Actividad tipo prueba 3: Proyecto de desarrollo de CRUD - MySQL

Datos de alumnos

Nombre alumno			
	Apellido Paterno	Apellido Materno	Nombres
RUT			
Puntaje máximo		**NOTA**	
Puntaje obtenido			

INSTRUCCIONES

Caso – Mantenedor Clientes y Mascotas

Actividades

Para el desarrollo de esta actividad deberás leer atenta y comprensivamente cada uno de los ítems a desarrollar.

- Desarrollar una solución para las problemáticas planteadas utilizando **Python**.

Actividad: Realización de CRUD

La veterinaria Patitas amadas, requiere un control de sus clientes y sus respectivas mascotas es por ello que le ha solicitado realizar un sistema para poder llevar un registro de lo anterior.

Inicialmente se cargará el siguiente menú, siempre y cuando se ***haya autenticado como recepcionista, mediante su correo y contraseña****.*

CRUD FICHA MASCOTAS

1. *CRUD Recepcionistas*
2. ***CRUD Clientes***
3. ***CRUD Mascotas***
4. *CRUD Ficha*
5. ***Salir del sistema***

Nota: La opción 1 y 4, no se encuentran operativas por el momento.

Al ingresar la opción 2:

CRUD Cliente

1. *Ingresar Cliente*
2. *Modificar Cliente*
3. *Eliminar Cliente*
4. *Mostrar todos los Clientes*
5. *Volver al menú principal*

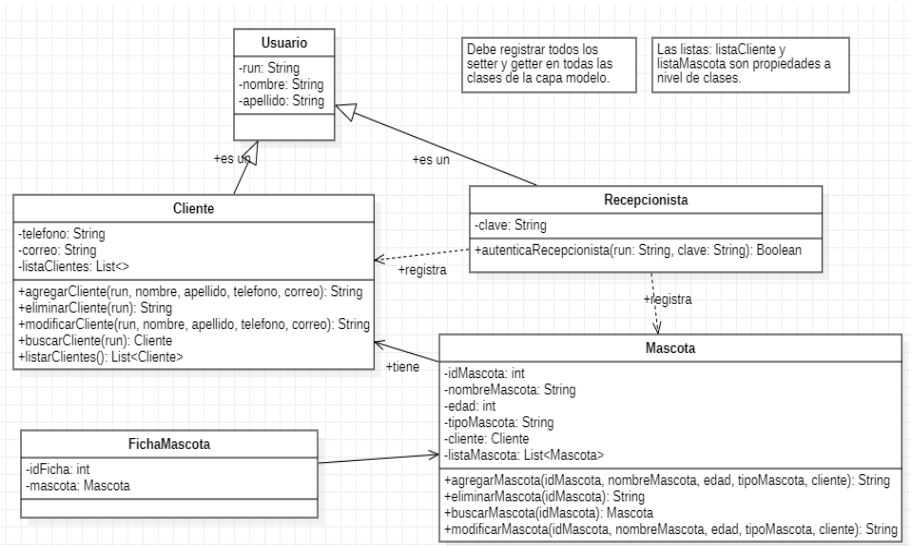
Al ingresar la opción 3:

CRUD Mascotas

1. *Ingresar Mascota*
2. *Modificar Mascota*
3. *Eliminar Mascota*
4. *Volver al menú principal*

- Se deberá construir patrón MVC (DAO-DTO):
  Vista:
  - **Main.py (Vista)**: El que debe contener el proceso de login de acceso al sistema.
  - Propiedades de las clases que se encuentran en la capa modelo, deben ser privadas.
  - Métodos: Puede agregar algunos métodos adicionales a los que se encuentran en el diagrama de clases, que necesite para construir su aplicación. Además, puede utilizar diccionarios para mejorar sus procesos.

Diagrama de Clases:



Métodos para trabajar con la Clase Cliente:

- **agregarCliente():** Este método debe permitir ingresar los datos de un cliente y almacenarlos en la tabla Cliente de la base de datos. Antes de realizar el ingreso, se debe validar que el cliente **NO** se encuentre (utilizar el método buscarCliente() para la búsqueda). De existir, se debe enviar mensaje respectivo y no permitir el ingreso.
    - **Los datos para ingresar por consola son los que indica el diagrama de clases.**

  El cliente, también se debe incorporar al atributo de clase listaClientes de la clase Cliente.

- **modificarCliente():** Este método debe permitir modificar solo las columnas que el usuario haya ingresado algún dato. Antes de realizar la modificación, se debe validar que el cliente se encuentre (utilizar el método buscarCliente() para la búsqueda). De **NO** existir, se debe enviar mensaje respectivo y no permitir la modificación.

  Observaciones:

  - Un dato vacío, implica que el dato en la tabla no se modifica.
  - El cliente también se debe modificar en el atributo de clase listaClientes de la clase Cliente.

- **eliminarCliente():** Este método debe permitir eliminar un cliente de la tabla cliente de la base de datos, según run de cliente. Además, se debe realizar una pregunta de confirmación de eliminación, antes de proceder con la baja del cliente. Antes de realizar la eliminación, se debe validar que el cargo se encuentra (utilizar el método buscarCliente() para la búsqueda). De **NO** existir, se debe enviar mensaje respectivo y no realizar eliminación.

