



# APP CRUD C#

Primero DAW/DAM

## **Descripción breve**

Primera toma de contacto para realizar operaciones básicas sobre una base de datos SQL Server utilizando C# y Procedimientos almacenados

**Antº Javier Miras Llamas**

## Tabla de contenido

Base de datos .....	2
Creación de la base de datos, tablas e inserción de datos .....	2
Creación de los procedimientos almacenados .....	5
Clases para la gestión de la base de datos .....	8
Clase CConexionBD .....	8
Clase CProductosBD .....	9
Clase CCategoriasBD .....	16
Clase CMarcasBD .....	18
APP CRUD .....	20
Ventana principal .....	20
Ventana inserción/modificación .....	31
Ejercicios .....	34

# Base de datos

## Creación de la base de datos, tablas e inserción de datos

La base de datos la vamos a crear en SQL Server, la cual tendrá las siguientes tablas:

Tabla	Campos	Tipo	Descripción
categorias	categoria_id	Entero (PK)	Clave primaria
	categoria	Cadena (32)	Nombre de la categoría
marcas	marca_id	Entero (PK)	Clave primaria
	marca	Cadena (32)	Nombre de la marca
productos	producto_id	Entero (PK)	Clave primaria
	categoria_id	Entero (FK)	Clave ajena a la tabla categorías
	marca_id	Entero (FK)	Clave ajena a la tabla marcas
	codigo	Entero	Código del producto
	producto	Cadena (32)	Nombre del producto
	precio	Decimal (10,2)	Precio del producto

El esquema entidad/relación es de la siguiente forma:



Ejecutamos el siguiente script con las instrucciones SQL necesarias para la creación de la base de datos.

```
-- Creación de la base de datos

CREATE DATABASE ERP;
GO

USE ERP;
GO

-- Creación de las tablas

CREATE TABLE categorias(
    categoria_id INT IDENTITY (1,1) PRIMARY KEY,
    categoria NVARCHAR(32)
);
GO

CREATE TABLE marcas(
    marca_id INT IDENTITY (1,1) PRIMARY KEY,
    marca NVARCHAR(32)
);
GO

CREATE TABLE productos(
    producto_id INT IDENTITY (1,1) PRIMARY KEY,
    categoria_id INT,
    marca_id INT,
    código INT,
```

```
producto NVARCHAR(32),
precio DECIMAL(10,2)

CONSTRAINT productos_categorias_fk FOREIGN KEY (categoria_id) REFERENCES categorias(categoria_id),
CONSTRAINT productos_marcas_fk FOREIGN KEY (marca_id) REFERENCES marcas(marca_id)
);
GO

-- Inserción de datos
INSERT INTO categorias VALUES
    (''),
    ('Portátiles'),
    ('PCs escritorio'),
    ('Impresoras'),
    ('Monitores'),
    ('Teclados'),
    ('Tarjetas vídeo'),
    ('Altavoces'),
    ('Micrófonos');
GO

INSERT INTO marcas VALUES
    (''),
    ('Brother'),
    ('HP'),
    ('LG'),
    ('Logitech'),
    ('Lenovo'),
    ('Asus'),
    ('Dell'),
    ('Samsung'),
    ('Gigabyte'),
    ('Epson'),
    ('Nvidia');
GO
```

## Creación de los procedimientos almacenados

Una vez creado la base de datos y las tablas, creamos los procedimientos almacenados para mostrar, insertar y modificar los datos. La gestión de los procedimientos almacenados la veremos más adelante. De momento ejecutaremos las sentencias SQL directamente en la capa de acceso a datos.

```
-- Procedimientos almacenados

-- Seleccionar las categorías
CREATE PROC CategoriasSeleccionar
@categoria_id INT
AS
BEGIN
    IF (@categoria_id = 0)
        SELECT * FROM categorias ORDER BY categoria;
    ELSE
        SELECT * FROM categorias WHERE categoria_id=@categoria_id;
END;
GO

-- Seleccionar las marcas
CREATE PROC MarcasSeleccionar
@marca_id INT
AS
BEGIN
    IF (@marca_id = 0)
        SELECT * FROM marcas ORDER BY marca;
    ELSE
        SELECT * FROM marcas WHERE marca_id=@marca_id;
END;
GO

-- Seleccionar los productos
CREATE PROC ProductosSeleccionar
@producto_id INT
AS
BEGIN
    IF (@producto_id = 0)
        SELECT producto_id AS Id, producto AS Producto, categorias.categoria AS Categoría,
            marcas.marca AS Marca, precio AS Precio
```

```
FROM productos
    INNER JOIN categorias ON productos.categoria_id=categorias.categoria_id
    INNER JOIN marcas ON productos.marca_id=marcas.marca_id
ORDER BY producto, categoria, marca;
ELSE
    SELECT producto_id AS Id, producto AS Producto, categorias.categoria AS Categoría,
        productos.categoria_id AS categoria_id, marcas.marca AS Marca,
        productos.marca_id AS marca_id, precio AS Precio
FROM productos
    INNER JOIN categorias ON productos.categoria_id=categorias.categoria_id
    INNER JOIN marcas ON productos.marca_id=marcas.marca_id
WHERE product_id=@producto_id
ORDER BY producto, categoria, marca;
END;
GO

-- Insertar producto
CREATE PROC ProductosInsertar
@categoria_id INT,
@marca_id INT,
@producto NVARCHAR(32),
@precio DECIMAL(10,2)
AS
BEGIN
    INSERT INTO productos VALUES (@categoria_id, @marca_id, @producto, @precio);

    DECLARE @producto_id INT;

    SET @producto_id = @@IDENTITY;

    RETURN @producto_id;
END;
GO

-- Modificar producto
CREATE PROC ProductosEditar
@categoria_id INT,
@marca_id INT,
@producto NVARCHAR(32),
@precio DECIMAL(10,2)
AS
BEGIN
    UPDATE productos SET categoria_id=@categoria_id, marca_id=@marca_id, producto=@producto, precio=@precio WHERE
```

```
        product_id=@producto_id;  
END;  
GO  
  
-- Borrar producto  
CREATE PROC ProductosBorrar  
@producto_id INT  
AS  
BEGIN  
    DELETE productos WHERE producto_id=@producto_id;  
END;  
GO
```

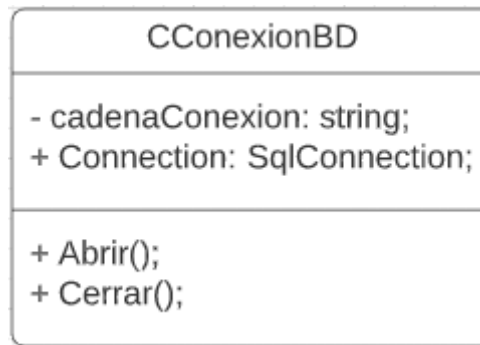


## Clases para la gestión de la base de datos

Vamos a crear las clases para la gestión de la base de datos. La primera que crearemos se la clase para conectarnos a la misma. Después, crearemos tres clases más para la gestión de las tres tablas de nuestro proyecto.

