TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CD. GUZMÁN

TITULACIÓN INTEGRAL

TESIS

TEMA:

IMPLEMENTACIÓN DE UNA PLATAFORMA DE GESTIÓN EN UN ENTORNO DE DESARROLLO INTEGRADO.

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO INFORMÁTICA

PRESENTA:

CARLOS ERNESTO CÁRDENAS ZEPEDA

ASESOR(A):

MC. RAQUEL OCHOA ORNELAS

CD. GUZMÁN JALISCO, MÉXICO, JUNIO DE 2017

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de Cd. Guzmán

"Año del Centenario de la Promulgación de la Constitución Política de los Estados Unidos Mexicanos"

Cd. Guzmán, Municipio de Zapotlán el Grande, Jal, 8/junio/2017

ASUNTO: Liberación de Proyecto para Titulación Integral.

M.C. FAVIO REY LUA MADRIGAL
JEFE DE LA DIVISION DE ESTUDIOS PROFESIONALES
P R E S E N T E

Por este medio le informo que ha sido liberado el siguiente proyecto para la Titulación Integral:

Nombre del Egresado:	Carlos Ernesto Cárdenas Zepeda
Carrera:	INGENIERÍA INFORMÁTICA
No. De Control:	12290677
Nombre del Proyecto:	"Implementación de una plataforma de gestión de un entorno de desarrollo integrado"
Producto:	TESIS

Agradezco de antemano su valioso apoyo en esta importante actividad para la formación profesional de nuestros egresados.

A T E N T A M E N T E

M.C. RUBÉN ZEPEDA GARCÍA
JEFE DEL DEPTO. SISTEMAS Y COMPUTACIÓN

SEP TecNM
I. T. DE CD. GUZMÁN
SISTEMAS Y COMPUTACIÓN

M.C. RAQUEL OCHOA ORNEALAS	DR. DANIEL FAJARDO DELGADO	M.C. LEONARDO ALCARAZ SILVA
Nombre y Firma del Asesor	Nombre y Firma del Revisor	Nombre y Firma del Revisor

RGV/JMTS/RZG/ljos*

Índice general.

Índice de figuras.

Índice de tablas.

Índice de términos.

Término	Descripción
.NET	La plataforma .NET de Microsoft es un componente de software que puede ser añadido al sistema operativo Windows. Provee un extenso conjunto de soluciones predefinidas para necesidades generales de la programación de aplicaciones, y administra la ejecución de los programas escritos específicamente con la plataforma.
Access	Es un sistema de gestión de bases de datos incluido en el paquete ofimático denominado Microsoft Office.
ActiveX Data Objects	Es uno de los mecanismos que usan los programas de computadoras para comunicarse con las bases de datos, darles órdenes y obtener resultados de ellas.
Angular	AngularJS es Javascript. Es un proyecto de código abierto, realizado en Javascript que contiene un conjunto de librerías útiles para el desarrollo de aplicaciones web y propone una serie de patrones de diseño para llevarlas a cabo. En pocas palabras, es lo que se conoce como un framework para el desarrollo, en esta caso sobre el lenguaje Javascript con programación del lado del cliente.
Blend	Ayuda a diseñar aplicaciones de escritorio de Windows basadas en XAML, aplicaciones web, aplicaciones de Windows Phone y aplicaciones de la Tienda Windows. Proporciona la misma experiencia de diseño XAML básica que Visual Studio y agrega diseñadores visuales para tareas avanzadas como animaciones y comportamientos.
Cloud Computing	Consiste en la posibilidad de ofrecer servicios a través de Internet.
ComboBox	El control ComboBox de Windows Forms se utiliza para mostrar datos en un cuadro combinado desplegable.

Command	Devuelve la parte del argumento de la línea de comandos utilizada para iniciar Visual Basic o algún programa ejecutable desarrollado con Visual Basic.
DataAdapter	Representa un conjunto de comandos SQL y una conexión a una base de datos que se usan para rellenar DataSet y actualizar el origen de datos.
DataGridView	Es un control el cual muestra datos en una cuadrícula personalizable.
DataReader	La recuperación de datos mediante DataReader implica crear una instancia del objeto Command y de un DataReader a continuación, para lo cual se llama a Command.ExecuteReader a fin de recuperar filas de un origen de datos.
DataSet	Es una representación residente en memoria de datos que proporciona un modelo de programación relacional coherente independientemente del origen de datos.
DataSource	Obtiene o establece el origen de datos cuyos datos se están mostrando en el control.
DataTable	Representa una tabla de datos en memoria.
DisplayMember	Obtiene o establece la propiedad que se va a mostrar para el control.
Framework	Un framework, entorno de trabajo o marco de trabajo es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.
GitHub	GitHub es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git.

Handlebars	Es un sencillo sistema de plantillas Javascript basado en Mustache Templates. Handlebars sirve para generar HTML a partir de objetos con datos en formato JSON.
LESS	Es un dinámico lenguaje de hojas de estilo que puede ser compilado en Hojas de estilo en cascada (CSS) y ejecutarse en el lado del cliente o en el lado del servidor.
ListBox	Representa un control de Windows para mostrar una lista de elementos.
Mustache	Es un sistema de plantillas sin lógica (logic-less templates) , desarrollado para varios lenguajes como JavaScript, Ruby, Python, .NET, PHP, etc. Provee plantillas y vistas, las vistas contienen los datos a ser incluidos en las plantillas, en formato JSON.
MySQL	Es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual GPL/Licencia comercial por Oracle Corporation y está considerada como la base datos open source más popular del mundo y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web.
Node.js	Es un intérprete Javascript del lado del servidor que cambia la noción de cómo debería trabajar un servidor. Su meta es permitir a un programador construir aplicaciones altamente escalables y escribir código que maneje decenas de miles de conexiones simultáneas en una sólo una máquina física.
PostgreSQL	Es un Sistema de gestión de bases de datos relacional orientado a objetos y libre, publicado bajo la licencia PostgreSQL, similar a la BSD o la MIT.
Python	Es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.
ReportViewer	Asistente para la generación de reportes.

Sass	(Syntactically Awesome Stylesheets) es un lenguaje de hoja de estilos inicialmente diseñado por Hampton Catlin y desarrollado por Nathan Weizenbaum.
Software	Conjunto de programas y rutinas que permiten a la computadora realizar determinadas tareas.
SQL Server	Es un sistema de manejo de bases de datos del modelo relacional, desarrollado por la empresa Microsoft.
TextBox	Representa un control de cuadro de texto de Windows.
Unity	Es un motor de desarrollo para la creación de juegos y contenidos 3D interactivos, con las características que es completamente integrado y que ofrece innumerables funcionalidades para facilitar el desarrollo de videojuegos.
ValueMember	Obtiene o establece la ruta de acceso de la propiedad que se utilizará como valor real para los elementos en el control.
Windows Azure	Es una plataforma general que tiene diferentes servicios para aplicaciones.
XAMARIN	Es una implementación libre de la plataforma de desarrollo .NET para dispositivos Android, iOS y GNU/Linux.
XAMPP	Es un paquete de instalación independiente de plataforma, software libre, que consiste principalmente en el sistema de gestión de bases de datos MySQL, el servidor web Apache y los intérpretes para lenguajes de script PHP y Perl.

Capítulo 1. Introducción.

Actualmente el desarrollo de sistemas de información ha ido en constante crecimiento. Las pequeñas, medianas y grandes empresas han tenido la necesidad de adoptar el uso de software para controlar y administrar procesos de una manera más eficiente logrando una mejor toma de decisiones y generando como consecuencia el crecimiento y el éxito de las mismas.

Como herramienta principal se utilizó Visual Studio Community 2015 que ofrece un Framework de desarrollo integrado con herramientas programables avanzadas que permiten cubrir todas las necesidades desde el desarrollador hasta ofrecer una interfaz amigable al usuario final, ya que es una plataforma intuitiva y amigable que permite facilitar su utilización en proyectos de desarrollo de sistemas de información.

Este proyecto de investigación se enfocó en implementar el código del sistema de información de acuerdo a los requerimientos de la empresa “Don Colchonero Muebles”. El análisis y diseño fue resultado de una investigación previa que no es contenida en este trabajo.

El contenido de este trabajo presenta una justificación de la investigación que enmarca la necesidad de investigar términos y ejemplos de programación en una versión actual de Visual Studio. Se exponen los objetivos generales y específicos, una revisión a la literatura sobre sistemas de información, características de Visual Studio Community 2015, el control ADO.NET, desarrollo de módulos de programación del sistema incluyendo reportes y la implementación de pruebas.

Capítulo 2. Planteamiento de la investigación.

2.1. Justificación de la investigación.

Actualmente los sistemas de información son indispensables para la gestión de las empresas que a la vez apoyan a una planeación estratégica para la toma de decisiones. Por otra parte, Visual Studio Community 2015 es una herramienta de desarrollo integrado que permite implementar sistemas de información de una manera eficiente.

Esta investigación surge como consecuencia de las necesidades que denota la empresa Don Colchonero Muebles, ya que no disponían de un sistema confiable para el seguimiento de los pedidos, control de mano de obra y materia prima, inventarios de productos terminados, pedidos, ventas, compras, cobros a clientes y pagos a proveedores. Sólo se disponía de la herramienta Excel para llevar el registro de estos datos, lo cual no resultaba eficiente ya que generaba inconsistencias, algunos errores y omisiones.

Para implementar esta plataforma fue necesario recurrir a nuevas técnicas de programación que ofrece Visual Studio Community 2015, descubriendo nuevas clases de acceso a datos con sus respectivas propiedades y métodos.

Durante esta investigación se diseñaron formularios de captura de datos, menús de opciones, consultas en rejillas, así como diversos informes en ReportViewer que permitieron cubrir las necesidades planteadas por la empresa.

2.2. Problema de la investigación.

Desarrollar la implementación de un Sistema Integral de Información utilizando Visual Studio Community 2015 y técnicas de programación avanzada que permitan automatizar las tareas administrativas propias de la empresa Don Colchonero Muebles.

2.3. Objetivo general.

Implementar un sistema de información en un entorno de desarrollo integrado, que logre una administración automatizada integral de las operaciones diarias en la empresa Don Colchonero Muebles, dedicada a la fabricación y comercialización de muebles.

2.4. Objetivos específicos.

- Desarrollar un módulo para el control de ventas y compras.
- Controlar los pagos a proveedores y abonos de clientes.
- Controlar un inventario actualizado de materia prima, productos terminados, así como un registro de mermas y sobrantes.
- Mantener un registro actualizado del personal.
- Generar diversos informes y estadísticas sobre activos y pasivos de la empresa.

Capítulo 3. Revisión de literatura.

3.1. Importancia de los Sistemas de Información.

Según Briceño (2005), un Sistema de Información se define como un conjunto de elementos los cuales interactúan entre sí con el objetivo de apoyar las actividades de una empresa o negocio, estos elementos están definidos en los siguientes: el equipo computacional, recurso humano, datos o información fuente, programas ejecutados por la computadora, las telecomunicaciones, los acuerdos de políticas y reglas de operación.

Un sistema de información realiza cuatro operaciones básicas:

- **Entrada de información**: Proceso en el cual el sistema toma los datos que requiere para procesar la información, por medio de estaciones de trabajo, teclado, diskettes, cintas magnéticas, código de barras, etc.
- **Almacenamiento de información**: Es una de las actividades más importantes que tiene una computadora, ya que a través de esta capacidad el sistema puede recordar y acceder a la información guardada en cualquier momento.
- **Procesamiento de la información**: Esta característica de los sistemas permite la transformación de los datos fuente en información que puede ser utilizada para la toma de decisiones, generación proyecciones financieras y de estados de resultados.
- **Salida de información:** Es la capacidad de un sistema para sacar la información procesada o bien datos de entrada al exterior. Las unidades típicas de salida son las impresoras, graficadores, cintas magnéticas, diskettes, la voz, etc.

Los sistemas de información y las tecnologías de información han cambiado la forma en que operan las organizaciones actuales. A través de su uso se logran importantes mejoras, pues automatizan los procesos operativos, suministran una plataforma de información necesaria para la toma de decisiones y, lo más

importante, su implantación logra ventajas competitivas o reducen la ventaja de los rivales (Briceño, 2005).

Menciona Briceño (2005), qué la implementación e implantación de la tecnología en cuanto a sistemas de información se refiere garantizan un éxito casi seguro en las empresas. Por otra parte, muchas veces las organizaciones no han logrado entrar en la etapa de cambio hacía la era de la información y tecnología, ya que el riesgo al fracaso esta siempre latente debido a las amenazas del mercado y su incapacidad para competir.

Por lo tanto, la administración apropiada de los sistemas de información es un desafío importante para los gerentes. La función de los sistemas de información representa (Briceño, 2005):

- Un área funcional principal dentro de la empresa, que es importante para el éxito empresarial como las funciones de contabilidad, finanzas, administración de operaciones, marketing y administración de recursos humanos.
- Una colaboración importante para le eficiencia operacional, la productividad y la moral del empleado, el servicio y satisfacción del cliente.
- Una fuente importante de información y respaldo importante para la toma de decisiones efectivas por parte de los gerentes.
- Un ingrediente importante para el desarrollo de productos y servicios competitivos que den a las organizaciones una ventaja estratégica en el mercado global.
- Una oportunidad profesional esencial, dinámica y retadora para millones de hombres y mujeres.