  El cliente también se debe eliminar del atributo de clase listaClientes de la clase Cliente.

- **buscarCliente():** Este método debe permitir buscar el cliente por su run ingresado por el usuario.

- **listarClientes():** Este método debe permitir mostrar los datos de todos los clientes ingresados en la tabla clientes.

Métodos para trabajar con la Clase Mascota:

- **agregarMascota():** Este método debe permitir ingresar los datos de una mascota en la tabla Mascota de la base de datos. Antes de realizar el ingreso, se debe validar que la mascota **NO** se encuentre (utilizar el método buscarMascota() para la búsqueda). De existir, se debe enviar mensaje respectivo y no permitir el ingreso.

  Observaciones:

  - La mascota, también se debe incorporar al atributo de clase listaMascota de la clase Mascota.
  - Es muy importante que al grabar la mascota en la tabla, deben dejar el idtipo de la mascota y no la descripción.

- **modificarMascota():** Este método debe permitir modificar las columnas de la tabla mascotas de la base de datos. Antes de realizar la modificación, se debe validar que la mascota se encuentre (utilizar el método buscarMascota() para la búsqueda). De **NO** existir, se debe enviar mensaje respectivo y no permitir la modificación.

  Observación:

  - Un dato vacío, implica que el dato en la tabla no se modifica.
  - La mascota, también se debe modificar en el atributo de clase listaMascota de la clase Mascota.

- **eliminarMascota():** Este método debe permitir eliminar una mascota de la tabla mascotas de la base de datos, según identificador de mascota. Además, se debe realizar una pregunta de confirmación de eliminación, antes de proceder con la baja de esa mascota. Antes de realizar la eliminación, se debe validar que la mascota se encuentra (utilizar el

método buscarMascota() para la búsqueda). De **NO** existir, se debe enviar mensaje respectivo y no realizar eliminación.

La mascota, también se debe eliminar del atributo de clase listaMascota de la clase Mascota.

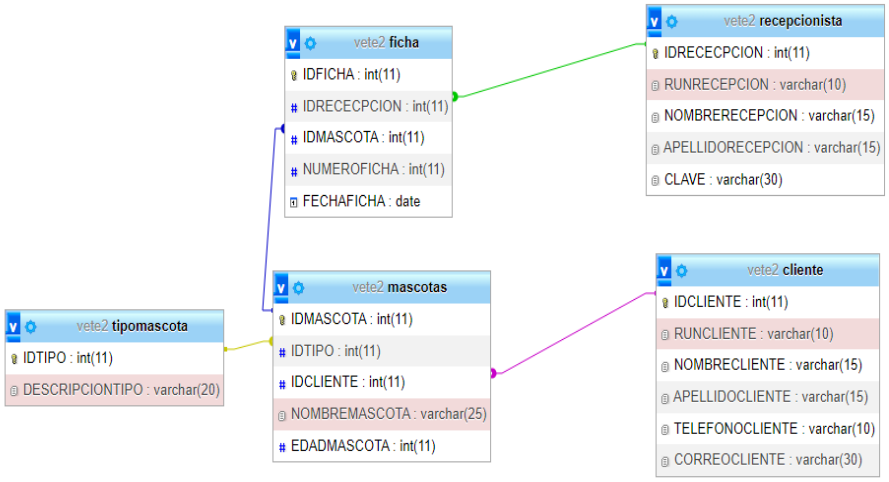
- **buscarMascota():** Este método debe permitir buscar la mascota por su identificador de mascota ingresado por el usuario. Los datos de la mascota, se deben desplegar de existir.

Construya la base de datos actividad y las tablas indicadas, utilizando el SGBD **MySQL**.

Recuerde:

- **Validar que los campos de entrada no sean vacíos en el método agregarCliente() y agregarMascota().**
- **Las pk de las tablas recepcionista y cliente deben ser auto incrementables.**

Modelo físico:



Script

```
/*==============================================================*/
/* Table: CLIENTE                                               */
/*==============================================================*/
create table CLIENTE
(
  IDCLIENTE          int not null auto_increment,
  RUNCLIENTE         varchar(10),
  NOMBRECLIENTE      varchar(15),
  APELLIDOCLIENTE    varchar(15),
  TELEFONOCLIENTE    varchar(10),
  CORREOCLIENTE      varchar(30),
  primary key (IDCLIENTE)
);

/*==============================================================*/
/* Table: FICHA                                                 */
/*==============================================================*/
create table FICHA
(
  IDFICHA            int not null,
  IDRECECPCION       int not null,
  IDMASCOTA          int not null,
  NUMEROFICHA        int,
  FECHAFICHA         date,
  primary key (IDFICHA)
);

/*==============================================================*/
/* Table: MASCOTAS                                              */
/*==============================================================*/
create table MASCOTAS
(
  IDMASCOTA          int not null,
  IDTIPO             int,
  IDCLIENTE          int not null,
  NOMBREMASCOTA      varchar(25),
  EDADMASCOTA        int,
  primary key (IDMASCOTA)
);

/*==============================================================*/
```

```
/* Table: RECEPCIONISTA                          */
/*==============================================================*/
create table RECEPCIONISTA
(
  IDRECECPCION       int not null auto_increment,
  RUNRECEPCION       varchar(10),
  NOMBRERECEPCION    varchar(15),
  APELLIDORECEPCION  varchar(15),
  CLAVE              varchar(32),
  primary key (IDRECECPCION)
);

/*==============================================================*/
/* Table: TIPOMASCOTA                            */
/*==============================================================*/
create table TIPOMASCOTA
(
  IDTIPO             int not null,
  DESCRIPCIONTIPO    varchar(20),
  primary key (IDTIPO)
);

alter table FICHA add constraint FK_GESTIONA foreign key (IDRECECPCION)
    references RECEPCIONISTA (IDRECECPCION) on delete restrict on update restrict;

alter table FICHA add constraint FK_TIENE foreign key (IDMASCOTA)
    references MASCOTAS (IDMASCOTA) on delete restrict on update restrict;

alter table MASCOTAS add constraint FK_ES_UN foreign key (IDTIPO)
    references TIPOMASCOTA (IDTIPO) on delete restrict on update restrict;

alter table MASCOTAS add constraint FK_TIENEN foreign key (IDCLIENTE)
    references CLIENTE (IDCLIENTE) on delete restrict on update restrict;
```

Pasos para insertar registros de manera directa en la base de datos:

Paso 1: Insertar registro, en recepcionista.

```
INSERT INTO recepcionista VALUES(NULL, '11111111-
1','ANA','GROVE','6b27261d2376675ed03f852cd30474d3');
```

Comentado [CPGG1]: 1234a#

Paso 2: Revisar el recepcionista creado y ver si id, que se generó de forma automática.

```
SELECT * FROM recepcionista WHERE runrecepcion= '11111111-1';
```

Paso 3: En la tabla tipomascota, se deben crear las siguientes tuplas:

1 ---> Perro

2 ---> Gato

Pd. Por el momento solo se atienden perros y gatos.

Instrumento de evaluación

Para verificar lo que han aprendido, luego de desarrollar esta evaluación, revise las respuestas o desarrollo a través de la siguiente pauta de cotejo.

Criterio de Evaluación	Indicadores	Escala de valoración		Observaciones
		SI	NO	
2.1.1 Desarrolla aplicación POO, considerando datos de ingreso para ser almacenados en base de datos. 2.1.2 Codifica Acceso a base de datos mediante Database API. 2.1.3 Desarrolla aplicación POO con procesos CRUD, considerando seguridad de los datos. 2.1.4 Codifica Login de acceso a la aplicación desarrollada, considerando hash MD5 para datos sensibles. 2.1.5 Contribuyendo en la organización de las tareas.	Funcional: genera el proceso de autenticación de los datos ingresados con la base de datos por medio del método autenticaRecepcionista().	4	1	
	Funcional: Realiza el proceso de incorporar un cliente a la tabla respectiva, utilizando el método agregarCliente().	4	1	
	Funcional: Realiza el proceso de incorporar una mascota la tabla respectiva, utilizando el método agregarMascota().	4	1	
	Funcional: Realiza el proceso de modificación de un cliente en la tabla respectiva, utilizando el método modificarCliente().	4	1	
	Funcional: Realiza el proceso de modificación de una mascota en la tabla respectiva, utilizando el método modificarMacota().	4	1	
	Funcional: Realiza el proceso de eliminación de un cliente de la tabla respectiva, utilizando el método eliminarCliente().	4	1	
	Funcional: Realiza el proceso de eliminación de una mascota de la tabla respectiva, utilizando el método eliminarMascota().	4	1	
	Funcional: Realiza el	4	1	

	proceso de buscar el cliente, utilizando el método buscarCliente().			
	Funcional: Realiza el proceso de buscar mascota, utilizando el método buscarMascota().	4	1	
	Funcional: Realiza el proceso de mostrar todos los clientes que se encuentran en la tabla respectiva, utilizando el método listarClientes().	4	1	
	Funcional: Permite eliminar y modificar los datos en las listas de cliente y mascota cuando corresponda, según lo solicitado.	6	1	
	Realiza las validaciones respectivas en los datos de entrada, en la lógica de negocios y construye los respectivos menús.	6	1	
	Los métodos que se encuentran en los módulos Dto del package Controlador, construyen los objetos respectivos para ser enviados a la lógica de negocios y al módulo Dao correspondiente.	6	1	
	Los métodos que se encuentran en los módulos del package Dao, se comunican con la base de datos de forma correcta.	6	1	
	Las clases que se encuentran en los módulos del package del Modelo, son utilizadas correctamente por los módulos Dto del package Controlador.	6	1	
	Carga las listas (listaCliente, listaMascota) para comuna y cargo antes de la carga del menú principal	4	1	
	Maneja excepciones, evitando caídas de sistema.	4	1	
	Permite encriptar la clave al momento de autenticarse, utilizando Hash MD5	4	1	
	Puntaje total	82		