### Clase CConexionBD

La primera clase que crearemos será *CConexionBD* que es la que nos realizará la conexión a la base de datos.



El código de la clase se muestra a continuación.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;

namespace crud
{
    public class CConexionBD
    {
        // Cadena de conexión con la base de datos.
```

```
// En mi caso el servidor es PC-I5, debéis cambiarlo por vuestro servidor.
static private string cadenaConexion = @"Server=PC-I5\SQLEXPRESS;DataBase=ERP;Integrated Security=true;";

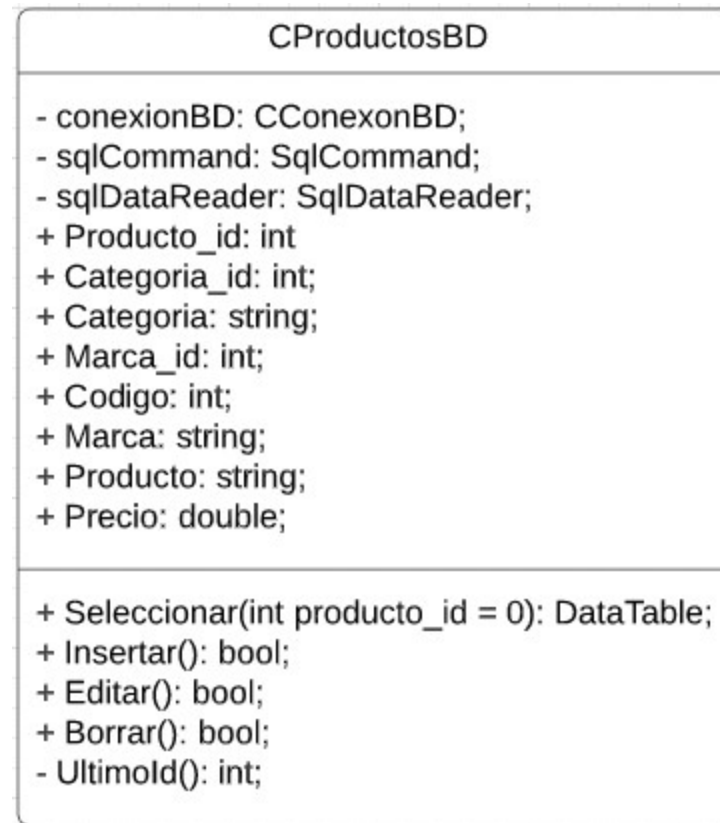
// Conexión a la base de datos.
public SqlConnection Connection { get; } = new SqlConnection(cadenaConexion);

public void Abrir()
{
    // Si la conexión está cerrada, la abrimos.
    if (Connection.State == ConnectionState.Closed)
        Connection.Open();
}

public void Cerrar()
{
    // Si la conexión está abierta, la cerramos.
    if (Connection.State == ConnectionState.Open)
        Connection.Close();
}
}
```

## Clase CProductosBD

La clase *CProductosBD* se encargará de gestionar la tabla *productos* de nuestra base de datos. En ella realizaremos las cuatro operaciones: SELECT, INSERT, UPDATE y DELETE.



El código es el siguiente:

```
using System;
using System.Data;
using System.Data.SqlClient;

namespace crud_procedimientos
{
    public class CProductosBD : CConexionBD
    {
        // Para realizar la conexión a la base de datos.
        private CConexionBD conexionBD = new CConexionBD();

        // Para ejecutar un procedimiento almacenado o realizar las sentencias SQL.
    }
}
```

```
private SqlCommand sqlCommand = new SqlCommand();

// Para almacenar los datos de una sentencia SELECT.
private SqlDataReader sqlDataReader;

// Variable privada para indicar el error que se ha producido.
private String sError;

// Propiedades para almacenar los datos de un registro de la tabla.
public int Producto_id { get; set; }
public int Categoria_id { get; set; }
public String Categoria { get; set; }
public int Marca_id { get; set; }
public String Marca { get; set; }
public String Producto { get; set; }
public double Precio { get; set; }
public intCodigo { get; set; }

// Propiedad que devuelve el error producido.
// Al poner solo "get", le indicamos que es de lectura.
public String Error { get { return sError; } }

public DataTable Seleccionar(int producto_id = 0)
{
    // Para almacenar la tabla leída en memoria.
    DataTable dataTable = new DataTable();

    try
    {
        // Realizamos la conexión.
        conexionBD.Abrir();

        sqlCommand.Connection = conexionBD.Connection;

        // Indicamos el tipo de comando. En este caso un procedimiento almacenado.
        sqlCommand.CommandType = CommandType.StoredProcedure;

        // Sentencia a ejecutar. En este caso un procedimiento almacenado.
        sqlCommand.CommandText = "ProductosSeleccionar";

        // Parámetro pasado al procedimiento almacenado.
        sqlCommand.Parameters.AddWithValue("@producto_id", producto_id);
```

```
// Ejecutamos la sentencia...
sqlDataReader = sqlCommand.ExecuteReader();

// y la guardamos en la tabla leída en la memoria.
dataTable.Load(sqlDataReader);

// Si me indicaron que seleccionase un único registro y este existe.
if ((producto_id != 0) && (dataTable.Rows.Count != 0))
{
    // Obtenemos las filas de la tabla en memoria (En este sólo hay una única fila).
    DataRow[] rows = dataTable.Select();

    // Asignamos a cada propiedad del producto el valor del registro leído.
    Producto_id = producto_id;
    Producto = rows[0]["producto"].ToString();
    Categoria_id = Convert.ToInt32(rows[0]["categoria_id"].ToString());
    Categoria = rows[0]["categoría"].ToString();
    Marca_id = Convert.ToInt32(rows[0]["marca_id"].ToString());
    Marca = rows[0]["marca"].ToString();
    Precio = Convert.ToDouble(rows[0]["precio"].ToString());
    Codigo = Convert.ToInt32(rows[0]["código"].ToString());
}
}
finally
{
    // Limpiamos los parámetros del comando ejecutado.
    sqlCommand.Parameters.Clear();

    // Cerramos los datos leídos.
    sqlDataReader.Close();

    // Cerramos la conexión.
    conexionBD.Cerrar();
}

// Devolvemos la tabla almacenada en memoria.
return dataTable;
}

public bool Insertar()
{
    // Para devolver si la operación se hizo correctamente, o no.
    bool bInsertada = false;
```