3.2. Visual Studio 2015 y nuevas características.

Microsoft Visual Studio es un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Incluye herramientas y servicios necesarios para diseñar, construir y administrar aplicaciones empresariales robustas para la plataforma de Microsoft y tecnologías multiplataforma (Danysoft, 2017).

Visual Studio 2015 al igual que .NET traen en su nueva versión, un gran número de cambios. En este caso en particular, Microsoft parece por fin haber dado el salto a un ecosistema donde interactúa con otras plataformas. Además, queda evidenciado su excelente trabajo en *Cloud Computing* con Azure y .NET es libre y de código abierto (Caminos, 2015).

El nuevo IDE, viene preparado para desarrollar aplicaciones para Windows, pero también para Android, iOS y Windows Phone, de tal modo que las implementaciones al IDE han permitido evolucionarlo. Esto permite desarrollar en C# nativo sin necesidad de usar Java por ejemplo, por medio de XAMARIN. También se pueden crear usando HTML y JavaScript.

Microsoft tiene su propio emulador de aplicaciones Android, pesa 41 MB y puede incluso usarse con otros IDE. No obstante, pertenece al ecosistema de Visual Studio. Admite varias resoluciones de pantalla y diferentes versiones del sistema operativo de Google. Se pueden simular sensores táctiles y es compatible con Hyper-V (es una herramienta de virtualización), ver Figura 1.

Azure es un producto de Microsoft fuerte en la nube desde la era Ballmer. Es por esto, que se tiene la posibilidad de conectar el proyecto con las herramientas de Azure en segundos (Caminos, 2015).

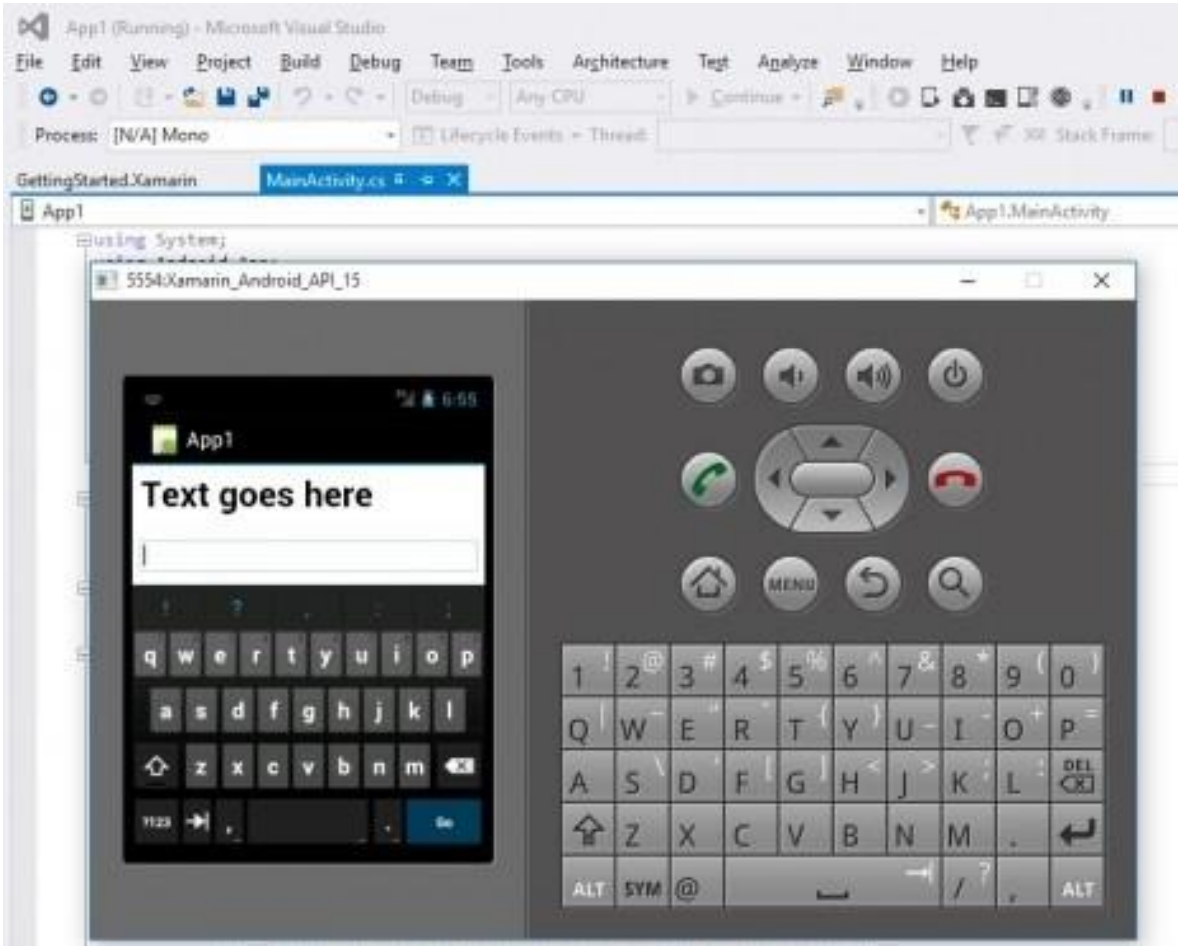


Figura 1. Emulador de aplicaciones Android (Caminos, 2015).

Las aplicaciones universales son una idea que de alguna manera Microsoft ha querido implementar hace tiempo, y que había fracasado un poco con Windows 8. Se trata de que un programa puede correr en una tableta, una PC, un móvil o incluso una consola.

El IDE está preparado para realizar aplicaciones para Windows (modo tradicional) de forma simple. Por intermedio de Blend se pueden crear también interfaces de usuario muy atractivas orientadas al nuevo modelo de apps de la tienda de Microsoft, que es la que unifica y las hace accesibles desde cualquier dispositivo.

Si existe interés por el desarrollo de juegos, el IDE está preparado para poderlo utilizar en conjunto con Unity. Ambos programas, coexisten e interactúan juntos de

forma muy fluida. Cualquier código trabajado en el IDE de Visual Studio es luego añadido a Unity sin problemas, ver Figura 2 (Caminos, 2015).

Figura 2. Aplicaciones en un entorno universal (Caminos, 2015).

La filosofía Open Source es uno de los pilares sobre los que se apoya el nuevo Visual Studio y también .NET.

Esto quiere decir que ya no es una suite de desarrollo cerrado, en la cual solo se puede programar para Windows. Ahora se puede trabajar en la nube por medio de Azure con una máquina virtual Linux o bien se puede acceder a recursos (plantillas de programas) de Microsoft en GitHub, utilizar Python o node.js.

Visual Studio Code por ejemplo, viene a ser un editor ideal para desarrollar para la web con tecnologías libres, como Node.js. Como es habitual se puede disponer de la versión Express del IDE llamada Community, orientada a desarrolladores independientes, instituciones educativas y pequeños equipos. No hay limitación alguna con ella para desarrollar en lenguajes como C#, Visual Basic, web o móviles (Caminos, 2015).

En la nueva versión de Visual Studio se dispone de una mejora sustancial en el depurador propiamente dicho del IDE. Entre sus herramientas de diagnóstico se tiene la posibilidad de recopilar y analizar datos de rendimiento de la aplicación. Se pueden setear puntos de interrupción y con Perftips ver comentarios rápidos y continuos referentes a rendimiento. Posee una herramienta para analizar el uso de memoria. Se pueden tomar instantáneas de la misma y también controlar el uso de CPU y red. Además se pueden realizar depuraciones remotas con solo instalar una herramienta.

En las versiones Enterprise y Professional puede consultarse información de los errores sin siquiera salir del IDE. Permite realizar código fuente de mayor calidad con IntelliTest. Ahora se cuenta con la posibilidad de hacer test unitarios y refactorizar. También, se puede generar mapas de clases en segundos, ver Figura 3 (Caminos, 2015).

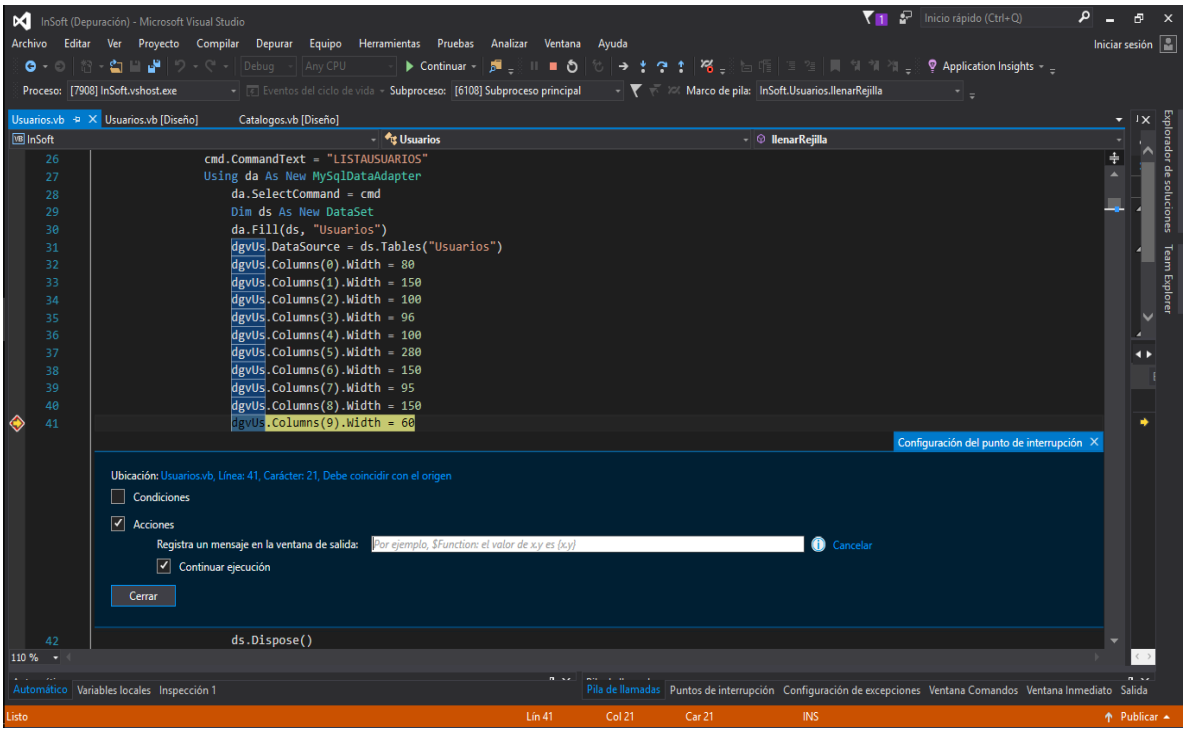


Figura 3. Depurador del IDE.

Microsoft se esmeró en proveer al desarrollador de documentación incluyendo una sección para escribir código, compilar aplicaciones, trabajar en equipo, hacer pruebas manuales, implementar y analizar.

Existe una comunidad y cualquier duda o inquietud que surge se puede consultar en los foros.

En Microsoft (2017), se tienen muestras de código de aplicaciones y ejemplos en todos los lenguajes. Visual Studio Community 2015 ofrece además un centro de conocimiento que incluye tutoriales para el desarrollo en la nube, juegos, C# y aplicaciones .NET. (Caminos, 2015).

La parte de desarrollo web también fue perfeccionada. Hubo muchas mejoras en el trabajo con JSON, ayuda por parte del editor para escribir código HTML y plantillas pre diseñadas con frameworks JavaScript incorporados. Entre ellos se puede contar: Angular, Handlebars y Mustache. También hay posibilidad de trabajar con LESS y Sass, ver Figura 4 (Caminos, 2015).

