Laboratorio de Programación y Lenguajes 2019

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.

Enunciado

El programa **ElectorApp** consiste de una aplicación que permite la consulta y actualización de una base de datos. En una primera parte del uso de la base de datos, se usara el lenguaje imperativo C,
donde habrán más consultas que actualizaciones, en una segunda etapa de desarrollo, se hará uso del lenguaje orientado a objetos C# con el cual se realizaran más operaciones de actualización y utilizaran elementos
gráficos de interfaz para poder interactuar con los usuarios.

Escenario

Dado que estamos en un año muy electoral, para adquirir un poco más de conocimiento al respecto, se me ocurrió implementar una versión reducida de una elección provincial.

Se sabe que tenemos partidos políticos (Todos presentan una lista única, pero no todos los partidos presentan candidaturas en todas las categorías posibles), con respecto a las categorías de elección, a nivel provincial tenemos (Gobernador, Diputados, Consejo de la Magistratura) a nivel municipal/comunal Intendente y Concejales o jefe comunal si es comuna.

Los partidos presentan candidatos para cada ciudad, no en todas se presentan, por ejemplo un partido solo puede participar en categoría de Gobernador y Diputados nada más en todas las localidades, pero no en cat. intendente ni concejales.

Se proveerá

- script con generación de cada tabla con relaciones.
- script con carga de datos de las tablas, inicial.
- proyecto de referencia para iniciar el desarrollo de la aplicación.
- archivos de ejemplos para compilación en Linux.

Estructuración del código fuente

Debido a la gran cantidad de archivos fuente, se distribuyó la funcionalidad por clase, es decir una carpeta /src donde se lista por carpeta cada entidad, allí se dejan los archivos .h y .c con la implementación.

Objetos – tablas

- Categoria
- Seccion
- Circuito
- Escuela
- Mesa
- Localidad
- Partido
- ListaPartido
- ListaPartidoLocalidad
- TelegramaActa
- TelegramaActaVotos

Desarrollo en lenguaje C

Objetivos

Comprender estructuración del programa y llevar a cabo una implementación de características dadas por el enunciado.

Se cuenta con una base de datos cargada con información básica para consultar. Se contara con un listado de funcionalidad, previamente programado para poder hacer uso de la persistencia en la base de datos.

El código entregado tiene un desarrollo de un 80% - 90% de completitud, se indicara y mostrara lo faltante para que este completo.

Se proveerá archivo script para la construcción de los objetos de la base de datos.

Argumentos a procesar por el programa:

- **-l**

Esta opción permite generar listado de consulta, se debe indicar información a listar, los listados salen en pantalla salvo que se agregue otro parámetro más (**-fNombre_archivo_salida.txt)**.

- **partido**

Listar partidos, todos los registrados en la base de datos.

❖ **Formato de salida**

NroPartido1|Nombre 1|Nombre reporta a1\n
NroPartido2|Nombre 2|Nombre reporta a2\n

- **localidad**

Listar clientes, todos los registrados en la base de datos.

❖ **Formato de salida**

Localidad Id1|Nombre 1 \n
Localidad Id2|Nombre 2 \n …..

- **circuito**

Lista los circuitos, todos los registrados en la base de datos.

❖ **Formato de salida**

Circuito_id1|NombreCircuito1|Seccion1|Localidad1 \n

Circuito _id2|NombreCircuito2|Seccion2|Localidad2 \n

- **escuela**

Listar Escuelas, todas las registradas en la base de datos.

- **Formato de salida**

EscuelaId1|Nombre1|Localidad1|Circuito1|Direccion1|\n….

- **mesa**

Listar Mesas, todas las registradas en la base de datos.

- **Formato de salida**

MesaNro1|Escuela|Localidad1|Circuito|Seccion| |\n….

- **categoria**

Listar las categorías registradas en la base de datos.

- **-fnombre de archivo.txt**

Esta opción permite indicar que se generara como salida a un archivo de texto, seguido del parámetro se indica el nombre del archivo de salida, esta opción es aplicable a cada opción indicada de listado.

Por ejemplo: **-flistado.txt**

Observación:

Separador de campos de registros tanto de consulta, como escritura, es el pipe **|**, separador de tuplas **\n**.

Consigna a resolver

Completar el modelo para incorporar las relaciones faltantes hay una de ejemplo verificar como se define e implementa.

Ayuda para la implementación – Librería orm.c

Se deja a disposición la librería para el manejo de la persistencia con la base de datos, un conjunto de archivos fuente que implementan el acceso orm a la base de datos.

El archivo config.h posee algunos #define's, hay una carpeta **src** que posee el fuente c de cada objeto(seccion, categoría, circuito….etc). La persistencia está pensada para que se tome como base el fuente orm.c en carpeta lib , es decir con el diseño de la librería se pretende un funcionamiento similar a la programación de objetos, ya que estamos trabajando con un lenguaje imperativo y no están presentes las construcciones y objetos necesarios para un uso puro del paradigma orientado a objetos. Se crearon las siguientes estructuras, donde simularemos el paradigma.
Se definen en structs propiedades y punteros a funciones, para dar soporte a las operaciones CRU(D): Create, Read, Update, (Delete no se implemento).
Por ejemplo **struct obj_empleado** tenemos la propiedad "nombre", los punteros a función **findAll, findbykey, saveObj toStringObj y getIsNewObj.**

Además por separado se define el constructor **obj_Mesa * Mesa_new()** que devuelve un puntero al objeto de la instancia.

Otra propiedad que posee este "pseudo" objeto, es un **data_set *ds** que permite la interacción directa con los objetos de la librería libpq-fe.h para serializar a PostgreSQL e hidratar desde la base de datos.

Las funciones básicas que posee cada objeto son:

- int (*findAll)(void *self, void **list,char *criteria);
- int (*findbykey)(void *self,…)
- int (*saveObj)(void *self, …)

El método **findAll**, devuelve un listado de todos los objetos de una determinada clase, si se desea listar todo, el ultimo parámetro debe ser NULL, caso contrario se puede especificar condición de selección que se aplicara a un *where* de una sentencia SQL en una capa de más bajo nivel. Devuelve la cantidad de elementos seleccionados, necesario para recorrer el objeto **list.**

Por ejemplo para recorrer listado de todos los pacientes obtenidos por este método:

```
obj_Categoria *cat, * cat_row;
void *list;
int i,size=0;
cat = Categoria_new();
size = cat->findAll(cat,&list,NULL); // se invoca sin criterio - listar todos...
  for(i=0;i<size;++i)
  {
    cat_row = ((obj_Categoria**)list)[i];
    printf("%s\n", cat_row-> getNombreCategoria (cat_row));
  }
```

Aclaracion 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 mesa por id /nro:

```
obj_Mesa *m;
m = Mesa_new();
if(m->findbykey(m,1) != -1)
{
  obj_Escuela *esc= m->getEscuelaObj(m);
  printf("NRO:%d –Escuela: %s - CantElectores: %d \n",m->getNroMesa(m),
          esc->getLocalidadObj(esc),   m->getCantElectores(m));
}
```

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:

```
obj_ActaTelegramaVotos *telActaVoto;
telActaVoto = ActaTelegramaVotos_new();
/*
verificar como ubicar para cargar todas las posibles opciones para la mesa dada.
...
*/
telActaVoto->setCantVotos(telActaVoto,50);
 telActaVoto->saveObj(telActaVoto);
```