```
sError = "";

try
{
    // Es similar a la selección, salvo cambiando el procedimiento almacenado y
    // añadiendo los parámetros correspondientes.
    conexionBD.Abrir();
    sqlCommand.Connection = conexionBD.Connection;

    sqlCommand.CommandType = CommandType.StoredProcedure;
    sqlCommand.CommandText = "ProductosInsertar";
    sqlCommand.Parameters.AddWithValue("@categoria_id", Categoria_id);
    sqlCommand.Parameters.AddWithValue("@marca_id", Marca_id);
    sqlCommand.Parameters.AddWithValue("@producto", Producto);
    sqlCommand.Parameters.AddWithValue("@precio", Precio);
    sqlCommand.Parameters.AddWithValue("@codigo",Codigo);

    // Valor devuelto por el procedimiento almacenado (En este caso la clave primaria).
    var returnParameter = sqlCommand.Parameters.Add("@producto_id", SqlDbType.Int);

    // Indicamos que es un valor de sólo retorno.
    returnParameter.Direction = ParameterDirection.ReturnValue;

    // Ejecutamos la sentencia, indicando que no es una consulta SELECT, y
    // aprovechamos el número de registros que nos devuelve. En este caso debe ser 1.
    bInsertada = sqlCommand.ExecuteNonQuery() == 1;

    // Si la inserción fue correcta, obtenemos el valor de la clave primaria.
    if (bInsertada)
        Producto_id = Convert.ToInt32(returnParameter.Value);
}
catch (Exception ex)
{
    sError = "Código duplicado.\n\n" + ex.Message;

    bInsertada = false;
}
finally
{
    // Limpiar parámetros. Haced esto siempre para que no se acumule en
    // la siguiente llamada a un procedimiento almacenado.
    sqlCommand.Parameters.Clear();
}
```

```
        conexionBD.Cerrar();
    }

    // Devolvemos si la operación fue correcta o no.
    return bInsertada;
}

public bool Borrar()
{
    bool bBorrada = false;

    try
    {
        conexionBD.Abrir();
        sqlCommand.Connection = conexionBD.Connection;

        sqlCommand.CommandType = CommandType.StoredProcedure;
        sqlCommand.CommandText = "ProductosBorrar";
        sqlCommand.Parameters.AddWithValue("@producto_id", Producto_id);

        bBorrada = sqlCommand.ExecuteNonQuery() == 1;
    }
    finally
    {
        conexionBD.Cerrar();
    }

    return bBorrada;
}

public bool Editar()
{
    bool bEditada = false;

    sError = "";

    try
    {
        conexionBD.Abrir();
        sqlCommand.Connection = conexionBD.Connection;

        sqlCommand.CommandType = CommandType.StoredProcedure;
```

```
        sqlCommand.CommandText = "ProductosEditar";
        sqlCommand.Parameters.AddWithValue("@producto_id", Producto_id);
        sqlCommand.Parameters.AddWithValue("@categoria_id", Categoria_id);
        sqlCommand.Parameters.AddWithValue("@marca_id", Marca_id);
        sqlCommand.Parameters.AddWithValue("@producto", Producto);
        sqlCommand.Parameters.AddWithValue("@precio", Precio);
        sqlCommand.Parameters.AddWithValue("@codigo", Codigo);

        bEditada = sqlCommand.ExecuteNonQuery() == 1;
    }
    catch (Exception ex)
    {
        sError = "Código duplicado.\n\n" + ex.Message;

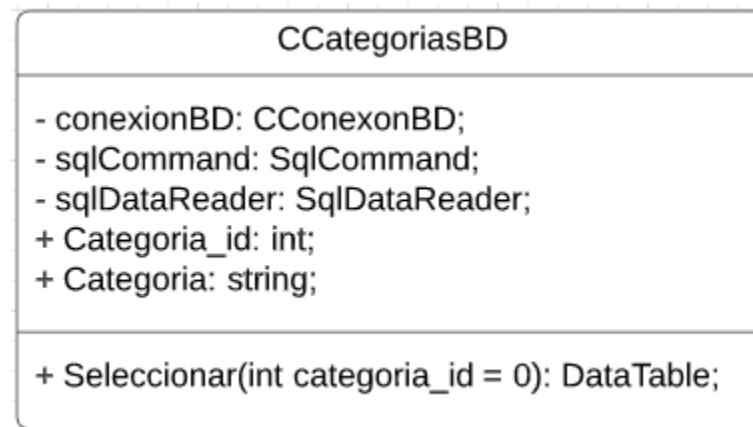
        bEditada = false;
    }
    finally
    {
        sqlCommand.Parameters.Clear();
        conexionBD.Cerrar();
    }

    return bEditada;
}
}
```



## Clase CCategoriasBD

La clase *CCategoriasBD* se encargará de gestionar la tabla *categorias* de nuestra base de datos. En ella realizamos sólo la operación: SELECT.



El código es el siguiente:

```
using System;
using System.Data;
using System.Data.SqlClient;

namespace crud_procedimientos
{
    class CCategoriasBD
    {
        private CConexionBD conexionBD = new CConexionBD();
        private SqlCommand sqlCommand = new SqlCommand();
        private SqlDataReader sqlDataReader;

        public int Categoria_id { get; set; }
        public String Categoria { get; set; }
    }
}
```

```
public DataTable Seleccionar(int categoria_id = 0)
{
    DataTable dataTable = new DataTable();
    try
    {
        conexionBD.Abrir();
        sqlCommand.Connection = conexionBD.Connection;

        // Esto cambia con respecto a trabajar con sentencias SQL.
        sqlCommand.CommandType = CommandType.StoredProcedure;
        sqlCommand.CommandText = "CategoriasSeleccionar";
        sqlCommand.Parameters.AddWithValue("@categoria_id", categoria_id);

        sqlDataReader = sqlCommand.ExecuteReader();

        dataTable.Load(sqlDataReader);

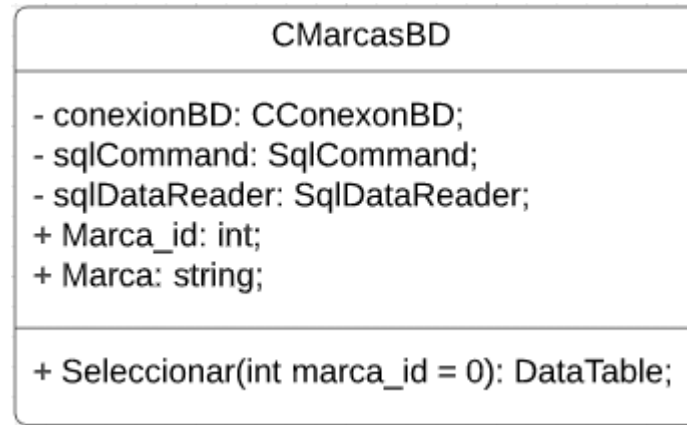
        if ((categoria_id != 0) &&
            (dataTable.Rows.Count != 0))
        {
            DataRow[] rows = dataTable.Select();