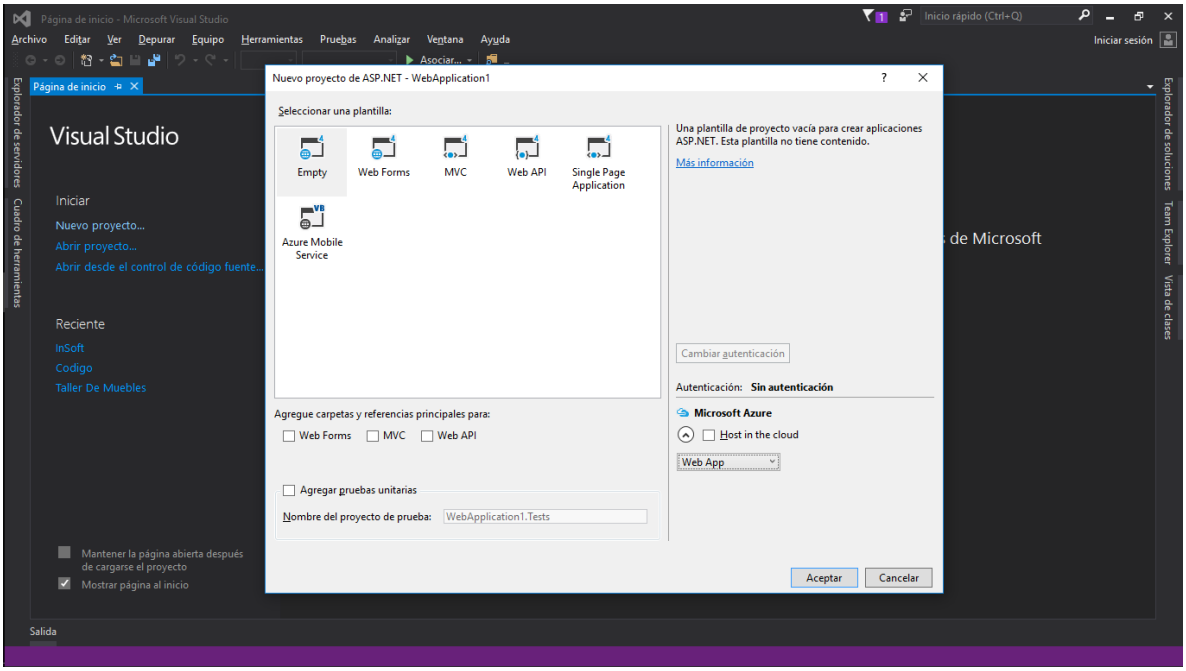


Figura 4. Desarrollo web (Caminos, 2015).

Se incorpora Visual Studio Online donde se tiene la posibilidad de realizar trabajo colaborativo. Se dispone de Rooms para interactuar con el equipo y el resto de los usuarios. Incluye repositorios de código, trabaja con cualquier lenguaje y es posible usarlo con cualquier herramienta, o sea que no están limitados a él IDE de Microsoft.

Se puede usar Eclipse o cualquier editor de código. Hasta cinco usuarios es gratuito, ver Figura 5 (Caminos, 2015).

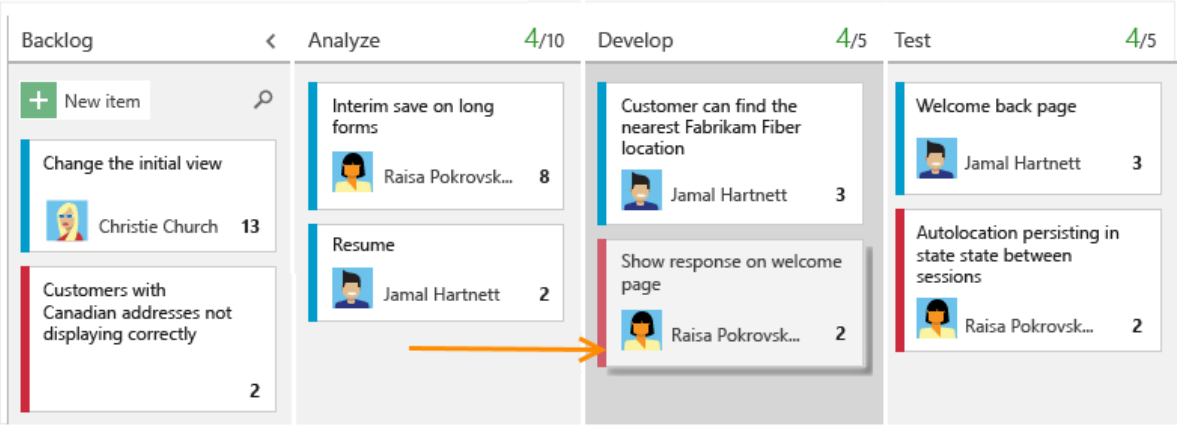


Figura 5. Visual Studio Online (Caminos, 2015).

3.3. Tipos de conexiones a bases de datos.

3.3.1. Conexión a base de datos de Access mediante Visual Studio.

Según Rodriguez (2017), las conexiones a bases de datos en nuevas plataformas o versiones actualizadas cambian un poco, para ello se realizó una búsqueda exhaustiva de información en diferentes tutoriales y medios de información para el manejo de bases de datos desde la plataforma de .NET, donde es importante saber que existen múltiples formas de realizar una conexión a bases de datos, por consiguiente en esta ocasión se realiza mediante un módulo, donde se declaran todos los objetos para el funcionamiento de la conexión.

En primer lugar, se debe importar los nombres de espacios necesarios para el acceso a datos, estos namespace (nombre de espacio) son:

Imports System.Data
Obligatorio para manejo de datos.

Dependiendo del tipo de base de datos a utilizar se utilizará:

Imports System.Data.OleDb (para base de datos Access)

Imports System.Data.SqlClient (para base de datos SQL Server)

Imports System.Data.ODBC (para base de datos mediante un controlador ODBC).

Con lo que los namespace quedarían de la siguiente forma si es para una base de datos Access:

Imports System.Data

Imports System.Data.OleDb

Dentro del módulo se procede a declarar una variable pública del tipo de conexión que se haya declarado en el namespace, para que sea visible desde cualquier parte del proyecto.

```
Module Conexion
  'Declaración de la variable
    Public conectarme As New OleDbConnection
```

Luego se declara un procedimiento público desde donde se maneja la variable que se acaba de crear.

```
  Public Sub conexiones()
  End Sub
End Module
```

Después se asigna a la variable “conectarme” la cadena de conexión, la misma debe hacerse dentro de comilla dobles.

```
conectarme.ConnectionString =
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\data.mdb;"
```

En seguida se hace uso del manejo de errores mediante la instrucción Try:

```
    Try
      If conectarme.State = ConnectionState.Open Then
        MessageBox.Show("Ya está conectado")
        Else
          conectarme.Open()
          'MessageBox.Show("Conectado a Access")
      End If
    Catch ex As Exception
      MessageBox.Show(ex.Message)
    End Try
```

```
  End Sub
End Module
```

A continuación, como se presenta como quedaría el código completo para manejar la conexión.

```
Imports System.Data
Imports System.Data.OleDb
Module Conexion
    Public conectarme As New OleDbConnection
    Public Sub conexiones()
        conectarme.ConnectionString =
        "Provider=Microsoft.Jet.OLEDB.4.0;Data
        Source=C:\data.mdb;"
        Try
            If conectarme.State = ConnectionState.Open Then
                MessageBox.Show("Ya está conectado")
            Else
                conectarme.Open()
                'MessageBox.Show("Conectado a Access")
            End If
        Catch ex As Exception
            MessageBox.Show(ex.Message)
            'Aquí se mostraría un mensaje en caso de que se
            produzca un error.
        End Try
    End Sub
End Module
```

3.3.2. Conexión a base de datos de MySQL mediante Visual Studio.

Pinzón (2014) presenta la conexión a realizar para interactuar con el origen de datos de MySQL y la plataforma de desarrollo de Visual Studio en el lenguaje de programación de Visual Basic, donde se mencionan los siguientes puntos como parte fundamental de la práctica:

1.- Tener instalado el Visual Studio.

2.- Tener instalado MySQL, ya sea por XAMPP, o alguno similar.

3.- Instalar el conector de MySQL con .NET.

Una vez teniendo en correcto funcionamiento todo lo anterior, es necesario crear un nuevo proyecto en Visual Studio y agregar la referencia del conector llamado “MySql.Data.dll”, ver Figura 6.

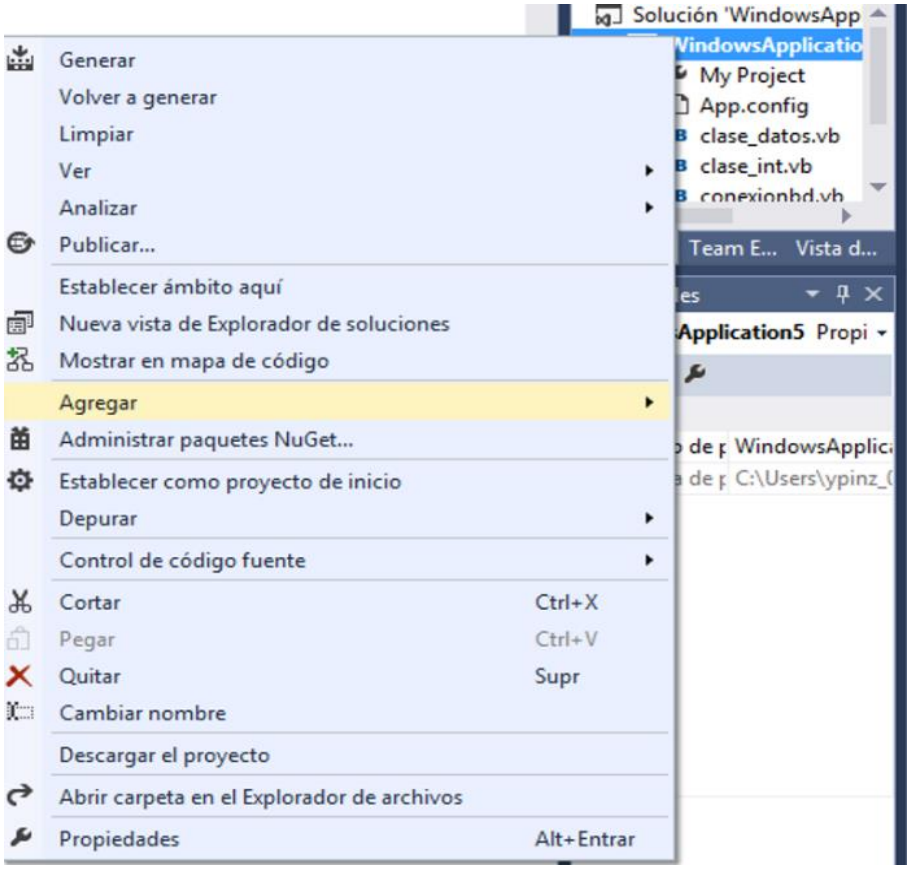


Figura 6. Agregar referencia (Pinzón, 2014).

Una vez enlazado es necesario crear un nuevo módulo, de forma que se debe de dar click derecho al proyecto, seleccionando agregar y posteriormente módulo.

Al nuevo módulo se le llamará “conexionbd”, ver Figura 7 para la creación de un módulo.

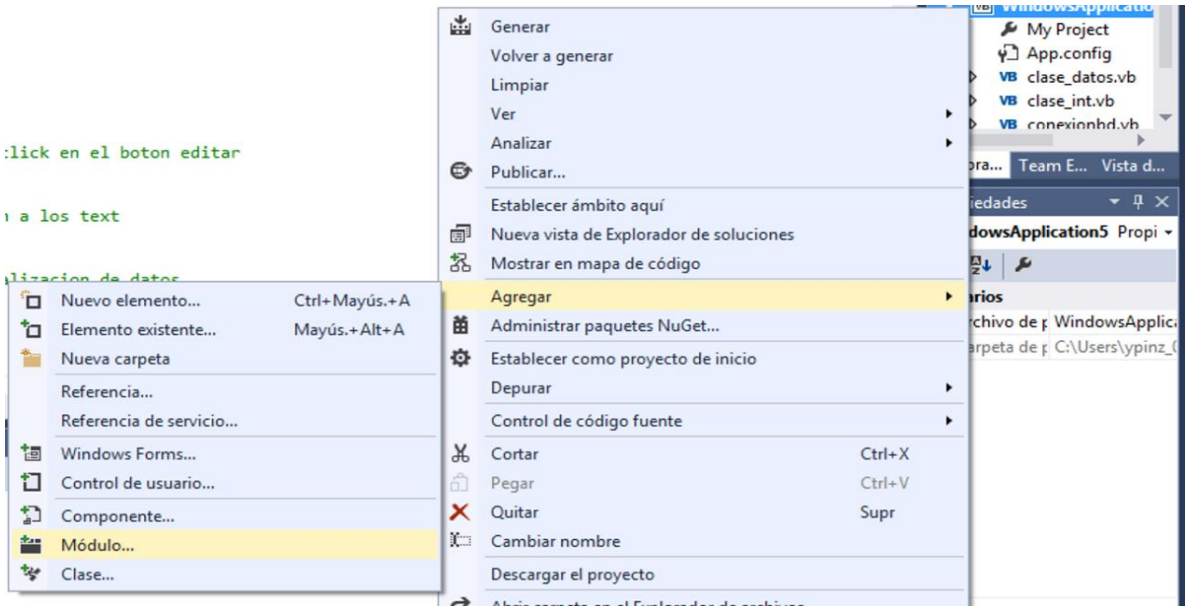


Figura 7. Creación de módulo (Pinzón, 2014).

