Laboratorio de Programación y Lenguajes 2022

Trabajo Práctico Obligatorio

Lenguaje de programación C

Facultad de Ingeniería
Universidad Nacional de la Patagonia San Juan Bosco

Especificación de los trabajos finales de lenguajes de programación unificados por el uso de una base de datos PostgreSQL.

Los trabajos finales de lenguajes utilizan una base relacional PostgreSQL, de esta forma se deberá interactuar de una forma ordenada y organizada para lograr la ejecución correcta de las consultas, ya que el motor procesa en forma de consultas las interacciones que se programen para dar solución a lo que indique en los requerimientos del sistema.

Sistema de Registro de información de una veterinaria

Se requiere crear una aplicación para tener el registro de la información de los clientes, profesionales, mascotas, tratamientos y medicamentos aplicados a las mascotas de una comercio ***"Buena Pata"*** veterinaria, la información del cliente consta de datos personales, dni, apellido, nombres, domicilio, teléfono, localidad donde vive. De la mascota se conoce los datos de especie, nombre, si posee vacunación, se debe registrar las consultas que se le realizan, de allí los tratamientos indicados y si hubieren medicamentos para los tratamientos, también se debe tener registrado.

Funcionalidad necesaria del sistema(aspectos generales).

A nivel funcional el sistema deberá realizar registro de todas las entidades indicadas, actualización de información.

Se va a requerir que cada ingreso de información o interacción con el usuario sea validado, por ejemplo no ingresar nombres de localidades sin texto, clientes, mascotas, profesionales con datos completos, si se debe leer un dato numérico o fecha se deben validar, .. etc.

El registro de la consulta se debe registrar previamente con anticipación, luego si el cliente asiste con su mascota se debe marcar que asistió, primero debe validar que exista la persona a la cual se le va a dar el turno de consulta, caso contrario se debe registrar en el sistema, se debe ingresar una fecha/hora válidos, va a implicar validar que disponga de lugar en la fecha/Hora ingresados, no debe existir turno de consulta ya registrado para la misma persona.

El sistema deberá proveer una serie de listados:

- de los turnos de consultas que se tendran en un rango de fecha desde/hasta
- listado de mascotas
    - por especie
    - por cliente, ya que un cliente puede tener más de una mascota.
- listado de profesionales

Desarrollo Lenguaje C

En el trabajo final de lenguaje C, deberán aplicarse los conceptos vistos en la primera parte de la materia, con el plus de contar con un ORM para la conexión e interacción con una base de datos PostGreSQL .

Se provee la implementación base de un ORM para usar en lenguaje C, en un proyecto de referencia.

Objetivos

Comprender el funcionamiento y uso del ORM en base a la interface que provee, poder modificar y adaptar según los requerimientos del sistema.

Requerimientos

- Completar el modelo, según los ejemplos de referencia (modelo, relaciones, configuraciones).
- Desarrollar pantallas con validaciones para ingresar y actualizar información de las siguientes entidades:

    - Clientes
    - Mascotas
    - Profesionales
    - Tratamientos
    - Turnos de Consultas
    - Medicamentos

- Gestionar en el sistema para producir listados y resúmenes estadísticos de la carga de información
    - Para el caso de los listado las pantallas de gestión debe permitir obtener opciones de ordenamiento y sentido. Por ejemplo para un cliente puede ser por Apellido o Nro de Documento
- Posibilitar la exportación a archivos la información del sistema, desde los listados(el usuario deberá indicar el nombre del archivo). Considerar las opciones de ordenamiento.

- Listados requeridos, en principio:
  - Listar todos los clientes.
  - Listar todos los profesionales.
  - Listar todas las mascotas.
  - Listar los turnos de consultas que se dieron entre un periodo de tiempo dado(indicar si asistió si se lista información histórica).
  - Listar todos los tratamientos de una mascota.
  - Listar los medicamentos asignados a una mascota.

Ejemplo de pantalla para agregar nueva Localidad

[Menú de opciones]

[1 - Ingresos]
[2 - Actualizaciones]
[3 - Listados]
[4 - Estadísticas]
[5 - Salir]

Opción : 1

[Menú de ingreso de información]

[1 - Cliente]
[2 - Mascotas]
…

Opción: 1

[Ingreso de Cliente]

1) Ingrese DNI : …
2) Ingrese Apellido(90 Caracteres): ..
- … Varios pasos mas…..

N) Resultado: (Cliente registrado correctamente o cartel de Error si existe el dni ya registrado y volver al menú anterior.