            Categoria_id = categoria_id;
            Categoria = rows[0]["categoría"].ToString();
        }
    }
    finally
    {
        sqlCommand.Parameters.Clear();
        sqlDataReader.Close();
        conexionBD.Cerrar();
    }

    return dataTable;
}
```

## Clase CMarcasBD

La clase *CMarcasBD* se encargará de gestionar la tabla *marcas* de nuestra base de datos. En ella realizamos sólo la operación: SELECT.



El código es el siguiente:

```
using System;
using System.Data;
using System.Data.SqlClient;

namespace crud_procedimientos
{
    class CMarcasBD
    {
        private CConexionBD conexionBD = new CConexionBD();
        private SqlCommand sqlCommand = new SqlCommand();
        private SqlDataReader sqlDataReader;
        private String sError = "";

        public int Marca_id { get; set; }
        public String Marca { get; set; }
        public intCodigo { get; set; }
        public String Error { get; }
    }
}
```

```
public DataTable Seleccionar(int marca_id = 0)
{
    DataTable dataTable = new DataTable();
    try
    {
        conexionBD.Abrir();
        sqlCommand.Connection = conexionBD.Connection;

        sqlCommand.CommandType = CommandType.StoredProcedure;
        sqlCommand.CommandText = "MarcasSeleccionar";
        sqlCommand.Parameters.AddWithValue("@marca_id", marca_id);

        sqlDataReader = sqlCommand.ExecuteReader();

        dataTable.Load(sqlDataReader);

        if ((marca_id != 0) && (dataTable.Rows.Count != 0))
        {
            DataRow[] rows = dataTable.Select();

            Marca_id = marca_id;
            Marca = rows[0]["marca"].ToString();
            Codigo = Convert.ToInt32(rows[0]["codigo"]);
        }
    }
    finally
    {
        sqlCommand.Parameters.Clear();
        sqlDataReader.Close();
        conexionBD.Cerrar();
    }

    return dataTable;
}
}
```

# APP CRUD

## Ventana principal

Para gestionar nuestro CRUD crearemos una aplicación de Windows Form, a la que llamaremos *crud*.

La ventana principal tendrá las siguientes propiedades:

Componente	Propiedad	Valor
Form	(Name)	FPrincipal
	ControlBox	false
	MaximizeBox	false
	MinimizeBox	false
	MinimunSize	600;480
	StartPosition	CenterScreen
	Text	CRUD C#

Le añadimos los siguientes componentes:

Componente	Propiedad	Valor
DataGridView	(Name)	dataGridView
	Anchor	Top, Bottom, Left, Right
	MultiSelect	False
	AutoSizeColumnsMode	AllCells
	MultiSelect	false

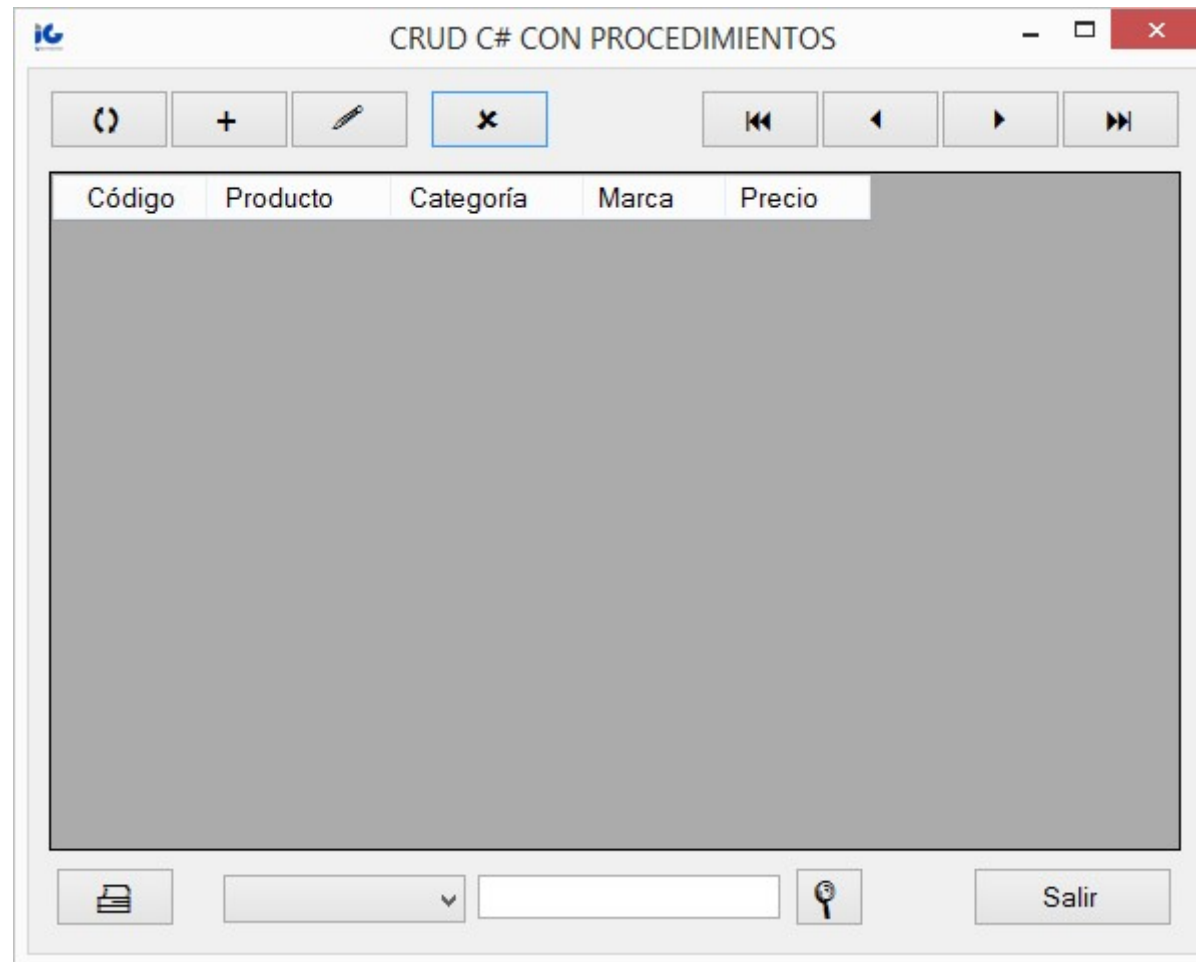
	ReadOnly	True
	RowHeadersVisible	false
	SelectionMode	FullRowSelect
Button	(Name)	btnRecargar
	Cursor	Hand
	Font	Webdings
	Text	q
Button	(Name)	btnNuevo
	Cursor	Hand
	Text	+
Button	(Name)	btnEditar
	Cursor	Hand
	Font	Webdings
	Text	!
Button	(Name)	btnBorrar
	Cursor	Hand
	Font	Webdings
	Text	û
Button	(Name)	btnPrimero
	Anchor	Top, Right
	Cursor	Hand
	Font	Webdings
	Text	9
Button	(Name)	btnAnterior

	Anchor	Top, Right
	Cursor	Hand
	Font	Webdings
	Text	3
Button	(Name)	btnSiguiente
	Anchor	Top, Right
	Cursor	Hand
	Font	Webdings
	Text	4
Button	(Name)	btnUltimo
	Anchor	Top, Right
	Cursor	Hand
	Font	Webdings
	Text	:
Button	(Name)	btnSalir
	Anchor	Bottom, Right
	Cursor	Hand
	Text	Salir
Button	(Name)	btnImprimir
	Anchor	Bottom, Left
	Cursor	Hand
	Text	Imprimir
Button	(Name)	btnBuscar
	Anchor	Top, left

	Cursor	Hand
	Text	L
	Font	Webdings; 18pt
ComboBox	(Name)	cbCampos
	DropDownStyle	DropDownList
	TabIndex	-1
	Items	Categoría Código Marca Producto
TextBox	(Name)	txtBuscar

La app tendrá un aspecto similar al que se muestra a continuación.





El código asociado a la ventana principal se indica a continuación.

