Laboratorio de Programación y Lenguajes 2024

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 gestión

La secretaría de Turismo de la Provincia quiere administrar los datos de las operaciones turísticas de las distintas agencias que están inscriptas como proveedoras de servicios turísticos en todo el territorio de la provincia. Para ello los directivos de la misma han participado de un relevamiento con nuestra consultora.

Del relevamiento se desprende que existen distintas agencias que ofrecen paquetes turísticos para observar, fotografiar y en algunos casos hasta interactuar con algunos animales marinos y/o terrestres como así también visitar y recorrer diferentes lugares naturales con una variada y atractiva fauna y vegetación.

Cada agencia se encuentra radicada en distintas localidades y posee un código identificatorio, nombre, dirección(Calle,nro/piso/dpto) y al menos 2 teléfonos.

Se encargan de ofrecer distintos paquetes turísticos que incluyen avistajes, ya sea de ballenas, delfines, pingüinos, lobos marinos, elefantes marinos, guanacos, avestruces, etc. También caminatas (trekking) en distintos parajes con variado nivel de dificultad y duración. Los costos de los paquetes se conforman del importe de las actividades que componen al mismo.

Dentro de la información del paquete se puede especificar el nivel o dificultad que a su vez incluye uno o varias actividades vinculadas a ese nivel indicado.

Las actividades pueden contar con un transporte o no, según sea el caso el importe del transporte se suma al valor de la actividad.

Los datos de los transportes son la identificación o dominio, y una descripción que registra el tipo de transporte, capacidad y detalles.

Cada agencia ofrece un conjunto de paquetes turísticos predeterminados.

Por último cada agencia permite la creación de un paquete a medida dependiendo de las actividades que puede proveer, dando lugar a una adaptación única por turista que lo arma a su medida.

Luego de tener determinado el paquete que el turista desea contratar, se genera para dicho paquete una factura, en el detalle figura el paquete.

Un turista podrá pagar más de un paquete en una misma factura.

Funcionalidad necesaria del sistema(aspectos generales).

A nivel funcional el sistema deberá realizar registro de las siguientes entidades indicadas, permitir actualización de información(No hay eliminación de datos).

- Turistas
- Agencias
- Paquetes (Crear Paquete)
- Facturar a turista y su detalle

Las demás entidades que son tablas en la base, gestionar la carga de información mediante script SQL, dejar copia en carpeta “scripts”.

El proyecto presenta gran completitud de código, pero aún así quedan cuestiones pendientes de implementación, como parte del aprendizaje se propone a cada grupo realice los ajustes en el código, para:

- completar las relaciones entre entidades
- completar las funciones de liberación de memoria
- implementaciones de la función toString.

En cuanto a la interacción con el usuario se deben generar menús con las opciones claras y fáciles de utilizar. Se pide incluir formas o pantallas de uso de las opciones o ayuda, de las opciones disponibles de menú.

Por ejemplo

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

Incorporar de las entidades indicadas a crear / actualizar, que permitan la generación de listados en pantalla o en archivo. Permitir al menos ordenamiento por 2 columnas de datos, por ejemplo en el caso de Turista orden por Nombre o Dni.

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. Implementar los requisitos planteados.

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:

    - localidad
    - pais
    - actividad
    - turista(dni,nombres, domicilio, teléfono)
    - agencia
    - paquete
    - factura
    - detalle factura

- ☐ **Listados**
    - ☐ Los turistas de un determinado país
    - ☐ Los paquetes más consumidos
    - ☐ Los destinos más concurridos

- ☐ 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.

Importante

Ejemplo de pantalla de menu principal.

[Menú de opciones]
[1 - Ingresos]
[2 - Actualizaciones]
[3 - Listados]
[4 - Salir]

Opción : 1

[Menú de ingreso de información]

[1 - Turistas]
[2 - Agencias]
…

Opción: 1

[Ingreso de Turista]

1) Ingrese DNI : … – validar, que no exista previamente cargado el dni en la base
2) Ingrese nombres(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 un turista se puede modificar el nombre, domicilio, telefono.

Por otro lado en el ingreso o modificación de una información relacionada, por ejemplo la información del pais del turista, se debe poder elegir de un listado obtenido desde la base.

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

- **pais**
  - codigo
  - nombre

- **forma de pago**
  - codigo
  - forma

- **tipos de actividad**
  - codigo
  - nombre
  - descripcion
  - duracion
  - nivel

- **tipo paquete tipo actividad**
  - legajo
  - dni
  - apellido
  - nombres
  - domicilio
  - telefono

- **factura**
  - codigo
  - cod_tipo_act
  - anio
  - mes
  - importe

- **detalle factura**
  - codigo
  - cod_act_socio
  - anio
  - mes
  - estado
  - importe

- **turistas**
  - dni
  - apellido
  - nombres
  - domicilio
  - telefono
  - cod_ppaisl -- relación con Pais

- **actividades**
  - codigo
  - cod_tipo_act
  - legajo_profe
  - fecha_desde
  - fecha_hasta
  - nivel
- **paquete actividad**
  - codigo paquete
  - codigo actividad
  - importe
  - fecha hora desde
  - fecha hora hasta
  - detalle
- **tipo paquete**
  - codigo
  - nombre
  - descripcion
  - duracion
  - nivel

- **paquete**
  - codigo
  - cod tipo paquete
  - cod_agencia
  - fecha
  - dni_turista
  - nivel

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, saveObj y toString***.

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)
{
  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_Turista*soc;
tur = Turista_new();
/*
Buscar y luego actualizar
...
*/
if( tur→findbykey(tur,22456952) != NOT_FOUND)
{
  tur→setTelefono(tur,”2804411050”);
  ….  // acceso a los atributos por los getters y setters
if  ( tur→saveObj(tur) == true )
{
   printf(“Actualizacion de turista realizada correctamente!\n”);
}
destroyObj(tur); // 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 parámetros de conexión 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=TpFinalLPL
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