
# Laboratorio de Programación y Lenguajes 2018

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 **comercioApp** 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

Un Comercio posee un conjunto de productos categorizados, que son provistos por diferentes proveedores, posee listado de clientes a los cuales les vende, previo a una orden cargada por empleado del comercio. En un primer análisis de la aplicación se organizo el modelo de datos con el conjunto de tablas que están a continuación.

### Objetos – tablas

- Clientes
- Empleados
- Productos
- Ordenes
- Categorias
- Proveedores
- Categorias
- Detalle_Ordenes

### 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)**.

  - **empleado**
    Listar empleados, todos los registrados en la base de datos.
    - **Formato de salida**
      **Empleado_Id1|Nombre y Apellido1|Nombre reporta a1\n**
      **Empleado_Id2|Nombre y Apellido2|Nombre reporta a2\n**

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

    - **Formato de salida**
      **Cliente_Id1|Nombre Cliente1|Nombre Contacto1\n**
      **Cliente_Id2|Nombre Cliente2|Nombre Contacto2\n …..**

  - **producto**
    Lista los productos, todos los registrados en la base de datos.

    - **Formato de salida**

      **Producto_id1|NombreProducto1|categoriaNombre|Precio1|Existencia1 \n**
      **Producto_id2|NombreProducto2|categoriaNombre|Precio2|Existencia2 \n**

  - **proveedor**
    Listar Proveedores, todos los registrados en la base de datos.

    - **Formato de salida**

      **ProveedorId1|NombreProv1|Contacto1|celu1|fijo1\n….**

  - **orden[-det]**
    Listar las ordenes registradas en la base de datos. Si se indica el(***-det***) se deberá agregar todos los detalles por orden

    - **Formato de salida**

      **OrdenId1|FechaOrden1|Descuento1 \n|….**
      **Si hay detalle va debajo de cada renglón de "Orden"**
      **- Det_Nro1|Nombre Prod1|Cant1\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**

***Procesar altas de objetos, es opcional progamar para todos los objetos del modelo,***
***Pero se podrá distribuir entre los grupos procesamiento de alta de:***

***Cliente – producto a un grupo.***
***Proveedor-Categoria a otro grupo***
***Empleado- Orden – Detalle para otro grupo***
***Validar cada ingreso de información e indicar con cartel apropiado.***

Las operaciones de alta se realizan de a una vez por ejecución del programa y los parámetros van separado por espacios según el objeto, si la cantidad de parámetros a guardar no es correcta indicar msj de error y cortar ejecución..

- **-a** Permite agregar información a la base de datos
  - **-producto**
    Permite agregar nuevo producto… seguido van los argumentos….
  - **-cliente**
    Permite agregar nuevo cliente … seguido van los argumentos….
  - **-proveedor**
    Permite agregar nuevo proveedor… seguido van los argumentos….
  - **-empleado**
    Permite agregar nuevo empleado… seguido van los argumentos….

    … etc

**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(cliente, categoría….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 showObj y getIsNewObj.**

Además por separado se define el constructor **obj_empleado * empleado_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_empleado *emp, * emp_row;
void *list;
int i,size=0;
emp = empleado_new();
size = emp->findAll(emp,&list,NULL); // se invoca sin criterio - listar todos...
  for(i=0;i<size;++i)
  {
    emp_row = ((obj_empleado**)list)[i];
    printf("%s\n", emp_row->getNombre(emp_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 paciente cuyo DNI es **22445717**:

```
obj_producto *p;
p = producto_new();
if(p->findbykey(p,1) != -1)
{
  printf("Nombre:%s – Precio: %.2f - Existencia: %d \n",p->getDescripcion(p), p->
getPrecioUnit(p),   p-> getExistencia(p));
}
```

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 paciente dado su dni:

```
obj_producto *p;
p = producto_new();
if(p->findbykey(p,1) != -1)
{
  p->setExistencia(p,100);
  p->saveObj(p);
}
```

Las estructuras públicas (config.h) para usar en la aplicación son aquellas cuyo prefijo sea **obj_**

- **obj_categoria**
- **obj_cliente**
- **obj_proveedor**
- **obj_empleado**
- **obj_orden**
- **obj_producto**
- **obj_orden_det**

En el programa se trabajará con punteros a esas estructuras.
Los constructores para obtener nueva referencia a los objetos son:

- **obj_categoria *c = categoria _new();**
- **obj_empleado *e = empleado_new();**
- **obj_cliente *cli= cliente_new();**
- **obj_proveedor *prov= proveedor_new();**
- **obj_orden *o= orden_new();**
- **obj_producto *prov= producto_new();**
- **obj_orden_det *odet = orden_det_new();**

Las relaciones se definen con prefijo ***getXxxxxObj*** por ejemplo **getEmpleadoObj**, en objeto obj_orden:

- getEmpleadoObj: Implementa objeto relación dado que posee referencia foránea mediante empleado_id, internamente posee una instancia que se usa temporalmente para referenciar a atributos de un objeto relación.
  Se debe liberar al momento de terminar el uso de la instancia, posee función **destroyInternal** que se debe programar para tal fin.

Son punteros a funciones que devuelven referencia a objeto empleado.

Para acceder a las propiedades de las instancias se poseen punteros a función getters y setters.