En las actualizaciones los únicos valores que ***no se podrán modificar son las claves primarias***, por ejemplo en el caso de la modificación de una mascota se puede modificar el nombre, fecha de nacimiento, observaciones, si esta vacunado.

Por otro lado en el ingreso o modificación de una información relacionada, por ejemplo la localidad de un cliente, se debe poder elegir de un listado obtenido desde la base. O por ejemplo al asignar un tratamiento a una consulta, se debe poder elegir el tratamiento desde listado de todos los tratamientos disponibles en el sistema.

El Modelo

Considerando el modelo que utiliza pseudo-Objetos, se utilizará en el proyecto de referencia este concepto un poco más ampliado ya que se va a simular una *interface* común para todos estos elementos útiles para la implementación de la capa ORM.

A continuación se detalla el conjunto de entidades:

- **localidad**
  - cod_postal
  - nombre

- **mascotas**
  - codigo
  - nombre
  - fecha_nac
  - dni_cliente
  - vacunado
  - observaciones

- **profesionales**
  - dni
  - apellido
  - nombres
  - telefono
  - domicilio
  - cod_postal -- relacion con la Localidad
  - matricula
  - observaciones

- **especie**
  - codigo
  - nombre

- **tratamientos**
  - codigo
  - descripcion
  - cod_especie

- **mascota_medicamento**
  - codigo
  - cod_tratamiento
  - cod_medicamento
  - cantidad

- **clientes**
  - dni
  - apellido
  - nombres
  - fecha_nac
  - domicilio
  - cod_postal -- relacion con la Localidad
  - observaciones

- **diagnostico**
  - codigo
  - nombre

- **consulta**
  - codigo
  - fecha
  - hora
  - dni_prof
  - cod_mascota
  - asistio -- S/N
  - cod_diag
  - observaciones

- **medicamentos**
  - codigo
  - descripcion
  - cod_especie
  - importe
  - cantidad

- **mascota_tratamientos**
  - codigo
  - cod_consulta
  - cod_tratamiento
  - fecha_desde
  - decha_hasta

Implementación de un ORM en C

El concepto de ORM se puede definir como una construcción de software que permite la gestión de entidades para una manejo más de alto nivel para la interacción con una base de datos relacional, es decir se relaciona con la persistencia y la posibilidad de una vista a nivel de objetos, en este caso Pseudo-Objetos.

Al contar con una base de datos relacional, se debe gestionar esa transformación de tipo tablas a una representación en modelo de pseudo-objetos.

Ayuda para la implementación – Librería orm.c

Tenemos el Pseudo-Objeto ***Object*** que implementa versiones genéricas de una interface que involucra a ***findbykey***, ***findAll*** y ***saveObj***.

Cabe recordar que el término Pseudo-objeto implica una estructura con punteros a función que se programan de tal forma que provea una serie de comportamiento encapsulado y reutilice código genérico para reducir la cantidad de código que es común a todas las entidades que componen al modelo del sistema.
Las funciones básicas que posee cada pseudo-objeto son:

- int (*findAll)(void *self, void **list,char *criteria);
- int (*findbykey)(void *self,…)
- bool (*saveObj)(void *self, …)
- void (*toString)(void *self) --implementación depende de que info posee.

Por ejemplo para recorrer listado de todas las localidades del sistema y mostrar la información, sería así:

```
obj_Localidad *loc;
void *list,*itm;
int i,size=0;
loc = Localidad_new(); // usar el contructor
size = loc->findAll(loc,&list,NULL); // se invoca sin criterio - listar todos...
  for(i=0;i<size;++i)
  {
      itm = ((Object **)list)[i];
      ((Object *)itm)->toString(itm);
  }
  destroyObjList(list,size);  // liberar listado, cada instancia creada en el listado
  destroyObj(loc); // liberar Recurso
```

Aclaración del criterio: Se utilizan las columnas disponibles en la base de datos.

El método **findbykey**, devuelve **1 si encontró según clave** o **-1 si no encontró.** Si obtuvo información desde la base de datos para una determinada clase, completa los datos de las propiedades de la instancia que invoca al método, el campo clave se configura de acuerdo al tipo de clave que tiene la clase.