Dentro del módulo se agregarán las siguientes líneas de comandos.

```
Imports MySql.Data 'en caso de que estos archivos generen algún error, será relacionado la referencia previamente instalada
Imports MySql.Data.Types
Imports MySql.Data.MySqlClient
Module moduloconexglobal
    Public cadena As String 'se usa únicamente para obtener los datos de conexión
    Public conexion As MySqlConnection 'esta variable se encarga de conectar la BD
    Public Function conexion_global() As Boolean 'se crea una función pública boleana, como esta en un módulo, esto evitara colocar el mismo método en diferentes forms
        Dim estado As Boolean = True 'automáticamente se vuelve verdadera la función
        Try
            cadena = ("server=localhost; database=nombre-de-la-base-de-datos; user id=id-de-usuario; password=aqui-va-tu-contraseña") 'aquí se conecta a la BD de mysql
            conexion = New MySqlConnection(_cadena)
            conexion.Open()  'Open hace el enlace con la BD
        Catch ex As Exception 'en caso contrario
```

```
            MessageBox.Show(ex.Message) 'se mostrará un mensaje de
error
            estado = False
        End Try 'fin del try
        Return estado 'en una función siempre debe de existir un
return. El caso es que en cuanto termine de revisar mandará
exactamente el estado en el que se encuentre
    End Function ' se termina la función
    Public Sub cerrar()
        conexion.Close()
    End Sub

End Module
```

Para finalizar en el form se escriben las siguientes líneas

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
    If conexion_global() Then 'if de precaución por si no está
conectada la base de datos
        MsgBox("Bd conectada")
    Else
        MsgBox("precaución no está conectada la BD")
    End If

End Sub
```

De esta forma ya se encuentra conectada una base de datos en MySQL.

3.3.3. Procedimientos almacenados en MySQL.

Un procedimiento almacenado o stored procedure en MySQL es una rutina que puede ser llamada en cualquier momento y que se encuentra almacenada dentro de la base de datos, permite validar datos de entrada, utilizar cualquier valor de la base de datos, devolver solo lo que la misma permita y además, ser usado por cualquier lenguaje de programación (CassiaNet, 2017).

Según CassiaNet (2017), cuando se desarrolla una aplicación por seguridad es necesario implementar validaciones, así como evitar que el usuario tenga que introducir la menor cantidad de datos posibles. Para esto es necesario tratar de

reducir la carga del servidor, dividir el trabajo y conseguir la manera más eficaz de llevar a cabo el desarrollo e implementación de procedimientos almacenados, etc.

Para CassiaNet (2017), es fundamental reducir el tiempo de desarrollo, el mantenimiento de los sistemas, evitar hacer peticiones de grandes cantidades de datos, si la lógica más relevante la dejamos del lado de los gestores de base de datos para la manipulación de información, específicamente, usando los procedimientos almacenados.

Para crear un procedimiento en MySQL es necesario utilizar la directiva CREATE PROCEDURE, al crear el procedimiento almacenado éste es ligado o relacionado con la base de datos que se está usando, de la misma forma en que se crea una tabla, puesto que para llamar a un procedimiento se hace mediante la instrucción Call. Desde un procedimiento se puede invocar a su vez a otros procedimientos o funciones.

Un procedimiento almacenado se estructura de la siguiente forma, puede o no contener parámetros (Goette, 2011):

- Un nombre.
- Puede tener una lista de parámetros.
- Tiene un contenido (sección también llamada definición del procedimiento, aquí se especifica qué es lo que va a hacer y cómo).
- Ese contenido puede estar compuesto por instrucciones SQL, estructuras de control, declaración de variables locales, control de errores, etc.

La siguiente sintaxis muestra la estructura general y necesaria para crear un procedimiento almacenado:

```
CREATE PROCEDURE nombre (parámetro)
      [características] definición
```

Un procedimiento almacenado puede contener más de un parámetro (se separan con comas) o puede no haber ninguno (en este caso deben seguir presentes los paréntesis, aunque no haya ningún parámetro dentro, los parámetros

tienen la siguiente estructura: Modo, Nombre y Tipo, donde cada sección representa lo siguiente (Goette, 2011):

- Modo: Es opcional y puede ser IN (el valor por defecto, son los parámetros que el procedimiento recibirá), OUT (son los parámetros que el procedimiento podrá modificar) INOUT (mezcla de los dos anteriores).
- Nombre: Es el nombre del parámetro.
- Tipo: Es cualquier tipo de dato de los provistos por MySQL.
- Dentro de características es posible incluir comentarios o definir si el procedimiento obtendrá los mismos resultados ante entradas iguales, entre otras cosas.
- Definición: Es el cuerpo del procedimiento y está compuesto por el procedimiento en sí. Aquí se define qué hace, cómo lo hace y bajo qué circunstancias lo hace.

Comenta Goette (2011) que, así como existen procedimientos almacenados, también existen funciones. Para crear una función en MySQL se debe utilizar la directiva CREATE FUNCTION. La diferencia entre una función y un procedimiento es que la función devuelve valores. Estos valores pueden ser utilizados como argumentos para instrucciones SQL, tal como se hace normalmente con otras funciones como son, por ejemplo, MAX() o COUNT(). Utiliza la cláusula RETURNS y es obligatoria al momento de definir una función, sirve para especificar el tipo de dato que será devuelto (sólo el tipo de dato, no el dato).

La sintaxis para una función es:

```
CREATE FUNCTION nombre (parámetro)
RETURNS tipo
    [características] definición
```

Para una función puede haber más de un parámetro (se separan con comas) o puede no haber ninguno (en este caso deben seguir presentes los paréntesis, aunque no haya parámetros), las funciones tienen la siguiente estructura: Nombre y Tipo, donde:

- Nombre: Es el nombre del parámetro.
- Tipo: Es cualquier tipo de dato de los provistos por MySQL.
- Dentro de características es posible incluir comentarios o definir si la función devolverá los mismos resultados ante entradas iguales, entre otras cosas.
- Definición: Es el cuerpo del procedimiento y está compuesto por el procedimiento en sí, aquí se define qué hace, cómo lo hace y cuándo lo hace.

Para hacer el llamado a una función se hace invocando su nombre, como se hace en muchos lenguajes de programación, desde una función se puede invocar a su vez a otras funciones o procedimientos.

Ejemplo básico de un procedimiento almacenado (Goette, 2011).

```
mysql> delimiter //
mysql> CREATE PROCEDURE procedimiento (IN cod INT)
    -> BEGIN
    ->    SELECT * FROM tabla WHERE cod_t = cod;
    -> END
    -> //
mysql> delimiter ;
mysql> Call procedimento(4);
```

En el código anterior lo primero que se hace es fijar un delimitador (delimiter), para MySQL el delimitador por defecto es el punto y coma (;): en los procedimientos almacenados se puede definir por los usuarios.

El procedimiento recibe un parámetro para luego trabajar con él, el parámetro es de tipo IN, el parámetro de tipo OUT se define cuando en él se va a guardar la salida del procedimiento, si el parámetro hubiera sido de entrada y salida a la vez, sería de tipo denominado INOUT, el procedimiento almacenado termina y posteriormente es llamado con la siguiente instrucción:

```
mysql> Call procedimento(4);
```

Cabe destacar que esta es una forma de utilizar el delimitador (delimiter) sin que el procedimiento termine, puesto que también puede ser omitido y haberse

utilizado el delimitador clásico, en el mismo ejemplo visto anteriormente se presentará la sintaxis utilizando el delimitador clásico ubicando el contenido entre las palabras reservadas BEGIN y END.

Ejemplo de procedimiento almacenado con delimitador clásico, BEGIN y END:

```
CREATE PROCEDURE procedimiento2 (IN a INTEGER)
BEGIN
      DECLARE variable CHAR(20);
      IF a > 10 THEN
            SET variable = 'mayor a 10';
      ELSE
            SET variable = 'menor o igual a 10';
      END IF;
      INSERT INTO tabla VALUES (variable);
END
```

- El procedimiento recibe un parámetro llamado a que es de tipo entero.
- Se declara una variable para uso interno con el nombre variable y es de tipo char.
- Se implementa una estructura de control y si a es mayor a 10 se asigna a variable un valor y sí no lo es se le asigna otro.
- Se utiliza el valor final de variable en una instrucción SQL para la inserción.

El siguiente ejemplo muestra la sintaxis necesaria para una función:

```
mysql> delimiter //
mysql> CREATE FUNCTION cuadrado (s SMALLINT) RETURNS SMALLINT
    -> RETURN s*s;
    -> //
mysql> delimiter ;
mysql> SELECT cuadrado(2);
```

Ejemplos de procedimientos almacenados con parámetros, IN, OUT y INOUT.

IN: Es opcional colocarlo, puede no indicarse el tipo de argumento que se usará y por defecto se tomará como **IN** (entrada), por lo tanto, los valores que tomen

las variables de este tipo no se conservarán una vez termine la ejecución del procedimiento (CassiaNet, 2017).

```
DROP PROCEDURE IF EXISTS ejemplo;
DELIMITER //
CREATE PROCEDURE ejemplo(IN id INT)
BEGIN
      -- El parámetro es IN (solo de entrada), no se podrá
      -- acceder a su valor luego de que termine el
      -- procedimiento.
      -- Se concatenan dos campos y se muestra el resultado.
      SELECT CONCAT(apellido,' ',nombre) as Nombre_Completo
      FROM usuarios WHERE idusuario=id;
END//
DELIMITER ;
-- Mostrará el registro de id 2
Call ejemplo(2);
```

OUT: Los valores de este tipo de argumentos se establecen dentro del procedimiento y pueden ser accedidos más adelante, es decir, aun cuando la ejecución del procedimiento haya terminado (CassiaNet, 2017).

```
DROP PROCEDURE IF EXISTS ejemplo;
DELIMITER //
CREATE PROCEDURE ejemplo
(
   -- Recibe la variable de usuario donde se almacenará el resultado.
      OUT total INT
)
BEGIN
      -- Se calcula el número de registros y se almacenan
      -- en la variable de usuario
      SELECT COUNT(*) INTO total FROM usuarios;
END//
DELIMITER ;
-- Se llama el procedimiento y se indica donde se
-- almacenará el resultado (@mi_variable)
Call ejemplo(@mi_variable);
-- Ahora se puede acceder a @mi_variable cuando sea necesario
SELECT @mi_variable;
```

INOUT: Es una combinación de los dos anteriores, la variable puede recibir valores de entrada y puede ser accedida más adelante (CassiaNet, 2017).

```
DROP PROCEDURE IF EXISTS ejemplo;
DELIMITER //
CREATE PROCEDURE ejemplo
(
      -- Recibe la variable de usuario donde se almacenará
      -- el resultado.
      -- Puede trabajar con el valor recibido por parámetro.
      -- El valor actualizado de la variable puede ser
      -- accedida en cualquier momento.
      INOUT var INT
)
BEGIN
      SET var := var + 6;
END//
DELIMITER ;
-- Se inicializa la variable
SET @mi_variable := 4;
-- Se llama el procedimiento indicándole donde almacenará el
-- resultado (@mi_variable)
Call ejemplo(@mi_variable);
-- Ahora se puede acceder a @mi_variable cuando se requiera
-- Devuelve 10 ya que se actualizó su valor dentro del procedure
SELECT @mi_variable;
```

3.3.4. Conexión a base de datos de SQL Server mediante Visual Studio.

Microsoft SQL Server es un sistema para la gestión de bases de datos producido por Microsoft basado en el modelo relacional. Microsoft SQL Server constituye la alternativa de Microsoft a otros potentes sistemas gestores de bases de datos como son Oracle, PostgreSQL o MySQL.

Para poder realizar una conexión desde Visual Basic .Net, se debe de preparar la base de datos con un usuario y contraseña y asignar al usuario los privilegios

necesarios. Desde “SQL Server Management Studio” se puede realizar esta tarea de una manera fácil y sencilla.

En primer lugar, se deben realizar las importaciones de las clases que son necesarias para realizar la conexión a una base de datos SqlServer con VB.net:

Imports System.Data

Imports System.Data.OleDb

Imports System.Data.SqlClient

Se define la conexión a la base de datos con el nombre del servidor, la base de datos, usuario y contraseña que se han definido en sqlServer:

```
myConnectionString = "Provider=sqloledb;" & _
                     "Data Source=localhost;" & _
                     "Initial Catalog=bd-gaSQL;" & _
                     "User Id=UserPrueva;Password=contraseñaPrueva"
```

Se va realizar una consulta a la base de datos mediante un “Select”, la lectura de los datos de la consulta se lee con el OleDbDataReader:

```
Try
    Dim s As String = ("SELECT * FROM Alumnos")
    conexion = New OleDbConnection(myConnectionString)
    myCommand = New OleDbCommand(s)
    myCommand.Connection = conexion
    conexion.Open()
    Dim myReader As OleDbDataReader = myCommand.ExecuteReader()
        While myReader.Read()
            Dim NOM As String = myReader("NOM")
            Dim COGNOM As String = myReader("COGNOM")
        End While
    Catch exc As Exception
        Throw New GestorExcepcio(exc.Message)
  End Try
```

Para añadir un nuevo registro se hacer mediante “Insert”:

```
Try
```

```
    Dim sent As String = ("INSERT INTO Alumnos VALUES
('JORDI','NAVES')")
    Dim conexion As New OleDbConnection(myConnectionString)
    myCommand = New OleDbCommand(sent)
    myCommand.Connection = conexion
    conexion.Open()
    myCommand.ExecuteNonQuery()
    conexion.Close()
Catch exc As Exception
    Throw New GestorExcepcio(exc.Message)
End Try

