio, precio_total) VALUES (?, ?, ?, ?)";
        PreparedStatement insertStatement = connection.prepareStatement(insertQuery);
        insertStatement.setString(1, id_reserva);
        insertStatement.setString(2, fecha_factura);
        insertStatement.setString(3, descripcion_servicio);
        insertStatement.setString(4, precio_total);
        insertStatement.executeUpdate();

        // Mostrar el mensaje de éxito
        JOptionPane.showMessageDialog(this, "Factura generada exitosamente.");
    } catch (SQLException ex) {
        JOptionPane.showMessageDialog(this, "Error al generar la factura: " + ex.getMessage());
    }
}


```

Ilustración 47.

- Luego, se implementa el procedimiento almacenado `sp_generar_recibo`, el cual se encarga de enviar en un mensaje la factura generada con el id del cliente y demás datos relevantes. Finalmente, la interfaz es:

Tus recibos

USUARIO: Nicolás ID: 1004563875

id_factura	fecha_factura	descripcion_serv...	precio_total	id_reserva
<div> <div>Recibo de reserva</div> <div>  <div> <div>Recibo de reserva</div> <div>=====</div> <div> <div>Cliente: Nicolás</div> <div>Mascota: Hachiko</div> <div>Documento cliente: 1004563875</div> <div>Ruta: Bogotá - Medellín</div> <div>Fecha de reserva: 2023-06-07</div> </div> <div>Aceptar</div> </div> </div> </div>				

ID de la reserva:

1034

Pagar Regresar

Ilustración 48.

Tus recibos

USUARIO: Nicolás ID: 1004563875

id_factura	fecha_factura	descripcion_serv...	precio_total	id_reserva
16	2023-05-29	Viaje de mascota	50000.0000	1034

ID de la reserva:

1034

Pagar Regresar

Ilustración 49.

Luego, el próximo botón “Consultar Mascotas” es una forma que tiene el cliente de visualizar sus mascotas, las que tienen relacionado su documento.

1. Se crea el frame ReadMascotas.java que se encarga de mostrar las mascotas que tiene el cliente. Se seleccionan todos los registros de la tabla mascota donde el documento del cliente sea el que está logueado y luego se obtienen los valores y se agregan los datos:

```
// Realizar la consulta SQL
try {
    String query = "SELECT * FROM dbc Mascota WHERE documento_cliente = ?";
    PreparedStatement statement = connection.prepareStatement(query);
    statement.setString(1, documento_cliente);
    ResultSet resultSet = statement.executeQuery();

    // Recorrer los registros obtenidos
    while (resultSet.next()) {
        // Obtener los valores de cada columna en el registro actual
        String id_mascota = resultSet.getString("id_mascota");
        String nombre_mascota = resultSet.getString("nombre_mascota");
        String edad_mascota = resultSet.getString("edad_mascota");
        String sexo_mascota = resultSet.getString("sexo_mascota");
        String peso_mascota = resultSet.getString("peso_mascota");
        String carnet_vacuna = resultSet.getString("carnet_vacuna");
        String id_raza = resultSet.getString("id_raza");
        String id_tipo_mascota = resultSet.getString("id_tipo_mascota");


        // Crear un vector para almacenar los datos de cada fila
        Vector<String> row = new Vector<>();
        row.add(id_mascota);
        row.add(nombre_mascota);
        row.add(edad_mascota);
        row.add(sexo_mascota);
        row.add(peso_mascota);
        row.add(carnet_vacuna);
        row.add(id_raza);
        row.add(id_tipo_mascota);

        // Agregar la fila al modelo de tabla
        model.addRow(row);
    }

    // Cerrar recursos
    resultSet.close();
    statement.close();
    connection.close();
}
```

Ilustración 50.

2. El funcionamiento de dicho frame es así:

 Tus mascotas

☐ ☐ ☐

USUARIO: Nicolás
ID: 1004563875

id_mascota	nombre_mascota	edad_mascota	sexo_mascota	peso_mascota	carnet_vacuna	id_raza	id_tipo_mascota
2	Hachiko	3	F	32.0	S	4	1

Regresar

Ilustración 51.

Para concluir con esta sección de operaciones que puede realizar el cliente, se crea un último frame para que el cliente registre una nueva mascota.

1. El frame de registro de las mascotas NewPet.java, permite al usuario registrar la mascota si está logueado. Además, tanto en esta clase como en las anteriores y posteriores, se generaron si era necesario filtros de caracteres para no permitir el ingreso de ciertos caracteres no permitidos, en este caso filtros para el nombre, la edad, y el peso.


```
// Filtro de nombres solo para letras - nombre
PlainDocument docNombre = new PlainDocument();
docNombre.setDocumentFilter(new DocumentFilter() {
    @Override
    public void insertString(DocumentFilter.FilterBypass fb, int offset, String string, AttributeSet attr)
        throws BadLocationException {
        fb.insertString(offset, string.replaceAll("[^\\p{L}]", ""), attr); // Remover caracteres no alfabéticos
    }

    @Override
    public void replace(DocumentFilter.FilterBypass fb, int offset, int length, String text, AttributeSet attr)
        throws BadLocationException {
        fb.replace(offset, length, text.replaceAll("[^\\p{L}]", ""), attr); // Remover caracteres no alfabéticos
    }
});
txtNombreMascota.setDocument(docNombre);

// Filtro de documentos solo para números - edad
PlainDocument docNumeros = new PlainDocument();
docNumeros.setDocumentFilter(new DocumentFilter() {
    @Override
    public void insertString(DocumentFilter.FilterBypass fb, int offset, String string, AttributeSet attr)
        throws BadLocationException {
        fb.insertString(offset, string.replaceAll("[^\\p{D}]", ""), attr); // Remover caracteres no numéricos
    }

    @Override
    public void replace(DocumentFilter.FilterBypass fb, int offset, int length, String text, AttributeSet attr)
        throws BadLocationException {
        fb.replace(offset, length, text.replaceAll("[^\\p{D}]", ""), attr); // Remover caracteres no numéricos
    }
});
txtEdadMascota.setDocument(docNumeros);
```

Ilustración 52.

2. A través del listener del botón de registrar se obtuvieron los valores de los campos de texto, valida que ninguno puede estar vacío al enviar el formulario, se crea la conexión a la base de datos y el query de insert para registrar los datos correctamente.