```
using System;
using System.Data;
using System.Linq;
using System.Windows.Forms;

namespace crud_procedimientos
{
```

```
public partial class FPrincipal : Form
{
    public FPrincipal()
    {
        InitializeComponent();
    }

    private void btnNuevo_Click(object sender, EventArgs e)
    {
        // Instanciamos la clase FProductosModificar para introducir los datos.
        FProductosModificar fProductosModificar = new FProductosModificar();

        // Mostramos el cuadro de diálogo.
        fProductosModificar.ShowDialog();

        // Si se ha pulsado el botón de aceptar.
        if (fProductosModificar.DialogResult == DialogResult.OK)
        {
            // Recargamos la tabla.
            Recargar();

            // Obtenemos la clave primaria del producto insertado.
            int producto_id = fProductosModificar.Producto_id;

            // Buscamos la fila del producto insertado.
            int rowIndex = dataGridView.Rows
                .Cast<DataGridViewRow>()
                .Where(r => r.Cells[0].Value.Equals(producto_id))
                .First()
                .Index;

            // Nos posicionamos en ella.
            dataGridView.CurrentCell = dataGridView[1, rowIndex];
        }
    }

    private void btnEditar_Click(object sender, EventArgs e)
    {
        // Si tenemos registros en la tabla.
        if (dataGridView.RowCount > 0)
        {
            // Obtenemos la clave primaria del producto.
            int producto_id = Convert.ToInt32(dataGridView.CurrentRow.Cells[0].Value);
        }
    }
}
```

```
// Instanciamos la clase FProductosModificar para modificar los datos.
// Observar que le pasamos en el constructor la clave primaria.
FProductosModificar fProductosModificar = new FProductosModificar(producto_id);

// Mostramos el cuadro de diálogo.
fProductosModificar.ShowDialog();

// Si se ha pulsado el botón de aceptar.
if (fProductosModificar.DialogResult == DialogResult.OK)
{
    // Recargamos la tabla.
    Recargar();

    // Buscamos la fila del producto editado.
    int rowIndex = dataGridView.Rows
        .Cast<DataGridViewRow>()
        .Where(r => r.Cells[0].Value.Equals(producto_id))
        .First()
        .Index;

    // Nos posicionamos en ella.
    dataGridView.CurrentCell = dataGridView[1, rowIndex];
}
}

private void btnBorrar_Click(object sender, EventArgs e)
{
    // Si tenemos registros en la tabla y...
    // el usuario me confirma que realmente quiere borrar el registro.
    if ((dataGridView.RowCount > 0) &&
        (MessageBox.Show("¿Realmente quiere borrar el producto seleccionado?",
            "Confirmación",
            MessageBoxButtons.YesNo,
            MessageBoxIcon.Question) == DialogResult.Yes))
    {
        // Creamos una instancia de la clase CProductosBD.
        CProductosBD productosBD = new CProductosBD();

        // Obtenemos la clave principal del producto a borrar.
        productosBD.Producto_id = Convert.ToInt32(dataGridView.CurrentRow.Cells[0].Value);
    }
}
```

```
// Si el producto se borra correctamente.
if (productosBD.Borrar())
{
    // Obtenemos la fila actual.
    int rowIndex = dataGridView.CurrentRow.Index;

    // Recargamos y vamos a la fila actual, que corresponderá al siguiente producto.
    Recargar(rowIndex);
}
else
    // Sino se ha podido borrar, mensaje de error.
    MessageBox.Show("Al borrar el producto.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

private void btnRecargar_Click(object sender, EventArgs e)
{
    // Miramos en qué fila nos encontramos.
    // Si no tenemos filas, nos posicionamos en la primera (0).
    // En caso contrario, en la fila actual del DataGridView.
    // Observar la utilidad, en este caso, del operador ternario. Más limpio que utilizar un if.
    int rowIndex = (dataGridView.RowCount == 0) ? 0 : dataGridView.CurrentRow.Index;

    // Otra forma de mirar en qué fila nos encontramos sería con un if:
    //
    // int rowIndex = 0;
    //
    // if (dataGridView.RowCount > 0)
    //     rowIndex = dataGridView.CurrentRow.Index;

    // Llamamos al procedimiento recargar y nos posicionamos en la fila actual.
    Recargar(rowIndex);
}

private void Recargar(int rowIndex = 0)
{
    // Instanciamos la clase CProductosBD.
    CProductosBD productosBD = new CProductosBD();

    // Recargamos el DataGridView asociando el DataSource con los datos devueltos.
    dataGridView.DataSource = productosBD.Seleccionar();
}
```

```
// Si tenemos datos...
if (dataGridView.RowCount > 0)
{
    // Comprobamos que la fila que nos indican no es superior a la cantidad de filas que tenemos.
    // Si es así, nos posicionamos en la última fila.
    if (rowIndex >= dataGridView.RowCount)
        rowIndex = dataGridView.RowCount - 1;

    // Si nos indican una fila negativa, nos posicionamos en la primera.
    // Si nos indican una fila negativa, nos posicionamos en la primera.
    if (rowIndex < 0)
        rowIndex = 0;

    // Nos posicionamos en la fila indicada.
    dataGridView.CurrentCell = dataGridView[1, rowIndex];
}
}

private void FPrincipal_Load(object sender, EventArgs e)
{
    // Cargamos la tabla de productos.
    Recargar();

    // No permitimos que nos inserten filas a través del DataGridView.
    dataGridView.AllowUserToAddRows = false;

    // Las filas de la cabecera las ponemos centradas.
    dataGridView.ColumnHeadersDefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleCenter;

    // Ocultamos la columna que muestra la clave primaria "id"
    dataGridView.Columns["id"].Visible = false;