Try
    Dim s As String = ("UPDATE Alumnos SET nom='JORDI' Where
id=1")
    conexion = New OleDbConnection(myConnectionString)
    myCommand = New OleDbCommand(s)
    myCommand.Connection = cnnexion
    conexion.Open()
    myCommand.ExecuteNonQuery()
    conexion.Close()
Catch exc As Exception
    Throw New GestorExcepcio(exc.Message)
End Try
```

Y finalmente el bloque anterior Try-Catch es utilizado para modificar un registro mediante un “update” (Solé, 2014).

3.3.5. Procedimientos almacenados en SQL Server.

Los procedimientos almacenados en SQL Server son rutinas de T-SQL (**Transact-SQL** es una extensión al SQL de Microsoft y Sybase. SQL, que frecuentemente se dice ser un Lenguaje de Búsquedas Estructurado por sus siglas en inglés). Consisten en un solo bloque de código que pueden ser ejecutados mediante el comando EXECUTE. Pueden recibir datos en sus parámetros de entrada y enviar nueva información a través de los parámetros de salida.

A diferencia de los procedimientos almacenados las funciones tienen ciertas limitaciones; no pueden efectuar ningún cambio en los datos (INSERT, DELETE, UPDATE), deben tener al menos un parámetro y devolver si o si un valor. Los procedimientos almacenados, pueden ser vistos como pequeños programas individuales dentro de nuestra base de datos. Estos “programas” pueden ser utilizados por aplicaciones externas o ser programados para que se ejecuten en el tiempo. Son más poderosos ya que sí pueden modificar datos, llamar a otros procedimientos almacenados, utilizar funciones y demás.

A continuación, se muestra la creación de un procedimiento almacenado que devuelva la cantidad de pedidos por cliente. Además, contiene la opción de dar la posibilidad al usuario del procedimiento de especificar un transportista en particular, si es que se necesita ese filtro (Demczuk, 2017):

```
IF OBJECT_ID('Ventas.PedidosPorCliente', 'P') IS NOT NULL
     DROP PROCEDURE Ventas.PedidosPorCliente;
GO

CREATE PROCEDURE Ventas.PedidosPorCliente
     @id_cliente INT,
     @id_transportista INT = NULL,
     @cantidad_pedidos INT OUTPUT
AS
IF @id_transportista IS NULL
BEGIN
SET @cantidad_pedidos = (SELECT COUNT(id_pedido)
     FROM Ventas.Pedidos
     WHERE id_cliente = @id_cliente);
END
ELSE
BEGIN
SET @cantidad_pedidos = (SELECT COUNT(id_pedido)
     FROM Ventas.Pedidos
     WHERE id_cliente = @id_cliente AND
           id_transportista = @id_transportista);
END
RETURN;
```

```
GO
```

De esta forma, primero se definen los parámetros que serán tomados como variables dentro del procedimiento. Cabe destacar que @cantidad_pedidos es un parámetro de salida, por lo que necesita estar acompañada de OUTPUT. Además, como @id_transportista es un parámetro donde el ingreso de un valor es opcional, debe tener asignado NULL por defecto.

Por lo tanto, si no se le asigna un valor en particular cuando el usuario ejecuta el procedimiento, devolverá los resultados para todos los transportistas de ese cliente. Por ejemplo, para ejecutar el procedimiento anterior se asigna un valor igual a 1 al parámetro @id_cliente, pero no se asigna ningún valor al parámetro @id_transportista. El resultado son todos los pedidos para el cliente con id igual a 1 (Demczuk, 2017):

```
DECLARE @pedidos INT
EXEC Ventas.PedidosPorCliente @id_cliente = 1,  @cantidad_pedidos
= @pedidos OUTPUT;
SELECT @pedidos AS CantidadPedidosCliente;
-- Resultado
CantidadPedidosCliente
---------------
5
```

En cambio, si esta vez se asigna un valor = 1 al parámetro @id_transportista, se puede apreciar como varía el resultado, limitando los pedidos del cliente con id 1 al transportista con id 1:

```
DECLARE @pedidos INT
EXEC Ventas.PedidosPorCliente @id_cliente = 1, @id_transportista =
1, @cantidad_pedidos = @pedidos OUTPUT;
SELECT @pedidos AS CantidadPedidosCliente;
-- Resultado
CantidadPedidosCliente
---------------
1
```

En este caso, no se ve la cantidad de pedidos total para el cliente con id = 1 si no que, al filtrar por transportista, se obtienen todos los pedidos entregados a este cliente por el transportista con id = 1.

En su artículo Demczuk (2017) dice que, insertar registros a tablas de una base de datos mediante procedimientos almacenados no es más que sentencias INSERT dentro de la declaración de un procedimiento.

```
CREATE PROCEDURE [Ventas].[InsertarTransportista]
      @Nombre_Empresa VARCHAR(55),
      @Direccion VARCHAR(55),
      @Ciudad VARCHAR(55),
      @Region VARCHAR(55),
      @Pais VARCHAR(55),
      @Telefono VARCHAR(55)
AS
BEGIN
      SET NOCOUNT ON
      INSERT INTO Ventas.Transportistas(nombre_empresa, direccion,
ciudad, region, pais, telefono)
      VALUES
      (
            NULLIF((RTRIM(LTRIM(@Nombre_Empresa))),''),
            NULLIF((RTRIM(LTRIM(@Direccion))),''),
            NULLIF((RTRIM(LTRIM(@Ciudad))),''),
            NULLIF((RTRIM(LTRIM(@Region))),''),
            ISNULL(NULLIF((RTRIM(LTRIM(@Pais))),''),'Argentina'),
            NULLIF((RTRIM(LTRIM(@Telefono))),'')
      );
END;
```

De tal forma, se definen parámetros de entrada al procedimiento, estos parámetros se llenarán con valores referentes a la información del transportista a insertar. Además, mediante las funciones NULLIF, RTRIM y LTRIM, se limpian caracteres vacíos que deja el usuario al escribir los valores, con la intención de almacenar de manera íntegra los datos, para utilizar este procedimiento se debe ejecutar la siguiente sintaxis:

```
EXECUTE [Ventas].[InsertarTransportista] @Nombre_Empresa = 'San
Fernando S.A', @Direccion = 'Avenida Pedro Goyena 1230', @Ciudad =
'Lomas de Zamora',
@Region = 'Buenos Aires', @Pais = 'Argentina', @Telefono = '4345-
2345'
```

El siguiente procedimiento almacenado se encarga de eliminar un transportista de la base de datos.

```
CREATE PROCEDURE [Ventas].[EliminarTransportista]
@Id_Transportista INT
AS
BEGIN
	SET NOCOUNT ON;
	DELETE FROM Ventas.Transportistas
	WHERE id_transportista = @Id_Transportista
END;
```

Para actualizar los datos de un transportista se muestras a continuación el siguiente procedimiento almacenado.

```
CREATE PROCEDURE [Ventas].[EditarTransportista]
	@Id_Transportista INT,
	@Nombre_Empresa VARCHAR(55),
	@Direccion VARCHAR(55),
	@Ciudad VARCHAR(55),
	@Region VARCHAR(55),
	@Pais VARCHAR(55),
	@Telefono VARCHAR(55)
AS
BEGIN
	SET NOCOUNT ON
	UPDATE Ventas.Transportistas
	SET nombre_BDEmpresa =
NULLIF((RTRIM(LTRIM(@Nombre_Empresa))),''),
	direccion = NULLIF((RTRIM(LTRIM(@Direccion))),''),
	ciudad = NULLIF((RTRIM(LTRIM(@Ciudad))),''),
	region = NULLIF((RTRIM(LTRIM(@Region))),''),
	pais = ISNULL(NULLIF((RTRIM(LTRIM(@Pais))),''),'Argentina'),
	telefono = NULLIF((RTRIM(LTRIM(@Telefono))),'')
	WHERE id_transportista = @Id_Transportista
```

```
END;
```

A diferencia de un procedimiento almacenado, una función puede retornar tanto valores escalares como tablas; es decir, puede simular algo parecido a una vista parametrizada.

Una función escalar retorna justamente un valor escalar (un valor único). Las funciones se pueden utilizar en todos los lugares donde se espere recibir un valor único.

Ejemplo de una función (Demczuk, 2017):

```
CREATE FUNCTION Ventas.TotalPorProducto(@Precio_Unitario MONEY,
@Cantidad INT)
RETURNS MONEY
AS
BEGIN
     RETURN @Precio_Unitario * @Cantidad AS Total
END;
GO
```

Una vez ejecutada la función en la base de datos, hay que hacer uso de ella y para ello hay que consultar una tabla que contenga pedidos con productos, precios y cantidades:

```
SELECT id_producto AS PRODUCTO, precio_unitario AS [PRECIO
UNITARIO],
cantidad AS CANTIDAD,
Ventas.TotalPorProducto(precio_unitario, cantidad) AS TOTAL
FROM Ventas.PedidoDetalles;
GO
```

De tal forma, se está mostrando una última columna computada a través de la función Ventas.TotalPorProducto. El resultado se muestra a continuación:

```
/*
PRODUCTO    PRECIO UNITARIO       CANTIDAD    TOTAL
----------- --------------------- ----------- --------------------
128         109,00                10          1090,00
150         17,00                 25          425,00
168         126,00                20          2520,00
```

```
179         31,10                   125         3887,50
184         28,50                   100         2850,00
219         220,00                  50          11000,00
263         65,00                   100         6500,00
114         121,00                  25          3025,00
175         82,50                   25          2062,50
193         85,51                   100         8551,00
157         156,20                  100         15620,00
*/
```

3.4. App.config.

App.config es un Archivo de configuración de Aplicaciones, que está formado por un conjunto de instrucciones en Xml (siglas en inglés de *eXtensible Markup Language*, traducido como "Lenguaje de Marcado Extensible" o "Lenguaje de Marcas Extensible", es un meta-lenguaje que permite definir lenguajes de marcas desarrollado por el *World Wide Web Consortium* (W3C) utilizado para almacenar datos en forma legible), que también se puede encriptar, lo cual da una ventaja de poder ocultar los parámetros de conexión al motor de bases de datos.

La incrustación de cadenas de conexión en el código de la aplicación puede producir vulnerabilidades en la seguridad y problemas de mantenimiento. Las cadenas de conexión sin cifrar compiladas en el código fuente de una aplicación se pueden visualizar. Además, si la cadena de conexión cambia en algún momento, será necesario compilar de nuevo la aplicación. Por estas razones, se recomienda almacenar las cadenas de conexión en un archivo de configuración de la aplicación (Microsoft, 2016).

Las cadenas de conexión se pueden almacenar como pares clave-valor en la sección connectionStrings del elemento configuration en el archivo de configuración de una aplicación. Los elementos secundarios incluyen add, clear y remove.

El siguiente fragmento del archivo de configuración muestra el esquema y la sintaxis para almacenar una cadena de conexión. El atributo name es un nombre

que se proporciona para identificar de forma única una cadena de conexión, de forma que se pueda recuperar en tiempo de ejecución. providerName es el nombre invariable del proveedor de datos .NET Framework registrado en el archivo machine.config (Microsoft, 2016).

```
<?xml version='1.0' encoding='utf-8'?>
<configuration>
    <connectionStrings>
        <add name="Name" providerName="System.Data.ProviderName"
            connectionString="Valid Connection String;" />
    </connectionStrings>
</configuration>
```

Los archivos de configuración externos son archivos independientes que contienen un fragmento de un archivo de configuración compuesto de una sola sección. El archivo de configuración principal hace referencia al archivo de configuración externo. El almacenamiento de la sección connectionStrings en un archivo físicamente independiente que resulta útil en situaciones en las que es posible que se edite la cadena de conexión después de implementar la aplicación.

Por ejemplo, si se modifican los archivos de configuración, ASP.NET reinicia de forma predeterminada el dominio de la aplicación, lo que provoca la pérdida de la información de estado. Sin embargo, la modificación de un archivo de configuración externo no provoca el reinicio de la aplicación.

Los archivos de configuración externos no se limitan a ASP.NET, también se pueden utilizar en aplicaciones Windows. Además, la seguridad y permisos de acceso a los archivos se pueden usar para restringir el acceso a los archivos de configuración externos. El trabajo con archivos de configuración externos en tiempo de ejecución es transparente y no requiere código especial.

Para almacenar cadenas de conexión en un archivo de configuración externo, se debe crear un archivo independiente que contenga únicamente la sección connectionStrings, no incluya elementos, secciones ni atributos adicionales. En este ejemplo se muestra la sintaxis de un archivo de configuración externo.

```
<connectionStrings>
```

```
    <add name="Name"
        providerName="System.Data.ProviderName"
        connectionString="Valid Connection String;" />
</connectionStrings>
```

En el archivo de configuración principal de la aplicación, usar el atributo configSource para especificar el nombre completo y la ubicación del archivo externo.

En este ejemplo se hace referencia a un archivo de configuración externo denominado connections.config (Microsoft, 2016).