```
btnRegistrar.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // Obtener los valores de los campos de texto
        String nombre_mascota = txtNombreMascota.getText();
        String edad_mascota = txtEdadMascota.getText();
        String sexo_mascota = cmbSexoMascota.getSelectedItem().toString();
        String peso_mascota = txtPesoMascota.getText();
        String carnet_vacuna = cmbCarnetVacuna.getSelectedItem().toString();
        String id_raza = cmbRaza.getSelectedItem().toString();
        String id_tipo_mascota = cmbTipoMascota.getSelectedItem().toString();

        // Verificar si alguno campo está vacío
        if (nombre_mascota.isEmpty() || edad_mascota.isEmpty() || sexo_mascota.isEmpty() ||
            peso_mascota.isEmpty() || carnet_vacuna.isEmpty() || id_raza.isEmpty() ||
            id_tipo_mascota.isEmpty()) {
            JOptionPane.showMessageDialog(null, "Por favor, complete todos los campos");
            return; // Salir del método sin ejecutar la consulta SQL
        }

        // Crear la conexión a la base de datos
        String conexionUrl = "jdbc:sqlserver://localhost:1433;"
            + "database=AnimalTripOficial;"
            + "user=sa;"
            + "password=hola123;"
            + "loginTimeout=30:";
```

Ilustración 53.

```
try {
    Connection conn = DriverManager.getConnection(url);
    // Preparar la sentencia SQL con los valores ingresados del formulario
    String sql = "INSERT INTO dbc_mascotas (nombre_mascota, edad_mascota, sexo_mascota, peso_mascota, esta_vacunado, id_raza, id_tipo_mascota) VALUES (?, ?, ?, ?, ?, ?, ?)";
    PreparedStatement statement = conn.prepareStatement(sql);
    statement.setString(1, nombre_mascota);
    statement.setInt(2, edad_mascota);
    statement.setString(3, sexo_mascota);
    statement.setString(4, peso_mascota);
    statement.setString(5, esta_vacunado);
    statement.setInt(6, id_raza);
    statement.setInt(7, id_tipo_mascota);
    // Ejecutar la sentencia SQL
    statement.executeUpdate();

    // Mostrar la pantalla y mostrar un mensaje de éxito
    statement.close();
    conn.close();
    JOptionPane.showMessageDialog(null, "Mascota registrada exitosamente");

    // Limpiar los campos de texto después de registrarse
    txtNombreMascota.setText("");
    txtEdadMascota.setText("");
    txtPesoMascota.setText("");
    txtEstaVacunado.setText("");

} catch (SQLException e) {
    JOptionPane.showMessageDialog(null, "Error al registrar la mascota");
}
```

Ilustración 54.

Por tanto, esta última opción del cliente se ve así:

Ilustración 55.

Tal como se expresó antes, el segundo apartado del desarrollo de la aplicación está dirigido a los empleados. Para esto se ubicó un botón de inicio de sesión para estos en el menú principal (ver ilustración 22). En primer lugar, actúa el frame Login_Employee.

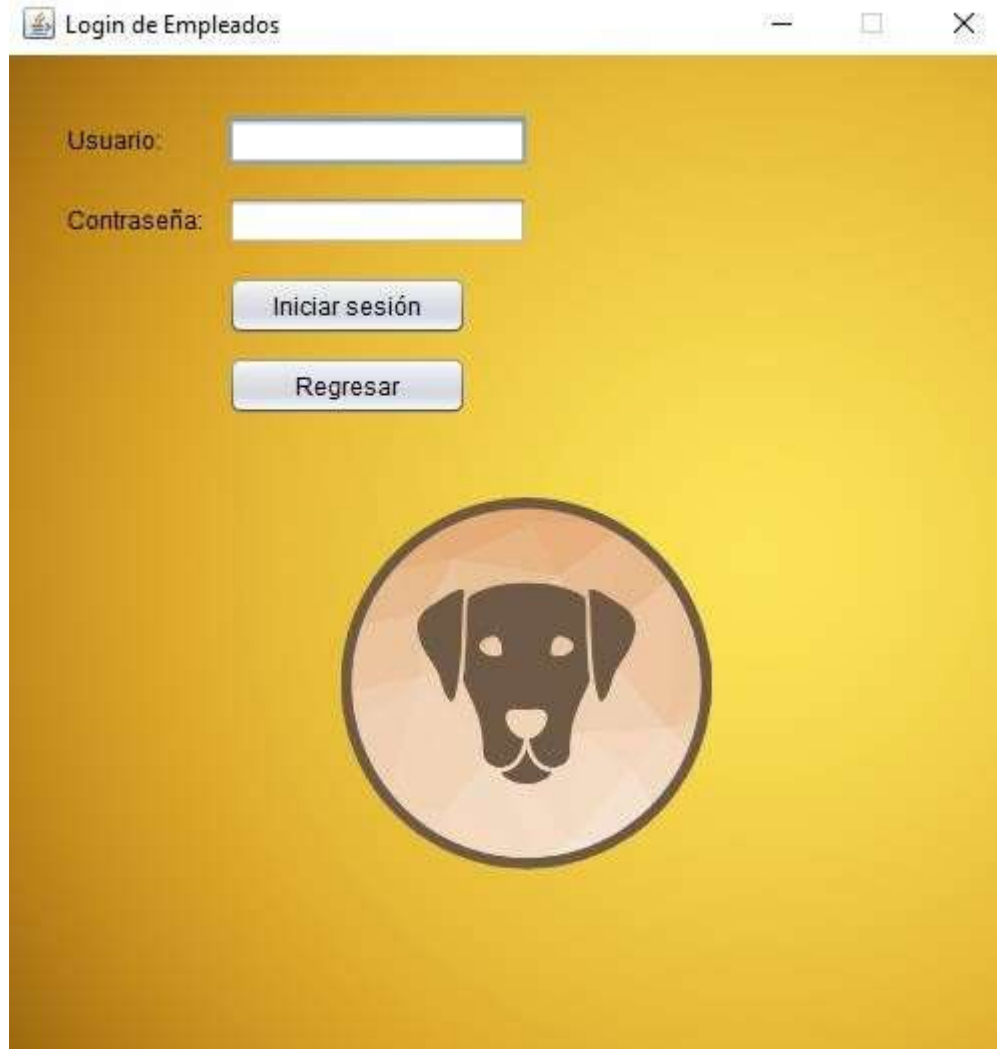


Ilustración 56.

1. A través del listener del botón “Iniciar sesión” se validan las credenciales del usuario llamando a la clase Login_Validator, anteriormente descrita.