Por ejemplo para buscar el la localidad por id (Codigo postal):

```
obj_Localidad *loc;
loc = Localidad_new();
// Si se encontro en la base. Valor (NOT_FOUND = -1)
if(loc->findbykey(loc,9000) != NOT_FOUND)
{
   printf("%s", loc->toString(loc)); // mostrar los datos de la entidad
}
destroyObj(loc); // liberar Recurso
```

El método **saveObj**, permite realizar el ingreso de nueva instancia o actualización de una instancia previamente recuperada mediante **findbykey**. Devuelve true(1) si lo pudo ejecutar bien false(0) sino

Por ejemplo para buscar el objeto dado su id para luego actualizarlo:

```
obj_Cliente *cli;
cli = Cliente_new();
/*
Buscar y luego actualizar
...
*/
if( cli->findbykey(cli,,22456952) != NOT_FOUND)
{
  cli->setTelefono(cli,"2804411050");
  ….  // acceso a los atributos por los getters y setters
if  ( cli->saveObj(cli) == true )
{
   printf("Actualizacion de paciente realizada correctamente!\n");
}
destroyObj(cli); // liberar Recurso
```

Proyecto de referencia.

Se provee un proyecto de referencia con la estructuración de los archivos fuentes que representan al ORM, y código de ejemplo para probar la conexión con la base, la configuración de las librerías en el proyecto también son importantes para la compilación del sistema.

El motor de base de datos es PostgreSQL, versión **9.1, 9.2, 9.3 y 9.4 Versión 32 bits** funcionan correctamente en la interacción.

Configuración del proyecto DevCPP

Configuracion proyecto DevCpp

ubicar el path del posgreSQL
por ejemplo
"C:\Program Files (x86)\PostgreSQL\9.1"
Se usa para poner acceso al archivo "libpq.lib"
En Proyecto-> opciones de proyecto
solapa "Parameters"

"Linker"
click en "Add Library".. y ubicar "libpq.lib" tiene que estar en carpeta "lib" en instalación de posgreSQL

Luego ir a Solapa
"Directories"
en solapa interna "Include Directories"
click en el icono que muestra un árbol de exploración(abajo)
buscar el path en carpeta de instalación de PostgreSQL, carpeta "Include"
click en "Add"

y click en "Ok"

Aclaracion: Si abren el "tpfinalc.dev" seguro esta configurado con mi version de prueba con PostgreSQl 9.1 en ese caso quiten los directorios no válidos. use según
su versión de postgresql
Hasta donde se anduvo bien hasta la versión 9.4 en otras no encontraba bien las referencias para compilar el proyecto

Otra Aclaración (en Windows):

para ejecutar tienen que tener en el mismo directorio
-- sacados desde carpeta bin de postgreSQL, si no funciona los que van en el proyecto de referencia busquen en su instalación
libeay32.dll
libiconv-2.dll (Este según la instalación puede tener otro nombre pero siempre aparece "iconv" en el nombre)
libintl-8.dll
libpq.dll
ssleay32.dll
zlib1.dll

Conexión con la base de datos

El primer argumento de la ejecución del sistema indicará el path de un archivo ini donde encontrar los parametros de conexion con la base de datos, es requerido para su conexión correcta.

En el main se deberá tener la llamada

```
if(!init_config(argv[POS_CONFIG]))
        exit(-1); // error en lectura del archivo ini
```

Estructura del contenido del archivo ini

```
[config]
server=localhost
database=buena_pata
port=5432
user=postgres
pwd=[Clave de mi motor.]
```

Funcionalidad Utils.c

Se cuenta con diversas funciones ya programadas que no deberán escribir, asi reutilizan lo disponible en el proyecto.

Por ejemplo hay:

```
// funciones sobre Cadena
char* rtrim(char* string, char junk);
//Elimina espacios a derecha
char** fStrSplit(char *str, const char *delimiters);
//Particiona Cadena según tokens
char* fStrJoin(char **str, const char *delimiters, int sz_opt);
//Une Cadena según Tokens

// funciones para información de Fecha
char* getFechaHora();
// obtiene fecha y hora actual del sistema
char* getFecha();
//obtiene fecha actual
char* getDiaFecha(char *fecha);
// obtiene Dia de la semana según la fecha o la actual si el parámetro es NULL
// por ejemplo: da Martes
printf("%s\n",getDiaFecha("2021-04-13"));
```

función ***getLastError()*** Permite obtener un detalle de la última ejecución sobre la base de datos. por ejemplo

al guardar información de un pseudo-Objeto se puede ver que error fue el que se devolvió.

```
if(!loc->saveObj(loc))  {
        printf("Ocurrió un error al agregar Localidad:\n%s\n",getLastError());
  }
```

función ***clearError()*** limpia el cartel de error de variable del sistema