```
<?xml version='1.0' encoding='utf-8'?>
<configuration>
    <connectionStrings configSource="connections.config"/>
</configuration>
```

3.5. ADO .NET.

ADO.NET es a veces considerado como una evolución de la tecnología ActiveX Data Objects (ADO), pero fue cambiado tan extensivamente que puede ser concebido como un producto enteramente nuevo.

ADO .NET es la tecnología propuesta por Microsoft para realizar la conectividad y gestión de datos. Se conforma por un conjunto de clases con funcionalidades específicas. Esas clases tienen el objetivo de ofrecer servicios de accesos de datos y componentes para el desarrollo de aplicaciones de uso compartido. El acceso se proporciona para bases de datos relacionales, XML, y aplicaciones.

ADO .NET permite realizar un acceso coherente a orígenes de datos SQL Server, XML, y de otras fuentes de datos a través de Object Linking and Embedding for Databases "Enlace e Incrustación de Objetos para Bases de Datos" en sus siglas (OLE DB) y es una tecnología desarrollada por Microsoft usada para tener acceso a diferentes fuentes de información, o bases de datos, de manera uniforme.

También permite realizar el acceso a orígenes de datos mediante Open DataBase Connectivity o lo que es lo mismo, conectividad abierta de bases de datos (ODBC), es un estándar de acceso a las bases de datos desarrollado por SQL

Access Group (SAG) en 1992. El objetivo de ODBC es hacer posible el acceder a cualquier dato desde cualquier aplicación, sin importar qué Sistema de Gestión de Bases de Datos (DBMS) almacene los datos.

Todas aquellas aplicaciones que consuman datos pueden utilizar ADO .NET para conectarse a los orígenes de datos pertinentes, recuperar esos datos, controlar la gestión de los mismos y actualizarlos adecuadamente en sus fuentes.

ADO .NET puede separar el acceso de los datos de su manipulación, creando componentes que se pueden utilizar de forma separada o en conjunto. Los datos se pueden procesar directamente en su fuente o también manejarse de manera desconectada en la memoria a través de DataSet. Las clases que conforman ADO. NET se encuentran implementadas en el espacio o librerías con nombres System.Data.dll y se integran completamente con las clases que se incluyen para XML y que se encuentran en System.Xml.dll.

ADO .NET es el modelo que utiliza .NET para el acceso a los orígenes de datos como SQL Server y XML, en forma nativa y a otros orígenes a través de OLE DB y ODBC. Cabe destacar que ADO .NET separa el acceso a los datos de la manipulación de los mismos. Incluye proveedores de datos que se conectan a una base de datos, se ejecutan comandos o se recuperan resultados en objetos DataSet.

Entre las ventajas de ADO .NET se destacan las siguientes:

- Soporte nativo para XML y la posibilidad de serializar objetos a XML.
- Se cuenta con el comando For Each que permite recorrer los elementos de un DataSet sin necesidad de utilizar el comando MoveNext.
- Es posible especificar que un DataAdapter realice una consulta sobre distintas bases de datos al mismo tiempo.
- Se puede trabajar desconectados de bases de datos, creándose una vista relacional en la memoria en el equipo del cliente.

En ADO .NET destacan dos objetos: DataSet y DataReader. Por lo tanto, estos objetos han sido diseñados para el acceso a la información almacenada en

las bases de datos. En el caso de un DataSet, se ha diseñado para el acceso a datos de manera independiente al origen de datos. Por tanto, se puede utilizar con múltiples y distintos orígenes de datos, con XML, o para administrar datos locales de una aplicación.

Un DataSet posee una colección de uno o más objetos DataTable, los cuales están formados por filas y por columnas de datos, así como también información sobre claves principales, externas, restricciones y relaciones entre objetos.

En conclusión, un DataSet es como una base de datos virtual que se carga en la memoria siendo como una copia fiel de la que radica en el servidor. Las relaciones de integridad referencial y reglas de validación se conservan en el DataSet.

Por otra parte los objetos DataReader se utilizan para leer datos desde una fuente, cargarlos a memoria y desconectarse de la fuente cuando ya los cargó. Proporcionan acceso únicamente hacia adelante (forward-only) y son de sólo lectura (read-only).

El objeto DataSet se mantiene desconectado de la fuente mientras que el DataReader se mantiene conectado mientras realiza la lectura de los datos. El acceso de DataReader es mucho más rápido que el de un DataSet.

Se debe utilizar DataSet o DataReader en los siguientes casos:

- Si se requiere lectura y escritura hacia y desde una fuente de datos (los DataReader son sólo de lectura).
- Si se requiere combinar tablas de una o más bases de datos, los DataReader lo hacen de una sola base de datos a través de una instrucción SQL.
- Si se requiere que un formulario web se enlacen los datos a más de un control, los DataReader lo hacen a un único control.
- Si se requiere que lo datos permanezcan desconectados, los DataReader mantienen conexión a la base de datos.

- Si se requiere realizar búsquedas en cualquier sentido en una colección de datos (hacia adelante y hacia atrás de los registros), el DataReader sólo realiza búsquedas hacia adelante.
- Si se requiere una consulta muy rápida se utiliza el DataReader, el DataSet es mucho más lento que un objeto DataReader para realizar una consulta. El DataSet almacena los datos en memoria lo que lo hace más lento y no establece conexión directa con el origen de datos.

Otro objeto importante es el objeto DataAdapter, este objeto se utiliza para rellenar un DataSet y realizar actualizaciones en el origen de datos (Jiménez, 2010).

3.5.1. Arquitectura de ADO .NET.

Tradicionalmente, el procesamiento de datos a dependido principalmente de un modelo de dos niveles basado en una conexión. A medida que aumenta el uso que hace el procesamiento de datos de arquitecturas de varios niveles, los programadores están pasando a un enfoque sin conexión con el fin de proporcionar una mejor escalabilidad a sus aplicaciones, ver Figura 8 (Jiménez, 2010).

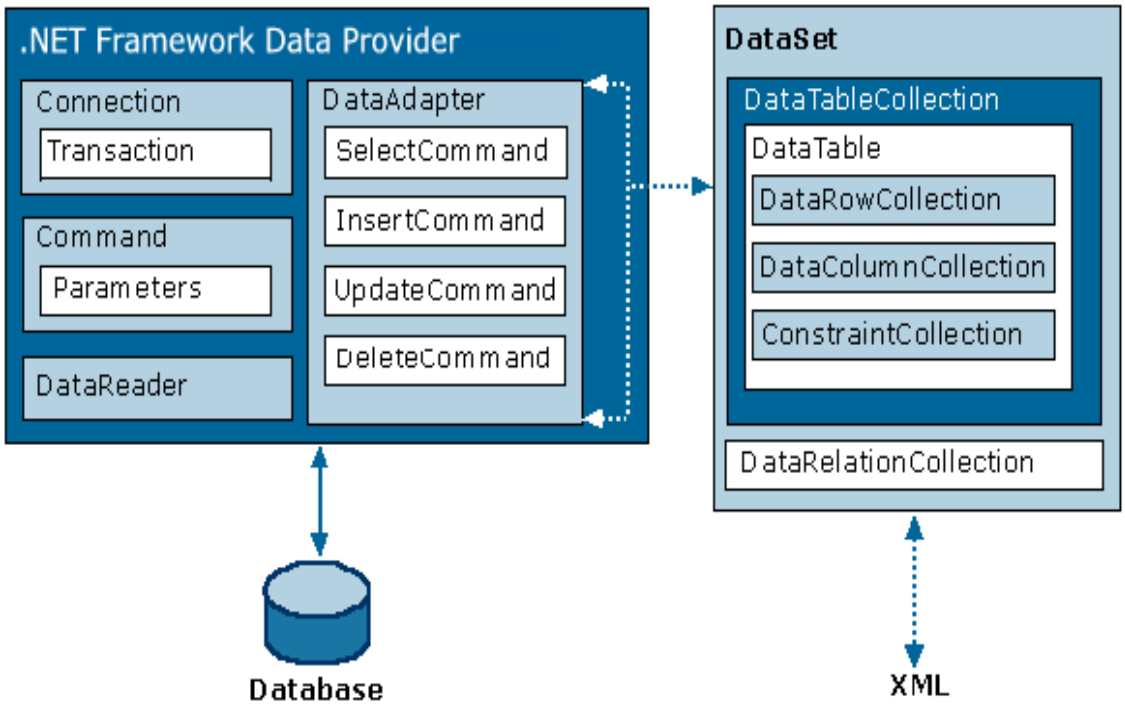


Figura 8. Arquitectura de ADO .NET (Jiménez, 2010).

3.5.2. Proveedores de ADO .NET.

Los proveedores de ADO .NET van a permitir realizar la conexión a bases de datos en distintos entornos, ejecutar comandos y recuperar resultados para la manipulación de información. Estos resultados que se generan se pueden almacenar localmente en un DataSet para que los usuarios puedan interactuar con ellos de una forma rápida y confiable. Estos datos cargados en DataSet trabajan en forma desconectada del origen de datos. Por los tanto también se pueden recuperar datos en forma conectada con objetos de tipo DataReader.

Entre los diversos proveedores de datos para ADO .NET se tienen los siguientes:

- **System.Data.SqlClient:** Proporciona acceso a datos nativos para Microsoft SQL Server de la versión 7.0 en adelante.
- **System.Data.OleDb:** Proporciona acceso a datos que se exponen mediante OLEDB. Microsoft Access es un ejemplo de accesibilidad y conectividad mediante OLEDB.
- **System.Data.Odbc:** Proporciona acceso a datos que se exponen mediante ODBC.
- **System.Data.OracleClient:** Proporciona acceso a datos que se originan en Oracle.
- **System.Data.EntityClient:** Proporciona acceso a datos para las aplicaciones que utilizan Entity Data Model o modelo de datos de entidad (EDM). Es una especificación que permite definir los datos que usan las aplicaciones integradas en Entity Framework. Las aplicaciones que usan el EDM definen las entidades y relaciones del dominio de la aplicación en un esquema de diseño.

Los proveedores de ADO .NET poseen cuatro clases principales que permiten brindar funcionalidad básica de los mismos en el .NET Framework.

- **DbConnection:** Sirve para establecer la conexión a un origen de datos determinado.

- **DbCommand:** Se utiliza para ejecutar un comando SQL en un origen de datos. Pueden utilizarse parámetros en conjunción con un objeto Transaction.
- **DbDataReader:** A través de este objeto se pueden leer datos de un sólo avance y lectura desde un origen de datos. Trabajan en un modo conectado al origen de datos.
- **DbDataAdapter:** Sirve como conector a un DataSet al cual rellena de datos según las condiciones de conexión y el objeto de consulta (textual, procedimiento almacenado, etc.)

ADO .NET también posee otras clases que son importantes para la gestión de datos de una aplicación .NET, las clases se presentan a continuación (Jiménez, 2010):

- **DbTransaction:** Con este objeto se pueden realizar transacciones en el origen de datos. Esta clase incluye los comandos necesarios para ello. Utiliza el espacio de nombres System.Transactions.