```

btnLogin.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        Login_Employee.username = txtUsername.getText();
        String password = new String(txtPassword.getPassword());
        Conexion databaseConnector = new Conexion();

        // Crear una instancia de Login_Validator
        Login_Validator validator = new Login_Validator(databaseConnector);
        boolean isValidCredentials = false;
        try {
            isValidCredentials = validator.validateCredentials(username, password);
        } catch (SQLException ex) {
            ex.printStackTrace();
            // Aquí puedes manejar el error de la manera que desees, por ejemplo, mostrando un mensaje de error específico
            JOptionPane.showMessageDialog(null, "Error en la validación de las credenciales");
        }

        if (isValidCredentials) {
            // Credenciales válidas, mostrar mensaje de éxito y navegar a otra página
            //JOptionPane.showMessageDialog(null, "Inicio de sesión exitoso");
            Main_Employee mainEmployeeFrame = new Main_Employee(username);

            // Hacer visible el frame Main_Employee
            mainEmployeeFrame.setVisible(true);

            // Cerrar el frame actual (Login frame)
            dispose();
        } else {
            // Credenciales inválidas, mostrar mensaje de error
            JOptionPane.showMessageDialog(null, "Credenciales inválidas");
        }
    }
});

```

Ilustración 57.

2. Luego de que las credenciales son validadas como correctas, entra en acción el siguiente frame `Main_Employee.java`. En este frame, se presentan las tablas que puede manipular el usuario de acuerdo con el rol y los permisos que tiene dentro de la base de datos. Cabe resaltar que estos permisos y roles ya fueron descritos (ver apartado *Usuarios y Logins*). Primero, se crean los botones para cada tabla:

```

JButton btnConductor = new JButton("Conductor");
btnConductor.setBounds(20, y, 120, 30);

JButton btnProfesional = new JButton("Profesional");
btnProfesional.setBounds(20, y + 40, 120, 30);

JButton btnReserva = new JButton("Reserva");
btnReserva.setBounds(20, y, 120, 30);

JButton btnFactura = new JButton("Factura");
btnFactura.setBounds(20, y + 40, 120, 30);

JButton btnDestino = new JButton("Destino");
btnDestino.setBounds(20, y + 80, 120, 30);

JButton btnReservaDestino = new JButton("Reserva_destino");
btnReservaDestino.setBounds(20, y + 120, 120, 30);

JButton btnCliente = new JButton("Cliente");
btnCliente.setBounds(20, y + 120, 120, 30);

JButton btnTipo_documento = new JButton("Tipo_documento");
btnTipo_documento.setBounds(20, y + 120, 120, 30);

JButton btnMascota = new JButton("Mascota");
btnMascota.setBounds(20, y + 120, 120, 30);

JButton btnLimitacion = new JButton("Limitación");
btnLimitacion.setBounds(20, y + 120, 120, 30);

```

Ilustración 58.

```

JButton btnCliente = new JButton("Cliente");
btnCliente.setBounds(20, y + 120, 120, 30);

JButton btnTipo_documento = new JButton("Tipo_documento");
btnTipo_documento.setBounds(20, y + 120, 120, 30);

JButton btnMascota = new JButton("Mascota");
btnMascota.setBounds(20, y + 120, 120, 30);

JButton btnLimitacion = new JButton("Limitación");
btnLimitacion.setBounds(20, y + 120, 120, 30);

JButton btnRaza = new JButton("Raza");
btnRaza.setBounds(20, y + 120, 120, 30);

JButton btnTipo_mascota = new JButton("Tipo_mascota");
btnTipo_mascota.setBounds(20, y + 120, 120, 30);

JButton btnReserva_mascota = new JButton("Reserva_mascota");
btnReserva_mascota.setBounds(20, y + 120, 120, 30);

JButton btnMascota_limitacionn = new JButton("Mascota_limitacionn");
btnMascota_limitacionn.setBounds(20, y + 120, 120, 30);

JButton btnRuta = new JButton("Ruta");
btnRuta.setBounds(20, y + 120, 120, 30);

JButton btnTipo_viaje = new JButton("Tipo_viaje");
btnTipo_viaje.setBounds(20, y + 120, 120, 30);

```

Ilustración 59.

- Posteriormente, se construye el listener para cada uno de los botones:

```

btnCliente.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Main_Employees_Cliente cliente= new Main_Employees_Cliente();
        cliente.setVisible(true);
        dispose();
    }
});

btnMascota.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Main_Employees_Mascota mascota= new Main_Employees_Mascota();
        mascota.setVisible(true);
        dispose();
    }
});

btnConductor.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Main_Employees_Conductor conductor= new Main_Employees_Conductor();
        conductor.setVisible(true);
        dispose();
    }
});

btnProfesional.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Main_Employees_Profesional profesional= new Main_Employees_Profesional();
        profesional.setVisible(true);
    }
});

```

Ilustración 60.