    // La columna con los precios la mostramos formateada como moneda (currency)...
    dataGridView.Columns["Precio"].DefaultCellStyle.Format = "c";

    // y la alineamos a la derecha.
    dataGridView.Columns["Precio"].DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleRight;

    // Si hay algún valor null, lo mostraremos con tres guiones.
    dataGridView.DefaultCellStyle.NullValue = "---";
}

private void btnSalir_Click(object sender, EventArgs e)
```

```
{
    // Si el usuarios realmente quiere salir, cerramos la app.
    if (MessageBox.Show("¿Realmente quiere salir de la App?",
        "Confirmación",
        MessageBoxButtons.YesNo,
        MessageBoxIcon.Question) == DialogResult.Yes)
    {
        Close();
    }
}

private void btnPrimero_Click(object sender, EventArgs e)
{
    // Nos posicionamos en la primera fila del DataGridView.
    dataGridView.CurrentCell = dataGridView[1, 0];
}

private void btnAnterior_Click(object sender, EventArgs e)
{
    // Buscamos la fila anterior.
    int rowIndex = dataGridView.CurrentRow.Index - 1;

    // Si es negativa, es porque estábamos ya en la primera fila.
    if (rowIndex < 0)
        rowIndex = 0;

    // Nos posicionamos en la fila del DataGridView.
    dataGridView.CurrentCell = dataGridView[1, rowIndex];
}

private void btnSiguiente_Click(object sender, EventArgs e)
{
    // Buscamos la fila siguiente.
    int rowIndex = dataGridView.CurrentRow.Index + 1;

    // Si es mayor que la cantidad de filas que hay en el DataGridView, entonces nos vamos a la última fila.
    if (rowIndex >= dataGridView.RowCount)
        rowIndex = dataGridView.RowCount - 1;

    // Nos posicionamos en la fila del DataGridView.
    dataGridView.CurrentCell = dataGridView[1, rowIndex];
}
```

```
private void btnUltimo_Click(object sender, EventArgs e)
{
    // Buscamos la última fila.
    int rowIndex = dataGridView.RowCount - 1;

    // Si no había filas en el DataGridView, entonces la fila será la primera.
    if (rowIndex < 0)
        rowIndex = 0;

    // Nos posicionamos en la fila del DataGridView.
    dataGridView.CurrentCell = dataGridView[1, rowIndex];
}

private void btImprimir_Click(object sender, EventArgs e)
{
    CProductosPDF cProductosPDF = new CProductosPDF();

    try
    {
        cProductosPDF.Imprimir();

        MessageBox.Show("Archivo \"test.pdf\" se ha generado correctamente.",
            "PDF", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "PDF", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void btBuscar_Click(object sender, EventArgs e)
{
    if (cbCampos.Text == "")
    {
        MessageBox.Show("Debe introducir un campo a filtrar",
            "Filtrar", MessageBoxButtons.OK, MessageBoxIcon.Error);

        return;
    }

    String sFiltro = "";

    if (txtBuscar.Text != "")
```

```
{
    sFiltro = cbCampos.Text + " " + txtBuscar.Text;
}

try
{
    (dataGridView.DataSource as DataTable).DefaultView.RowFilter = sFiltro;
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "Filtrar", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}
}
```

## Ventana inserción/modificación

A continuación, añadiremos una nueva ventana con las siguientes propiedades:

Componente	Propiedad	Valor
Form	(Name)	FProductosModificar
	FormBorderStyle	FixedDialog
	StartPosition	CenterScreen
	Text	Productos :: Nuevo

Al formulario le añadiremos los siguientes componentes:

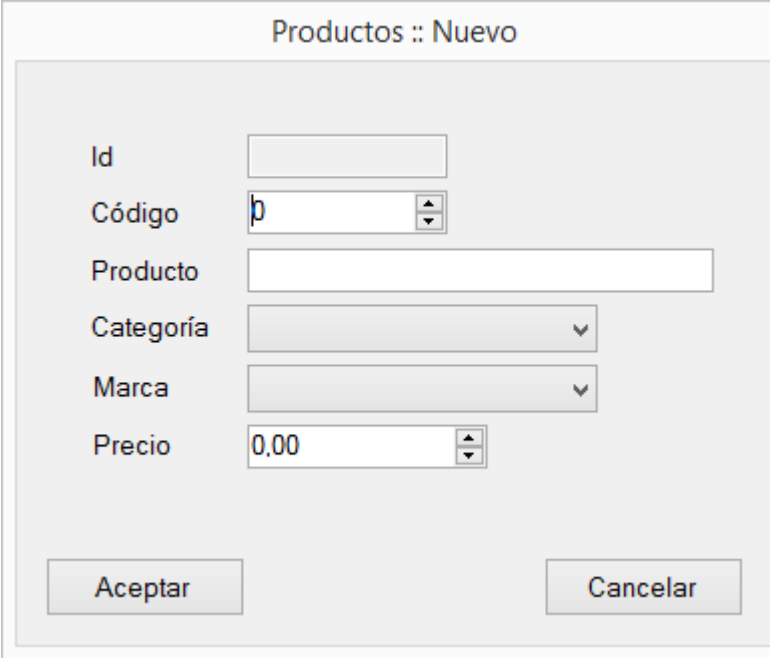
Componente	Propiedad	Valor
Button	(Name)	btnAceptar
	Cursor	Hand



	TabStop	False
	Text	Aceptar
Button	(Name)	btnCancelar
	Cursor	Hand
	DialogResult	Cancel
	TabStop	False
	Text	Cancelar
TextBox	(Name)	txtId
	ReadOnly	true
	TabStop	False
Label	(Name)	lblId
	Text	Id
NumericUpDown	(Name)	txtCodigo
	DecimalPlaces	0
	Increment	1
	Maximum	1000000
	TabIndex	0
	ThousandsSeparator	false
TextBox	(Name)	txtProducto
	TabIndex	1
Label	(Name)	lblProducto
	Text	Producto
ComboBox	(Name)	cbCategorias
	DropDownStyle	DropDownList

	TabIndex	2
Label	(Name)	lblCategorias
	Text	Categorías
ComboBox	(Name)	cbMarcas
	DropDownStyle	DropDownList
	TabIndex	3
Label	(Name)	lblMarcas
	Text	Marcas
NumericUpDown	(Name)	txtPrecio
	DecimalPlaces	2
	Increment	0,05
	Maximum	1000000
	TabIndex	4
	ThousandsSeparator	true

El aspecto del formulario será el siguiente.



Productos :: Nuevo

Id

Código

Producto

Categoría

Marca

Precio

Aceptar Cancelar

El código asociado a la ventana de inserción/modificación de un producto es el siguiente.