- **DbCommandBuilder:** Actúa como un objeto auxiliar, generando automáticamente las propiedades de comando de un DataAdapter. También puede obtener de un procedimiento almacenado información acerca de los parámetros con que puede rellenar la colección parameters de un objeto Command.
- **DbConnectionStringBuilder:** Igual que el CommandBuilder, este objeto proporciona la funcionalidad para crear y administrar el contenido de las cadenas de conexión utilizadas por el objeto connection.
- **DbParameter:** Mediante este objeto se puede definir los parámetros, sean de entrada, salida o ambos, y los valores devueltos para los comandos y procedimientos almacenados.
- **DbException:** Permite devolver una excepción cuando se produce un error en el origen de datos. También cuando el error se detecta en el cliente.
- **DbError:** Permite obtener la información relacionada con una advertencia o un error que se produce en un origen de datos.

- **DbClientPermission:** Es proporcionado para los atributos de seguridad de acceso al código por los proveedores de datos del .NET Framework.

3.5.3. Proveedores de datos para SQL Server (SqlClient).

El proveedor de acceso a datos definido para SQL Server, desde la versión 7.0 en adelante, se encuentra implementado en el espacio de nombres System.SqlClient. Este proveedor utiliza su propio protocolo de comunicación con SQL Server y de esta forma brindar un acceso directo, sin capas intermedias y mejor rendimiento.

Para utilizar la funcionalidad de la gestión de datos de SQL Server en una aplicación, por ejemplo, en Visual Basic .NET, se debe incluir el espacio de nombres correspondiente. Por lo tanto, en la siguiente línea se presenta cómo se incluye en la cabecera de un formulario o en una biblioteca de clases la referencia a este espacio de nombres.

Imports System.Data.SqlClient

Mientras que para utilizar una aplicación con Access de Microsoft se utiliza la funcionalidad de OLE DB, debe incluirse en la aplicación el espacio de nombres System.Data.OleDb. La siguiente línea muestra cómo debe incluirse la referencia.

Imports System.Data.OleDb

En cuanto al uso de un proveedor de datos para ODBC, éste utiliza el administrador de controladores de ODBC nativos para habilitar el acceso a datos. Puede utilizarse un ODBC para conectarse a una base de datos de ORACLE o Access, por ejemplo. EL espacio de nombres que se requiere para la gestión de datos es System.Data.Odbc. La siguiente línea muestra cómo incluir este espacio de nombres en una aplicación .NET.

Imports System.Data.Odbc

Cabe agregar que para Oracle posee su propio proveedor de datos para ADO .NET (OracleClient). Para hacer uso de su funcionalidad se requiere que el software del cliente de Oracle (versión 8.1.7 en adelante) esté instalado en el sistema antes

de conectarse a un origen de datos Oracle. En cuanto a las clases que le dan funcionalidad a este proveedor están integradas en el espacio de nombres System.Data.OracleClient. Las siguientes líneas habilitan el uso de la funcionalidad de este proveedor de datos.

Imports System.Data.OracleClient

EntityClient es un proveedor de datos que permite acceso a los mismos a través de un modelo conceptual denominado Entity Data Model. En realidad, este proveedor no interactúa directamente con ningún origen de datos. En su lugar se utiliza Entity SQL para comunicarse con el proveedor de datos subyacente (Jiménez, 2010).

3.5.4. DataSet en ADO .NET

Un objeto DataSet brinda la funcionalidad requerida para escenarios de datos distribuidos y desconectados. El objeto DataSet es una representación de los datos consumidos en memoria y proporciona un modelo programático relacional coherente e independiente del origen de datos. Por ende, se puede vincular con diferentes orígenes de datos incluyendo formatos XML.

Incluye las tablas de una base de datos, sus relaciones y restricciones, permite la integridad entre la fuente de datos y los datos que son cargados en memoria, ver la Figura 9.

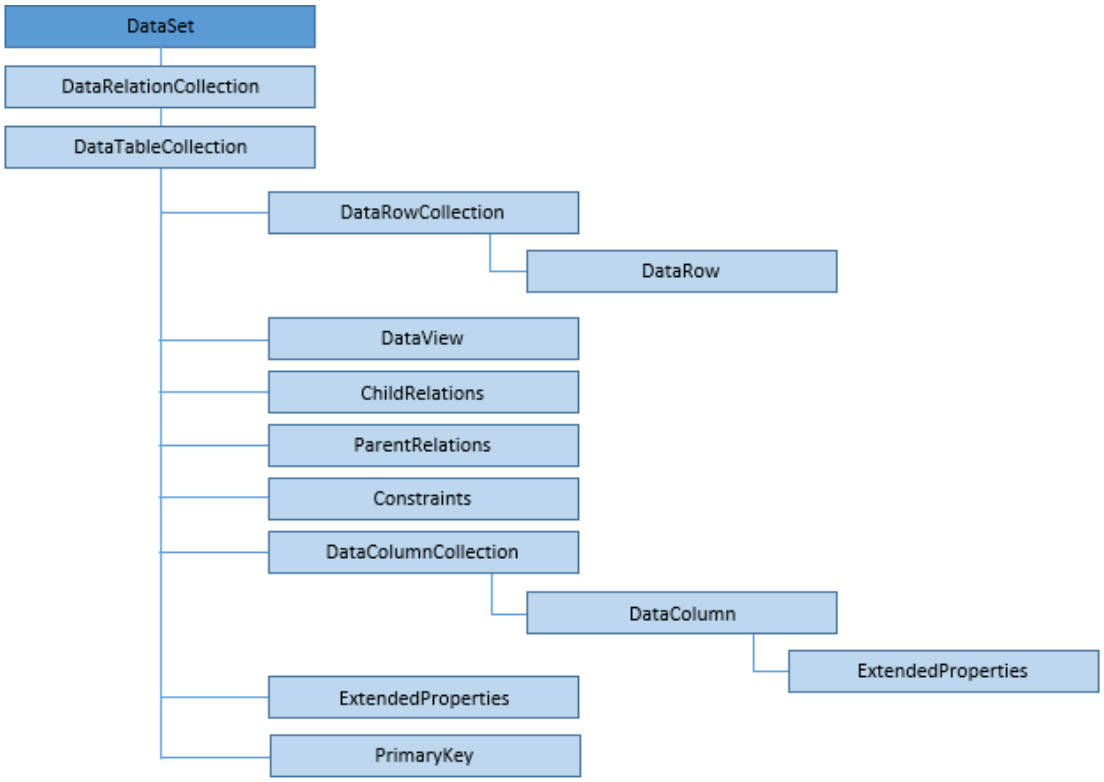


Figura 9. El modelo de objetos del DataSet (Jiménez, 2010).

Un DataSet es una caché de memoria interna con datos recuperados de una fuente previamente conectada. Está compuesta, según la figura anterior, de una colección de objetos DataTable que se pueden relacionar mediante objetos DataRelation. Se pueden administrar aspectos de integridad de los datos mediante objetos UniqueConstraint y ForeignKeyConstraint.

La clase DataTable se define en el espacio de nombres System.Data y representa una única tabla de datos residente en memoria. Internamente posee una colección de columnas representadas mediante DataColumnCollection, así como las restricciones representadas por ConstraintCollection. En conjunto representan el esquema de la tabla. Una clase DataTable contiene además una colección de filas que se representan mediante DataRowCollection que es quién contiene los datos de la tabla. Un DataRow conserva las versiones actuales y originales de los datos para determinar en todo momento los cambios que se presenten en los datos.

Mediante la clase DataView se pueden crear diferentes vistas de los datos que se encuentran almacenados en una clase DataTable. Esta capacidad suele utilizarse en aplicaciones orientadas a enlaces de datos, dado que se pueden exponer los datos de una tabla según diversos criterios de ordenación y filtrado.

Las relaciones que se establecen en un DataSet son gestionadas a través del objeto DataReationCollection. Las relaciones se representan mediante el objeto DataRelation, las cuales son asociaciones de filas de una DataTable con otras DataTable. Es una analogía de las relaciones que se establecen en un diagrama relacional de bases de datos. Mediante las relaciones establecidas en un DataSet se permite la navegación entre las clases DataTable.

Para crear y mantener actualizados los datos originales de un DataSet, es necesario realizar los siguientes pasos (Jiménez, 2010):

- A través de un DataAdapter, el cual establece la conexión con el origen de datos, se construyen y llenan los datos de cada DataTable.
- Modificar los datos de los objetos DataTable individuales mediante la inserción, modificación o eliminación de objetos DataRow.
- Hacer un llamado al método GetChanges y crear un segundo DataSet con los datos modificados.
- Invocar al método Update del DataAdapter pasándole como parámetro el segundo DataSet (el que contiene los datos modificados).
- Invocar el método Marge para sincronizar los datos del segundo DataSet con los del primero.
- Invocar el método AcceptChanges del DataSet para aceptar los cambios o RejectChanges para cancelarlos.

3.5.5. DataAdapter en ADO .NET.

Un DataAdapter se utiliza para recuperar datos de un origen de datos y llenar tablas con un DataSet. DataAdapter también resuelve los cambios realizados en DataSet de vuelta al origen de datos. Mediante el objeto Connection del proveedor de datos .NET Framework, DataAdapter se conecta a un origen de datos

y utiliza objetos Command para recuperar datos del origen de datos y resolver los cambios a dicho origen. En la Figura 10 se muestra la manera en que un objeto DataAdapter permite la conexión de los datos contenidos en un DataSet.

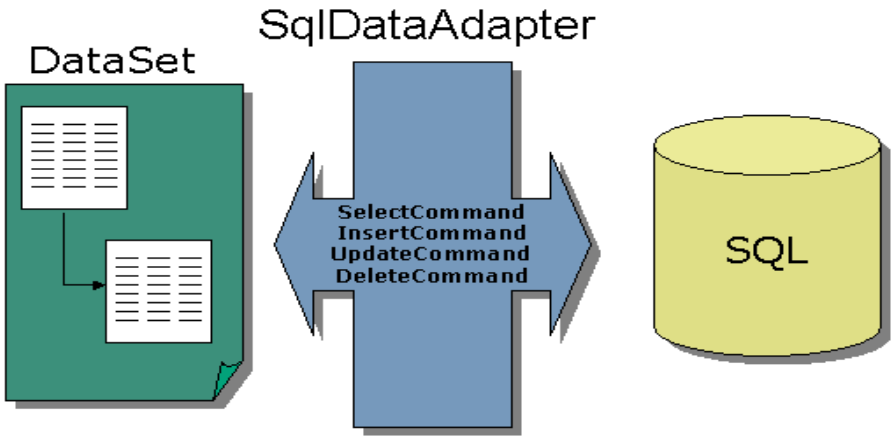


Figura 10. Conectividad de un DataSet mediante un DataAdapter (Jiménez, 2010).

De igual forma también existen otras funcionalidades importantes con el DataAdapter. Por ejemplo, las acciones que se pueden ejecutar sobre los datos que referencia el DataAdapter.

Las propiedades del DataAdapter que permiten estas acciones son las siguientes (Jiménez, 2010):

SelectCommand: Permite recuperar filas de la fuente de datos según la instrucción Select.

InsertCommand: Permite escribir filas insertadas en un DataSet en la respectiva fuente de datos referenciada por el DataAdapter, quien es el que los relaciona.

UpdateCommand: Este comando escribe filas modificadas en el DataSet a la base de datos referenciada por el DataAdapter que usa el DataSet.

DeleteCommand: Elimina filas en el DataSet a la base de datos referenciada por el DataAdapter que usa el DataSet.

En la Figura 11 se muestra la funcionalidad de estas propiedades entre los modelos DataSet, DataAdapter y el origen de datos.

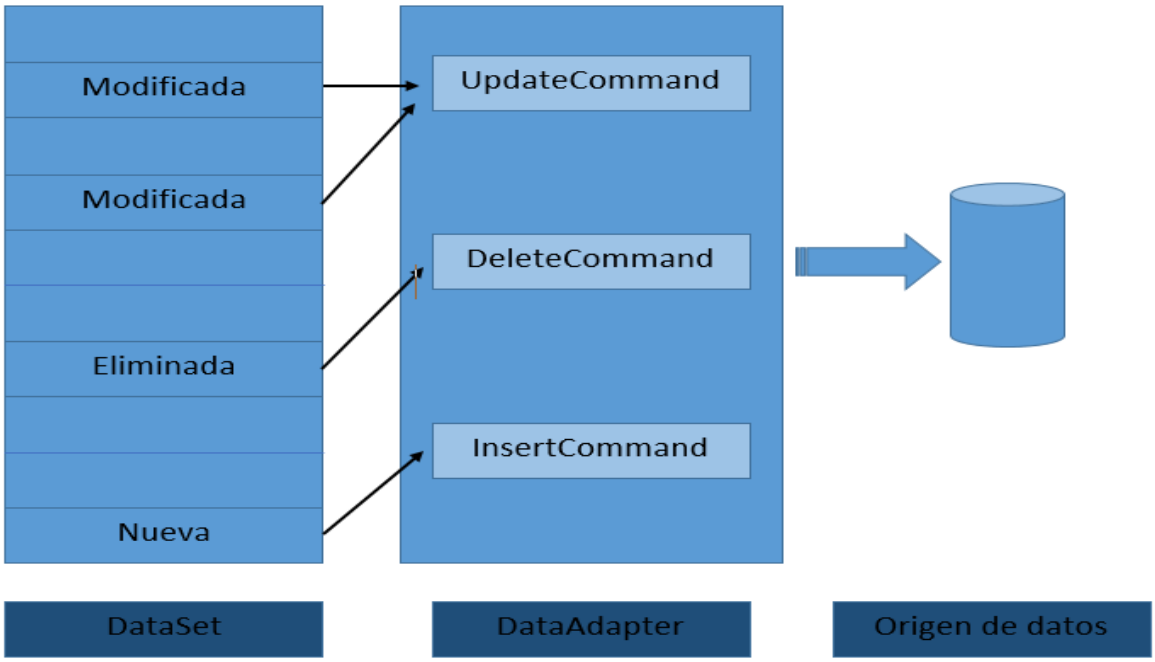


Figura 11. Propiedades del DataAdapter (Jiménez, 2010).

Observar que el DataSet es un conjunto de datos almacenados en memoria, los cuales son marcados con una bandera que indica si el dato fue modificado, eliminado o es una nueva inserción. Cuando el DataAdapter realiza la sincronización con la base de datos física (indicada en el origen) detecta los cambios en el DataSet y procede a ejecutar, sea UpdateCommand, DeleteCommand o InsertCommand, según corresponda (Jiménez, 2010).

En seguida se muestra un trozo de código que demuestra el uso de la clase DataSet conjuntamente con la clase DataAdapter. También están incluidos los comentarios respectivos.