```

btnRuta.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Main_Employees_Ruta ruta= new Main_Employees_Ruta();
        ruta.setVisible(true);
        dispose();
    }
});

btnDestino.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Main_Employees_Destino destino= new Main_Employees_Destino();
        destino.setVisible(true);
        dispose();
    }
});

btnRaza.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Main_Employees_Raza raza= new Main_Employees_Raza();
        raza.setVisible(true);
        dispose();
    }
});

btnLimitacion.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Main_Employees_Limitacion limitacion= new Main_Employees_Limitacion();
        limitacion.setVisible(true);
    }
});

```

Ilustración 61.

4. A continuación, se muestra la interfaz para cada uno de los usuarios, teniendo en cuenta que cada una de ellas es diferente.
 - Usuario: Andrés Sánchez.

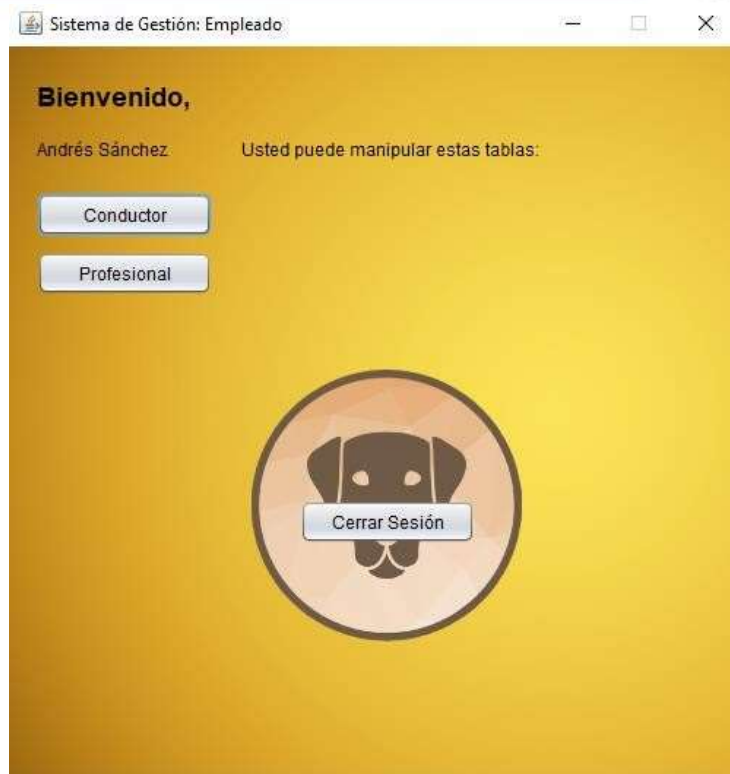


Ilustración 62.

- Usuario: Laura Hernández.

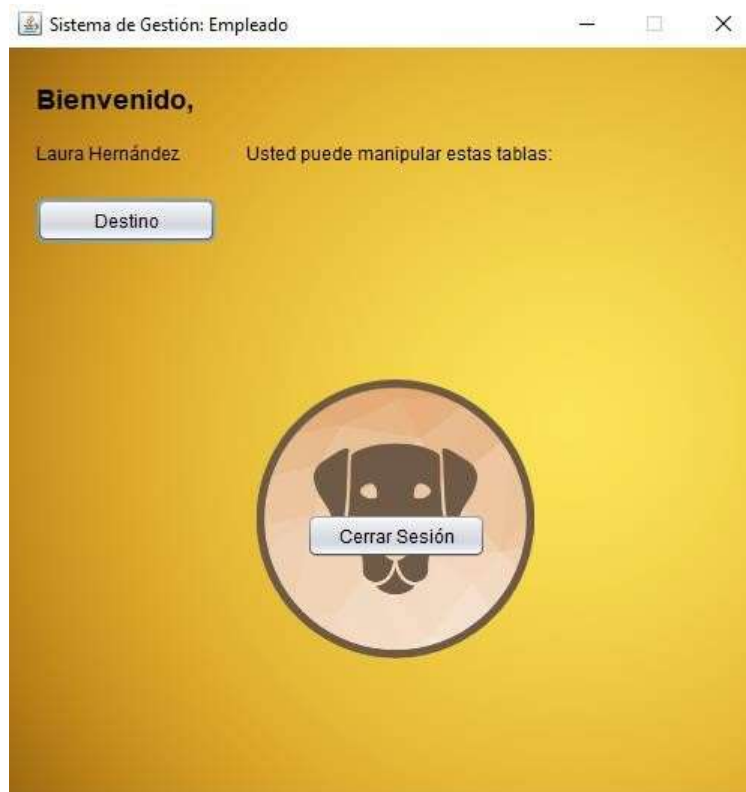


Ilustración 63.

- Usuario: Javier Rojas.

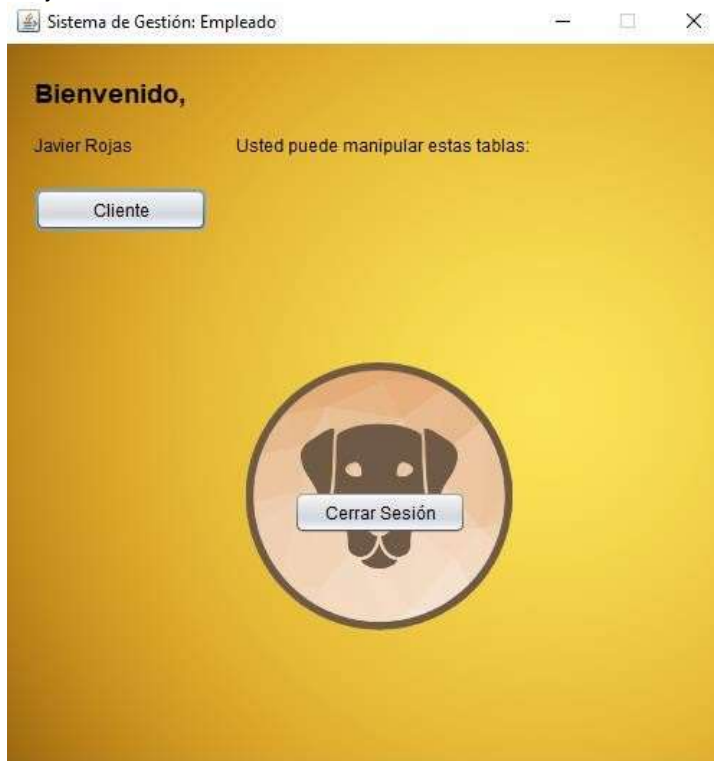


Ilustración 64.

- Usuario: María Torres.

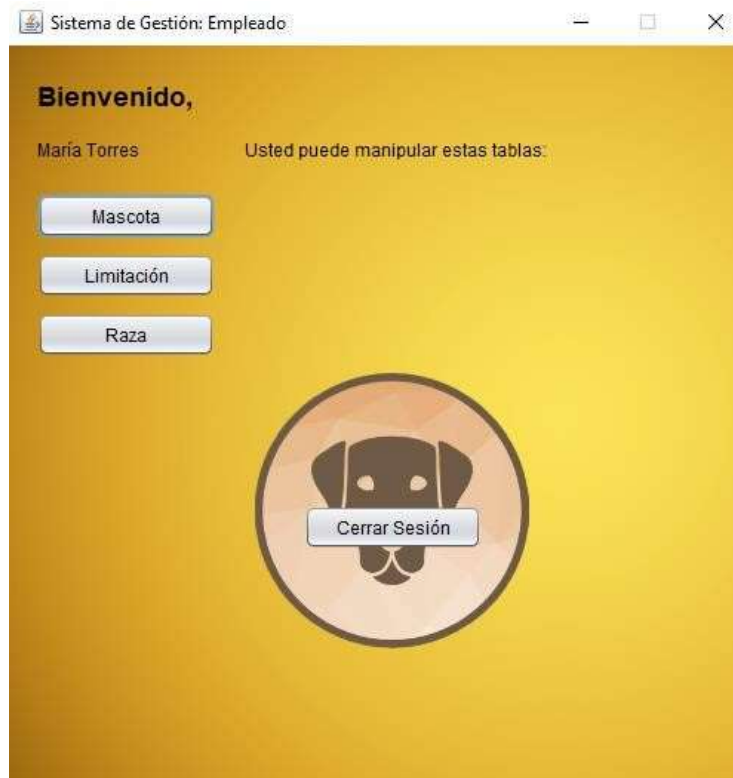


Ilustración 65.

- Usuario: Jhonathan Palacios.

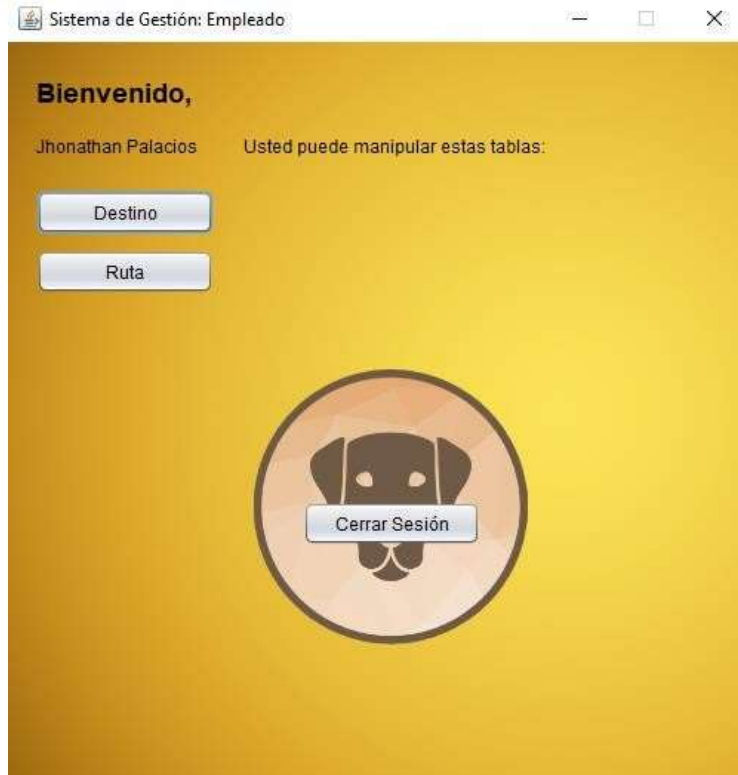


Ilustración 66.

- Usuario: Héctor Cárdenas (administrador).



Ilustración 67.

5. Como se demostró cada usuario puede acceder sólo a ciertas tablas, excepto Héctor Cárdenas quien es el administrador y puede acceder a todas. Tomando como ejemplo a este último, cada botón se convierte en un frame:

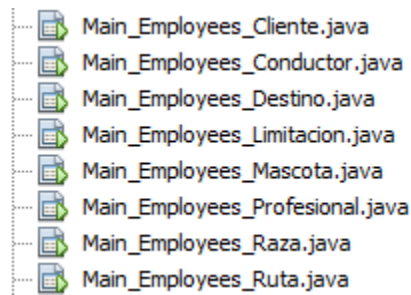


Ilustración 68.

6. Lo anterior, con el fin de que al dar clic en el botón un listener se active y se muestren las opciones del CRUD de acuerdo con la tabla, por ejemplo, en la tabla Mascota pasaría algo así:



Ilustración 69.

7. Entonces, dentro del frame de `Main_Employees_Mascota.java` se hace lo propio para realizar las acciones del CRUD, todo mediante listeners. Para registrar a una mascota:

Ilustración 70.