```
using System;
using System.Windows.Forms;

namespace crud_procedimientos
{
    public partial class FProductosModificar : Form
    {
        public int Producto_id { get; set; }

        public FProductosModificar(int producto_id = 0)
        {
            InitializeComponent();

            // Clave primaria del producto indicado.
            Producto_id = producto_id;
        }
    }
}
```

```
private void FProductosModificar_Load(object sender, EventArgs e)
{
    // Instanciamos las clases CCategoriasBD y CMarcasBD.
    CCategoriasBD categoriasBD = new CCategoriasBD();
    CMarcasBD marcasBD = new CMarcasBD();

    // Obtenemos todos los registros de la tabla.
    cbCategorias.DataSource = categoriasBD.Seleccionar();

    // Mostramos el valor del campo categoría.
    cbCategorias.DisplayMember = "categoria";

    // Indicamos que le valor seleccionado es la clave primaria.
    cbCategorias.ValueMember = "categoria_id";

    // Para las marcas hacemos lo mismo que para las categorías.
    cbMarcas.DataSource = marcasBD.Seleccionar();
    cbMarcas.DisplayMember = "marca";
    cbMarcas.ValueMember = "marca_id";

    // Si me indican un producto en concreto, es que queremos modificarlo.
    if (Producto_id != 0)
    {
        // Instanciamos la clase CProductosBD.
        CProductosBD productosBD = new CProductosBD();

        // Buscamos el producto.
        productosBD.Seleccionar(Producto_id);

        // Mostramos la clave primaria.
        txtId.Text = Convert.ToString(productosBD.Producto_id);

        // El código del producto.
        txtCodigo.Value = productosBD.Codigo;

        // El nombre del producto.
        txtProducto.Text = productosBD.Producto;

        // Buscamos en el ComboBox el índice de la categoría seleccionada.
        cbCategorias.SelectedIndex = cbCategorias.FindStringExact(productosBD.Categoria);
        // cbCategorias.SelectedValue = productosBD.Categoria_id;

        // Otra forma de asignar el índice.
```

```
        // cbMarcas.SelectedIndex = cbMarcas.FindStringExact(productosBD.Marca);
        cbMarcas.SelectedValue = productosBD.Marca_id;

        // Y finalmente, el precio.
        txtPrecio.Value = Convert.ToDecimal(productosBD.Precio);

        // Indicamos que estamos modificando.
        Text = "Productos :: Modificar";
    }
}

private void btnAceptar_Click(object sender, EventArgs e)
{
    // Verificamos que todo es correcto antes de proseguir.
    if (!Correcto())
        return;

    // Por defecto, indicamos que se pulsa el botón OK.
    DialogResult = DialogResult.OK;

    // Instanciamos la clase CProductodBD.
    CProductosBD productosBD = new CProductosBD();

    // Le pasamos a cada una de las propiedades los valores correspondientes.
    productosBD.Producto = txtProducto.Text;
    productosBD.Categoria_id = (int)cbCategorias.SelectedValue;
    productosBD.Marca_id = (int)cbMarcas.SelectedValue;
    productosBD.Precio = Convert.ToDouble(txtPrecio.Value);
    productosBD.Codigo = Convert.ToInt32(txtCodigo.Value);

    // Si estamos insertando...
    if (Producto_id == 0)
    {
        // Insertamos y verificamos que todo ha ido bien.
        if (productosBD.Insertar())
        {
            Producto_id = productosBD.Producto_id;
        }
        else
        {
            MessageBox.Show("Al insertar el producto.\n" + productosBD.Error,
                            "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
}
```

```
        // Si no se ha podido insertar, devolvemos Cancel.
        DialogResult = DialogResult.Cancel;
    }
}
else
{
    // y sino, estamos modificando.

    // Indicamos el producto a modificar.
    productosBD.Producto_id = Producto_id;

    // Verificamos que si ha habido un error.
    if (!productosBD.Editar())
    {
        MessageBox.Show("Al modificar el producto.\n" + productosBD.Error,
            "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

        // Si no se ha podido modificar, devolvemos Cancel.
        DialogResult = DialogResult.Cancel;
    }
}

private bool Correcto()
{
    if (txtProducto.Text == "")
    {
        MessageBox.Show("Debe indicar el nombre del producto",
            "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        txtProducto.Focus();

        return false;
    }

    if (txtCodigo.Value == 0)
    {
        MessageBox.Show("Debe indicar el código del producto",
            "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        txtProducto.Focus();

        return false;
    }
}
```

```
        return true;
    }

    private void Cancelar_Click(object sender, EventArgs e)
    {
    }
}
}
```

## CProductosPDF

A continuación, se muestra el código para realizar la impresión de los productos.

```
using System;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using iTextSharp.text;
using iTextSharp.text.pdf;

namespace crud_procedimientos
{
    class CProductosPDF
    {
        private const float milimetro = 2.83465f; // 1 milímetro son 2.83465 puntos de pantalla.

        public void Imprimir()
        {
            Document document = new Document(); // Documento PDF.

            document.SetPageSize(iTextSharp.text.PageSize.A4); // Tamaño de la hoja A4.
            document.SetMargins(10f * milimetro, // Margen derecho 10 mm ó 1 cm.
                                10f * milimetro, // Margen izquierdo 10 mm ó 1 cm.
                                30f * milimetro, // Margen superior 30 mm ó 3 cm.
                                10f * milimetro); // Margen inferior 10 mm ó cm.

            try
```

```
{
    // Guardaremos el contenido en un fichero "test.pdf"
    PdfWriter pdfWriter = PdfWriter.GetInstance(document, new FileStream("test.pdf", FileMode.Create));

    pdfWriter.PageEvent = new HeaderFooter(); // Indicamos nuestro pie de página personalizado.
}
catch (Exception ex)
{
    // En caso de no poder guardar en "test.pdf" lanzamos excepción.
    throw new Exception("Al guardar el fichero \"test.pdf\".\n\n" + ex.Message);
}

document.Open(); // Abrimos el documento.

try
{
    CProductosBD productos = new CProductosBD(); // Creamos el objeto productos.

    DataRow[] rows = productos.Seleccionar().Select(); // Seleccionamos todos los productos.