```
btnRegistrar.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // Obtener los valores de los campos de texto
        String nombre_mascota = txtNombreMascota.getText();
        String edad_mascota = txtEdadMascota.getText();
        String sexo_mascota = cmbSexoMascota.getSelectedItem().toString();
        String peso_mascota = txtPesoMascota.getText();
        String carnet_vacuna = cmbCarnetVacuna.getSelectedItem().toString();
        String id_raza = cmbRaza.getSelectedItem().toString();
        String id_tipo_mascota = cmbTipoMascota.getSelectedItem().toString();
        String documento_cliente = txtDocumento.getText();

        // Verificar si algún campo está vacío
        if (nombre_mascota.isEmpty() || edad_mascota.isEmpty() || sexo_mascota.isEmpty()
            || peso_mascota.isEmpty() || carnet_vacuna.isEmpty() || id_raza.isEmpty()
            || id_tipo_mascota.isEmpty() || documento_cliente.isEmpty()) {
            JOptionPane.showMessageDialog(null, "Por favor, complete todos los campos");
            return; // Salir del método sin ejecutar la consulta SQL
        }
    }
});
```

Ilustración 71.

8. La consulta SQL en la que se envían los datos de los campos que llenó el usuario para posteriormente hacer la inserción en la tabla Mascota:

```
try {
    Connection conn = DriverManager.getConnection(conexionUrl);

    // Preparar la sentencia SQL con los valores capturados del formulario
    String sql = "INSERT INTO dbo.Mascota (nombre_mascota, edad_mascota, sexo_mascota, peso_mascota, carnet_vacuna, id_raza, documento_cliente, id_tipo_mascota) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
    PreparedStatement statement = conn.prepareStatement(sql);
    statement.setString(1, nombre_mascota);
    statement.setString(2, edad_mascota);
    statement.setString(3, sexo_mascota);
    statement.setString(4, peso_mascota);
    statement.setString(5, carnet_vacuna);
    statement.setString(6, id_raza);
    statement.setString(7, documento_cliente);
    statement.setString(8, id_tipo_mascota);
    // Ejecutar la sentencia SQL
    statement.executeUpdate();

    // Cerrar la conexión y mostrar un mensaje de éxito
    statement.close();
    conn.close();
    JOptionPane.showMessageDialog(null, "Mascota registrada exitosamente");

    // Limpiar los campos de texto después de registrar
    txtNombreMascota.setText("");
    txtEdadMascota.setText("");
    txtPesoMascota.setText("");
    txtDocumento.setText("");
} catch (SQLException ex) {
    JOptionPane.showMessageDialog(null, "Error al registrar la mascota");
}
```

Ilustración 72.

9. Luego de realizar el registro, se muestra un mensaje de éxito:



Ilustración 73.

- El siguiente botón “Ver mascotas”, relacionado con la selección de todos los registros de la tabla, simplemente, mediante la consulta de SQL los selecciona y los muestra en una tabla, así:

CRUD Mascota

Id_ma...	Nombre	Edad	Sexo	Peso	¿Vacu...	Raza	Dueño	Tipo d...
1	Toby	5	M	45.2	S	1	10036...	1
2	Hachiko	3	F	32.0	S	4	10045...	1
3	Michi	2	F	18.4	S	15	10132...	2
4	Milo	3	M	13.0	S	12	10234...	2
5	Rambo	7	M	34.9	S	7	10244...	1
7	Romeo	2	F	10.3	S	16	10256...	2
8	Arwen	1	M	35.2	S	10	10327...	1
9	Jerry	6	F	12.0	S	15	10425...	2
10	Joe	2	F	11.5	S	13	10458...	2
20	España	4	F	12.0	N	3	12345...	1
1016	Lucas	2	M	17.0	S	10	10567...	1
1017	Lupe	5	F	27.9	S	5	27492...	1
1018	Tomy	1	M	54.3	S	7	17399...	1
1021	Juan	1	M	54.3	S	7	10244...	1
1023	Andres	5	F	54.3	S	7	18473...	1
1025	Lulu	2	F	12.0	S	9	10234...	1
1026	Topo	5	M	32.0	S	6	21112...	1

Regresar

Ilustración 74.

- Para hacer esta acción se implementa en el listener del botón antes mencionado lo siguiente:

```
btnVer.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        panel.removeAll();
        panel.revalidate();
        panel.repaint();
        modeloTabla = new DefaultTableModel();
        tablaMascotas = new JTable(modeloTabla);
        JScrollPane scrollPane = new JScrollPane(tablaMascotas);
        getContentPane().add(scrollPane);
        getContentPane().add(panel);
        modeloTabla.addColumn("Id_mascota");
        modeloTabla.addColumn("Nombre");
        modeloTabla.addColumn("Edad");
        modeloTabla.addColumn("Sexo");
        modeloTabla.addColumn("Peso");
        modeloTabla.addColumn("¿Vacunado?");
        modeloTabla.addColumn("Raza");
        modeloTabla.addColumn("Dueño");
        modeloTabla.addColumn("Tipo de mascota");
        btnRegresar2 = new JButton("Regresar");
        panel.setLayout(new BorderLayout());
        panel.add(scrollPane, BorderLayout.CENTER);
        panel.add(btnRegresar2, BorderLayout.SOUTH);
    }
});
```

Ilustración 75.

```
try {
    Connection conn = DriverManager.getConnection(conexionUrl);
    String sql = "SELECT * FROM dbo.Mascota";
    Statement statement = conn.createStatement();
    ResultSet resultSet = statement.executeQuery(sql);
    // Borra los registros existentes en la tabla
    modeloTabla.setRowCount(0);
    // Agrega los registros de la base de datos a la tabla
    while (resultSet.next()) {
        String id_mascota = resultSet.getString("id_mascota");
        String nombre_mascota = resultSet.getString("nombre_mascota");
        String edad_mascota = resultSet.getString("edad_mascota");
        String sexo_mascota = resultSet.getString("sexo_mascota");
        String peso_mascota = resultSet.getString("peso_mascota");
        String carnet_vacuna = resultSet.getString("carnet_vacuna");
        String id_raza = resultSet.getString("id_raza");
        String documento_cliente = resultSet.getString("documento_cliente");
        String id_tipo_mascota = resultSet.getString("id_tipo_mascota");

        Object[] fila = {id_mascota, nombre_mascota, edad_mascota, sexo_mascota, peso_mascota, carnet_vacuna, id_raza, documento_cliente, id_tipo_mascota};
        modeloTabla.addRow(fila);
    }