    PdfPTable tProductos = new PdfPTable(5); // Tabla para mostrar los productos.
    PdfPCell celdaCabecera = new PdfPCell(); // Celda para la cabecera.
    PdfPCell celdaDerecha = new PdfPCell(); // Celda para alineación derecha.

    tProductos.WidthPercentage = 100f; // La tabla tendrá una anchura del 100%

    celdaDerecha.HorizontalAlignment = Element.ALIGN_RIGHT; // La celda derecha pues eso a la derecha.
    celdaDerecha.FixedHeight = 7f * milimetro;

    celdaCabecera.HorizontalAlignment = Element.ALIGN_CENTER; // La cabecera se alinean al centro,
    celdaCabecera.BackgroundColor = new BaseColor(Color.LightGray); // con el fondo en gris claro
    celdaCabecera.FixedHeight = 7f * milimetro; // y una altura de celda de 7 mm.

    celdaCabecera.Phrase = new Phrase("Código"); // Título de la celda código.
    tProductos.AddCell(celdaCabecera); // Añadimos la celda.

    celdaCabecera.Phrase = new Phrase("Producto");
    tProductos.AddCell(celdaCabecera);

    celdaCabecera.Phrase = new Phrase("Categoría");
    tProductos.AddCell(celdaCabecera);

    celdaCabecera.Phrase = new Phrase("Marca");
```



```
tProductos.AddCell(celdaCabecera);

celdaCabecera.Phrase = new Phrase("Precio");
tProductos.AddCell(celdaCabecera);

tProductos.HeaderRows = 1; // La primera fila, la cabecera, se repetirá en cada nueva página.

// Para cada una de las filas de nuestra consulta...
for (int i = 0; i < rows.Count(); i++)
{
    celdaDerecha.Phrase = new Phrase(rows[i]["Código"].ToString()); // Código.
    tProductos.AddCell(celdaDerecha); // Observar que no lo insertamos directamente porque
                                     // lo queremos alinear a la derecha.

    tProductos.AddCell(rows[i]["Producto"].ToString()); // Producto.
    tProductos.AddCell(rows[i]["Categoría"].ToString()); // Categoría.
    tProductos.AddCell(rows[i]["Marca"].ToString()); // Marca.

    celdaDerecha.Phrase = new Phrase(rows[i]["Precio"].ToString() + " €"); // Precio.
    tProductos.AddCell(celdaDerecha);
}

document.Add(tProductos); // Añadimos la tabla al documento.
}
catch (Exception ex)
{
    // En caso de error lanzamos la excepción con el error que se ha producido.
    throw new System.Exception("Al generar el archivo \"test.pdf\".\n\n" + ex.Message);
}
finally
{
    document.Close(); // y lo cerramos.
}
}

class HeaderFooter : PdfPageEventHelper
{
    private const float milimetro = 2.83465f; // 1 milímetro equivale a 2.83465 puntos.

    public override void OnEndPage(PdfWriter writer, Document document)
    {
        base.OnEndPage(writer, document);
    }
}
```

```
PdfPTable tCabecera = new PdfPTable(3); // Tabla para poner la información en la cabecera.
PdfPTable tPie = new PdfPTable(3); // Tabla para poner la información en el pie.

PdfPCell celdaLogo = new PdfPCell(); // Celda para poner el logo.
PdfPCell celdaTitulo = new PdfPCell(); // Celda para poner el título.
PdfPCell celdaFecha = new PdfPCell(); // Celda para poner la fecha.
PdfPCell celdaNumPagina = new PdfPCell(); // Celda para poner el número de página

iTextSharp.text.Font fontTitulo = FontFactory.GetFont("Arial", 18f); // Fuente para utilizar en el título.
iTextSharp.text.Font fontPie = FontFactory.GetFont("Arial", 10f); // Fuente para utilizar en el pie.

fontTitulo.Color = new BaseColor(0, 71, 185); // Color y
fontTitulo.SetStyle(iTextSharp.text.Font.BOLD); // estilo.

fontPie.Color = fontTitulo.Color; // Color y
fontPie.SetStyle(iTextSharp.text.Font.NORMAL); // estilo.

celdaLogo.VerticalAlignment = Element.ALIGN_MIDDLE; // Alineación vertical de la celda.
celdaLogo.Border = 0; // Sin borde.

celdaTitulo.VerticalAlignment = Element.ALIGN_BOTTOM;
celdaTitulo.HorizontalAlignment = Element.ALIGN_RIGHT; // Alineación horizontal de la celda.
celdaTitulo.Border = 0;

celdaFecha.HorizontalAlignment = Element.ALIGN_LEFT;
celdaFecha.Border = 0;

celdaNumPagina.HorizontalAlignment = Element.ALIGN_RIGHT;
celdaNumPagina.Border = 0;

tCabecera.TotalWidth = document.PageSize.Width - document.LeftMargin - document.RightMargin; // El tamaño de
la tabla para poner la cabecera es del 100%
tCabecera.DefaultCell.Border = 0;

string url = "https://www.igformacion.com/wp-content/uploads/2019/04/logo-igformacion.png"; // Dirección del
logo.

celdaLogo.Image = iTextSharp.text.Image.GetInstance(new Uri(url)); // Cargamos la imagen desde la dirección
indicada en la celda.

celdaTitulo.Phrase = new Phrase("Listado de productos", fontTitulo); // Ponemos el título del listado.
```

```
tCabecera.AddCell(celdaLogo); // Añadimos el logo a la tabla cabecera.
tCabecera.AddCell("");
tCabecera.AddCell(celdaTitulo); // Añadimos el título a la tabla cabecera.

// Escribimos la tabla cabecera
tCabecera.WriteSelectedRows(0, // Desde la posición 0.
    -1, // Toda la tabla (Sólo hay una fila)
    document.LeftMargin, // Left -> Margen izquierda.
    writer.PageSize.GetTop(document.TopMargin) + (25f * milimetro)22, // Top -> Margen superior más 25 mm.
    writer.DirectContent); // Dirección.

tPie.TotalWidth = document.PageSize.Width - document.LeftMargin - document.RightMargin;
tPie.DefaultCell.Border = 0;

celdaFecha.Phrase = new Phrase(DateTime.Now.ToString("dd/MM/yyyy"), fontPie); // Fecha de impresión.

celdaNumPagina.Phrase = new Phrase("Página " + writer.PageNumber, fontPie); // Número de página.

tPie.AddCell(celdaFecha);
tPie.AddCell("");
tPie.AddCell(celdaNumPagina);

tPie.WriteSelectedRows(0, -1, document.LeftMargin,
    writer.PageSize.GetBottom(document.BottomMargin), writer.DirectContent);
}
}
```

El resultado debería ser el siguiente.

Código	Producto	Categoría	Marca	Precio
2	Portátil barato	Portátiles	Samsung	2,35 €
3	PC Salón	PCs escritorio	Dell	12,00 €
4	Impresora	PCs escritorio	Epson	250,00 €
5	Logi	Micrófonos	Logitech	11,56 €
6	GT 200	Tarjetas vídeo	Nvidia	1,45 €
9	Monitor	Monitores	Samsung	98,00 €

## Ejercicios

1. Se tiene que completar la clase CategoriasBD con las operaciones: INSERT, UPDATE y DELETE.
2. Añadir a las categorías el campo código. Este campo es numérico entero y debe ser único.
3. Se tiene que completar la clase CMarcasBD con las operaciones: INSERT, UPDATE y DELETE.
4. Añadir a las marcas el campo código. Este campo es numérico entero y debe ser único.
5. Realizar la impresión de Categorías y Marcas.