    statement.close();
    conn.close();
} catch (SQLException ex) {
    ex.printStackTrace();
    JOptionPane.showMessageDialog(null, "Error al obtener los registros de la mascota");
}
```

Ilustración 76.

- Para el siguiente botón “Actualizar mascotas”, se pide al usuario escoger en un combobox la columna que quiere actualizar, la actualización y el id de la mascota, esto para saber que registro actualizar.

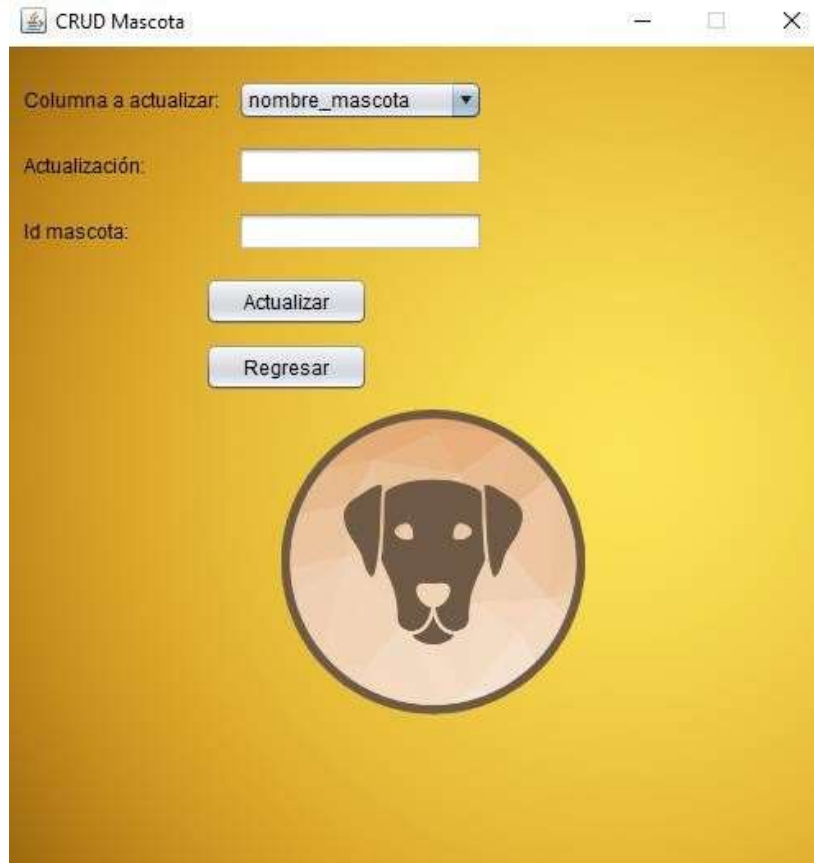


Ilustración 77.

6. Luego, en el listener del botón “Actualizar” se reciben los datos y se ejecuta la consulta correspondiente:

```
btnActualizar1.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        String id_mascota = txtId_mascota.getText();  
        String columnaSeleccionada = (String) comboBoxColumnas.getSelectedItem();  
        String actualizacion = txtActualizacion.getText();  
    }  
});
```

Ilustración 78.

```
try {
    Connection conn = DriverManager.getConnection(conexionUrl);
    String sql = "UPDATE dbo.Mascota SET " + columnaSeleccionada + " = ? WHERE id_mascota = ?";

    PreparedStatement statement = conn.prepareStatement(sql);
    statement.setString(1, actualizacion);
    statement.setString(2, id_mascota);

    // Ejecutar la sentencia SQL
    int filasActualizadas = statement.executeUpdate();

    statement.close();
    conn.close();

    if (filasActualizadas > 0) {
        JOptionPane.showMessageDialog(null, "Mascota actualizada exitosamente");
    } else {
        JOptionPane.showMessageDialog(null, "No se encontró ninguna mascota con el id especificado");
    }

    // Limpiar los campos de texto después de actualizar
    txtActualizacion.setText("");
    txtId_mascota.setText("");
} catch (SQLException ex) {
    ex.printStackTrace();
    JOptionPane.showMessageDialog(null, "Error al actualizar la mascota");
}
```

Ilustración 79.

7. Luego de accionar el botón de actualizar, se muestra un mensaje de éxito:

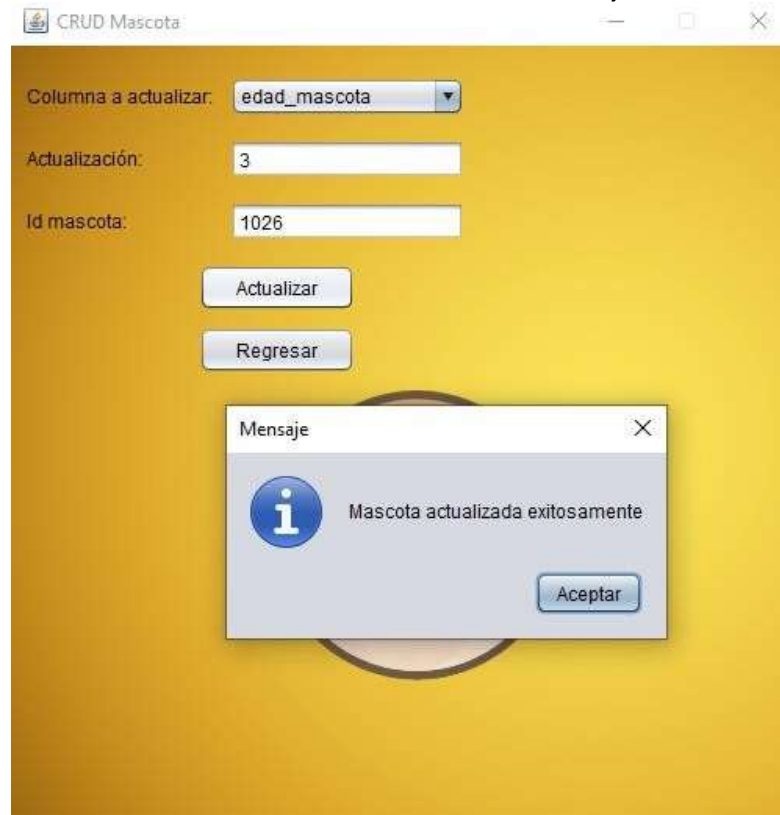


Ilustración 80.

8. Finalmente, el botón "Eliminar mascota" integra dentro de su lógica la consulta de eliminación del registro de acuerdo con el id de la mascota, el cual se le pide al usuario:

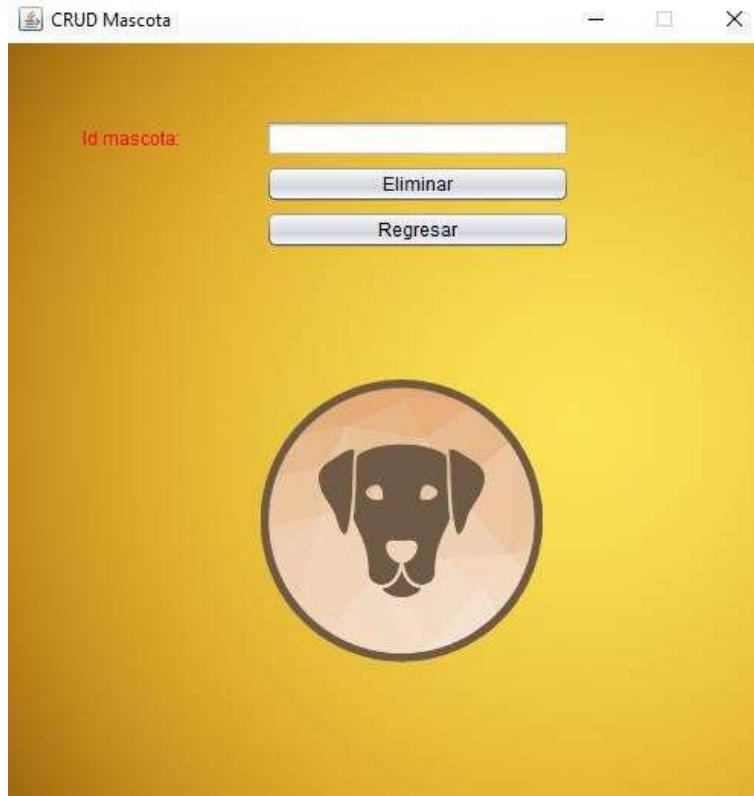


Ilustración 81.

9. Entonces, dentro del listener del botón, según el número de id de mascota escrito, ese registro se eliminará de la tabla.

```
btnEliminar1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String id_mascota = txtId_mascotal.getText();
```

Ilustración 82.

```
try {
    Connection conn = DriverManager.getConnection(conexionUrl);

    // Preparar la sentencia SQL con el valor del documento a eliminar
    String sql = "DELETE FROM dbo.Mascota WHERE id_mascota = ?";
    PreparedStatement statement = conn.prepareStatement(sql);
    statement.setString(1, id_mascota);

    // Ejecutar la sentencia SQL
    int rowsAffected = statement.executeUpdate();

    // Verificar si se eliminó correctamente el registro
    if (rowsAffected > 0) {
        JOptionPane.showMessageDialog(null, "Mascota eliminada exitosamente");
    } else {
        JOptionPane.showMessageDialog(null, "No se encontró ninguna mascota con el id especificado");
    }

    statement.close();
    conn.close();
} catch (SQLException ex) {
    ex.printStackTrace();
    JOptionPane.showMessageDialog(null, "Error al eliminar la mascota");
}
}
```

Ilustración 83.

10. Por último, se muestra el mensaje de éxito al eliminar un registro:



Ilustración 84.

Las acciones del CRUD están presentes en cada una de las tablas, con la misma estructura que se presentó antes.

Por consiguiente, el último botón presente en el menú de inicio es "Crea tu cuenta". La funcionalidad de este botón es que, si un usuario no está registrado, pueda hacerlo y luego acceda a su sesión para que realice las acciones que se mostraron anteriormente.

1. Para este fin se crea un nuevo frame `NewClient.java`, en el que se presenta un formulario de registro, con campos como documento de identificación, nombre, etc.

Ilustración 85.

2. En el botón de registrar actúa un listener en el que se hace la inserción a la tabla Cliente:

```
btnRegistrar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Obtener los valores de los campos de texto
        String documento_cliente = txtDocumento.getText();
        String nombre_cliente = txtNombre.getText();
        String direccion_cliente = txtDireccion.getText();
        String telefono_cliente = txtTelefono.getText();
        String tipo_documento = cmbTipoDocumento.getSelectedItem().toString();
        String contraseña_cliente = new String(txtContraseña.getPassword()); // Obtener la contraseña como texto

        // Verificar si algún campo está vacío
        if (documento_cliente.isEmpty() || nombre_cliente.isEmpty() || direccion_cliente.isEmpty()
            || telefono_cliente.isEmpty() || contraseña_cliente.isEmpty()) {
            JOptionPane.showMessageDialog(null, "Por favor, complete todos los campos");
            return; // Salir del método sin ejecutar la consulta SQL
        }
    }
});
```

Ilustración 86.

```
try {
    Connection conn = DriverManager.getConnection(conexionUrl);

    // Preparar la sentencia SQL con los valores capturados del formulario
    String sql = "INSERT INTO dbo.Cliente (documento_cliente, nombre_cliente, direccion_cliente, telefono_cliente, tipo_documento, contraseña_cliente) VALUES (?, ?, ?, ?, ?, ?)";
    PreparedStatement statement = conn.prepareStatement(sql);
    statement.setString(1, documento_cliente);
    statement.setString(2, nombre_cliente);
    statement.setString(3, direccion_cliente);
    statement.setString(4, telefono_cliente);
    statement.setString(5, tipo_documento);
    statement.setString(6, contraseña_cliente);
    // Ejecutar la sentencia SQL
    statement.executeUpdate();
    // Cerrar la conexión y mostrar un mensaje de éxito
    statement.close();
    conn.close();
    JOptionPane.showMessageDialog(null, "Cliente registrado exitosamente");
    // Limpiar los campos de texto después de registrar
    txtDocumento.setText("");
    txtNombre.setText("");
    txtDireccion.setText("");
    txtTelefono.setText("");
    txtContraseña.setText("");
} catch (SQLException ex) {
    ex.printStackTrace();
    JOptionPane.showMessageDialog(null, "Error al registrar el cliente");
}
```

Ilustración 87.

En última instancia, se presenta el botón para cerrar la aplicación. Como se pudo evidenciar esta aplicación cuenta con diferentes operaciones dirigidas tanto a clientes como a empleados, cuenta con consultas SQL, validación de campos de texto, validación de credenciales y es intuitiva, así como amigable con el usuario final.