```
'Se declara una variable de tipo DataSet (ds)
Dim ds As New DataSet
'Se establece la conexión a la base de datos.
Dim cn As String = “data source=localhost; initial
catalog=dbInventario; integrated security=true”
'Se implementa el DataAdapter conectándolo a la base de datos y
extrayendo la información.
Dim da As New SqlClient.SqlDataAdapter(“Select * from
tbProveedores”, cn)
```

```
'Se carga el DataSet (ds) con los datos consumidos por el
DataAdapter (da).
da.Fill(ds)
```

3.5.6. DataReader en ADO .NET.

El objeto DataSet proporciona una copia desconectada de la base de datos de manera íntegra. La mejor opción para gestionar datos de larga duración en una aplicación es mediante el uso de DataSet. Sin embargo, muchas veces solo se requiere realizar pequeñas operaciones de datos como por ejemplo autenticarse (identificarse a la hora de iniciar sesión) en sistemas donde solamente se utilizan los datos de usuario y contraseña. En este caso, es mejor hacer uso de un DataReader.

Hay ocasiones en que se necesita acceder a una gran cantidad de registros de una base de datos. Suponer que se requiere leer unos 5000 registros de una base de datos y se deben de almacenar en un objeto DataTable. Se requerirá memoria para que el DataTable almacene esa cantidad de registros.

Si se conectan 100 usuarios y realizan la misma operación se necesitará la cantidad de memoria requerida por los 5000 registros para los 100 usuarios conectados. Esto es un factor crítico de uso de memoria. En este tipo de situaciones, el objeto DataReader, el cual está diseñado para generar un flujo de sólo lectura (Read Only) y sólo hacia adelante (Forward Only). Por tanto, únicamente se almacena 1 registro en memoria cada vez en el servidor.

Esta diferencia radica en que el objeto DataSet está sobrecargado por la capacidad que tienen que leer y escribir datos, y de examinar hacia adelante y hacia atrás.

El DataReader, por su parte, solamente lee y recorre los registros hacia adelante evitando la escritura.

ADO .NET incluye dos tipos de objetos DataReader: sqlDataReader y OleDbDataReader. El primero es proveedor de datos de exclusivamente para la gestión de datos de SQL Server y por su parte OleDbDataReader es para cualquier

proveedor de datos. De tal manera los objetos SqlCommand y OleDbCommand, conjuntamente con el método Execute-Reader, sirven para transferir datos a un objeto DataReader.

Por consiguiente, un objeto DataReader tiene las siguientes características:

- La conexión de un DataReader a una fuente de datos debe gestionarse por si misma. Es decir, la conexión a una fuente de datos debe realizarse en forma manual, a diferencia del DataAdapter que abre y cierra una conexión automáticamente.
- La navegación entre registros debe realizarse programáticamente en el DataReader. También se puede realizar mediante vinculación de un control enlazado a lista. En ambos casos se debe escribir el código necesario para que se realice la operación.
- Al estar los datos cargados en memoria, el DataReader consume menos recursos del servidor.

Para realizar un objeto DataReader se debe considerar los siguientes pasos:

1. Crear una conexión a la fuente de datos con el objeto requerido. Para una base de datos SQL Server sería un objeto SQLConnection y para otras OleDbConnection.
2. Abrir la conexión a la fuente de datos.
3. Crear un objeto Command en lugar del DataAdapter
4. Crear un objeto DataReader desde el objeto Command.
5. Invocar al objeto ExecuteReader.
6. Utilizar el objeto DataReader.
7. Cerrar el objeto DataReader.
8. Cerrar la conexión a la fuente de datos.

Se puede utilizar el controlador de excepciones Try… Catch… Finally como aglutinador de los pasos anteriores para evitar cualquier detención del programa durante la posible generación de algún error en el proceso de ejecución.

A continuación se presenta un trozo de código, de un ejemplo de manejo del objeto DataReader.

```
Dim strConn As String = “data source = localhost; initial catalog
= dbInventario; integrated security = true”
Dim objConn As New SqlConnection(strConn)
Dim cmdTbProveedores As New Data.SqlClient.SqlCommand(“Select *
from TbProveedores”, objConn)
Try
    objConn.Open
    Dim dr As sqlDataReader
    dr = cmdTbProveedores.ExecuteReader()
    Do While dr.Read()
        TextBox1.Text = dr.Items(“IdProveedor”)
        TextBox2.Text = dr.Items(“Nombre”)
    Loop
    dr.Close()
    objConn.Close()
Catch (e As Exception)
    MsgBox(e.message)
End Try
```

De tal forma, en el bloque de código anterior un DataReader que mediante un objeto Command se ejecuta y se cargan los datos programáticamente en controles de tipo TextBox. En el siguiente bloque de código se puede apreciar cómo se vincula un objeto DataReader a un objeto lista.

```
Dim strConn As String = “data source = localhost; initial catalog
= dbInventario; integrated security = true”
Dim objConn As New SqlConnection(strConn)
Dim cmdTbProveedores As New SqlCommand(“Select * from
TbProveedores”, objConn)
Dim dr As SqlDataReader
Try
    objConn.Open
    dr.cmdTbProveedores.ExecuteReader()
    lstTbProveedores.DataSource = dr
    lstTbProveedores.DataTextField = “Nombre”
    lstTbProveedores.DataBind()
```

```
    dr.Close()
    objConn.Close()
Catch (e As Exception)
    MsgBox(e.message)
End Try
```

Como se ha observado en el código anterior se vincula un objeto de tipo DataReader a un objeto ListBox y le almacena los valores contenidos en uno de sus atributos (Jiménez, 2010).

3.5.7. Instrucción Using.

Según Remon (2012), se define como un bloque el cual permite delimitar el control de un recurso ADO.NET, es decir, un objeto se podrá crear, aperturar y cerrar desde el uso de un bloque Using. Técnicamente esta instrucción garantiza la eliminación de uno o más recursos cuando su código termina de usarlos. De esta forma esto libera y pone a disposición de otro código para que los pueda usar.

Los recursos administrados se eliminan mediante el recolector de elementos no utilizados de .NET Framework sin necesidad de código adicional por su parte. Para los recursos administrados no es necesario utilizar ningún bloque Using. Sin embargo, se puede utilizar un bloque Using para forzar la eliminación de un recurso administrado en lugar de esperar al recolector de elementos no utilizados.

Este bloque se comporta como una construcción Try…Finally, en la que el bloque Try utiliza los recursos y el bloque Finally los desecha. Gracias a esta instrucción, el uso de un bloque Using garantiza la eliminación de los recursos, independientemente de cómo se salga del mismo.

Sintaxis general del bloque Using:

```
Using {Expresion de Origen}
     [Expresiones o Instrucciones]
End Using
```

Expresión de origen: Es obligatorio especificar un recurso que controle el bloque Using, un recurso tiene la siguiente estructura:

nombreObjeto As New nombreClase (Argumentos opcionales)

Expresiones o instrucciones: Es el bloque de instrucciones que se ejecuta dentro del marco de un objeto usando Using.

End Using: Finaliza el bloque y cierra el recurso controlado por Using.

A continuación se muestra un ejemplo del uso de un bloque Using para crear el objeto cn que establece la conexión a una base de datos llamada Agencia del servidor local (Remon, 2012).

```
Using cn As New SqlConnection("SERVER = .; DATABASE = Agencia;
INTEGRATED SECURITY = SSPI")
     ....
End Using
```

Capítulo 4. Metodología.

4.1. Codificación de módulos.

De acuerdo con los resultados obtenidos del análisis y diseño del sistema, se generaron los siguientes módulos programables que se detallan en la Figura 12.

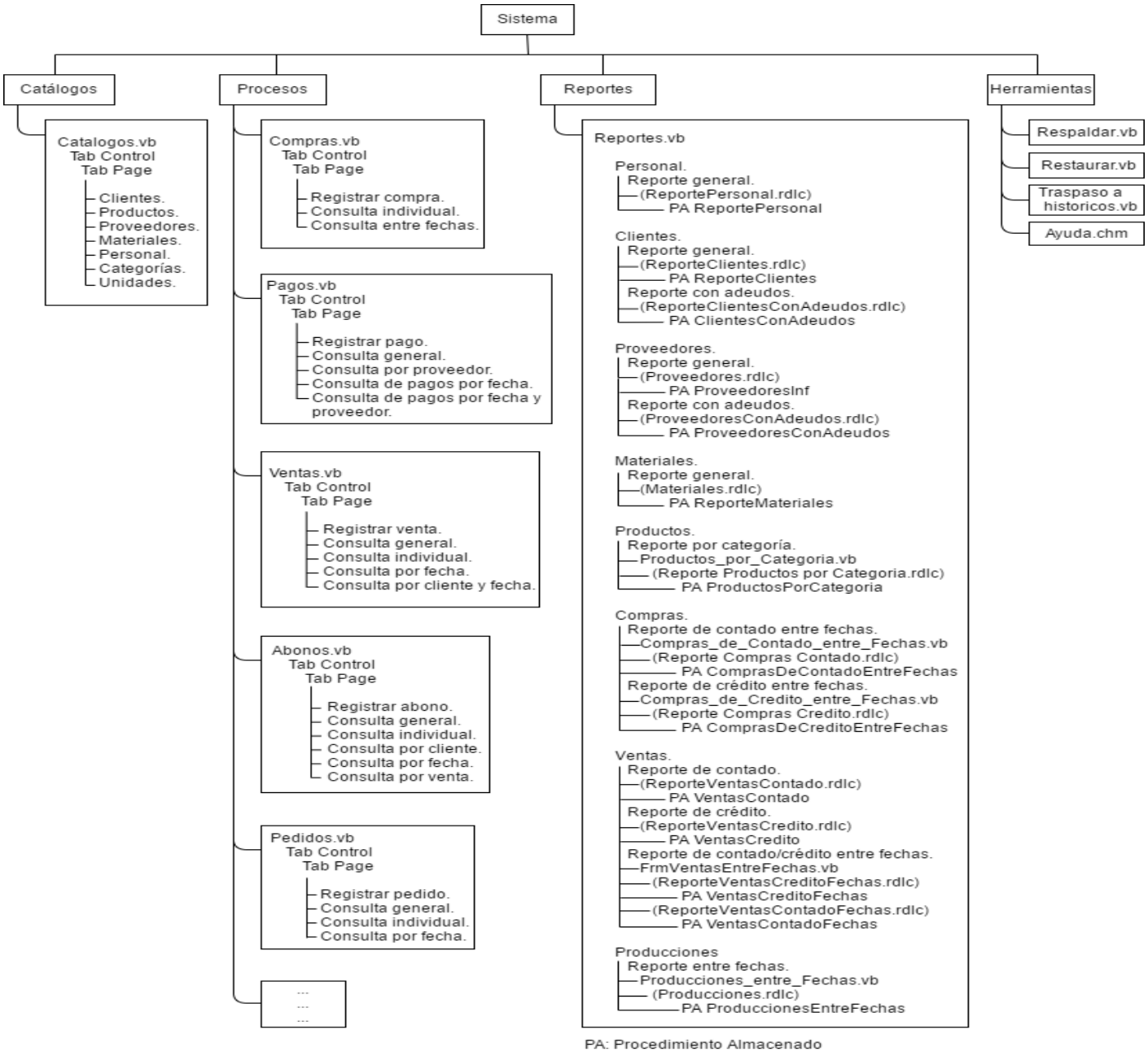


Figura 12. Diagrama general de codificación. Parte 1.

En la Figura 13 se muestra la segunda parte del diagrama general de codificación.

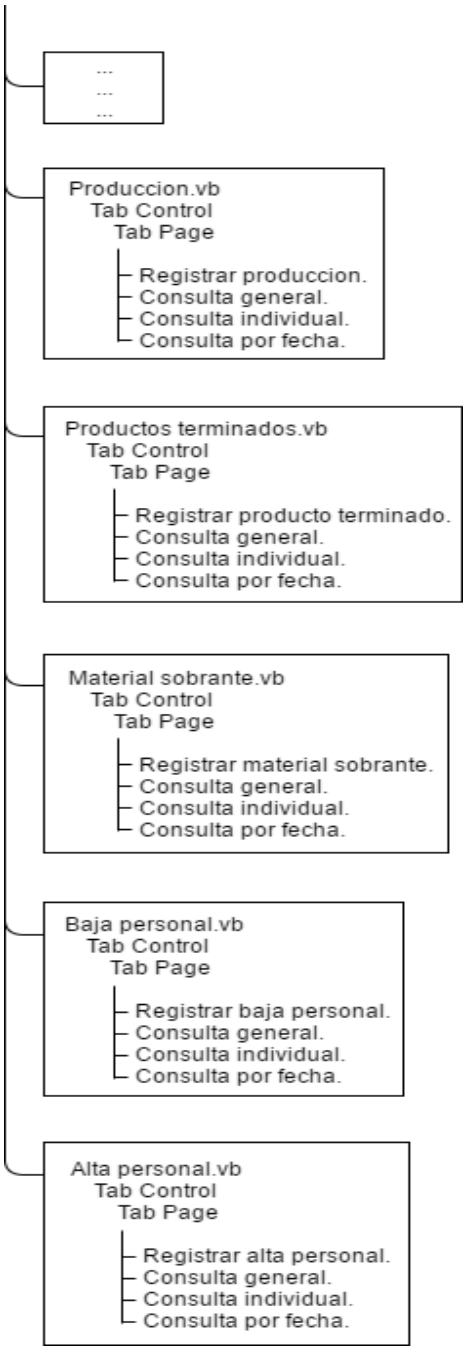


Figura 13. Diagrama general de codificación. Parte 2.

4.2. Pruebas.

4.2.1 Caso de prueba: Registrar venta.

Procedimiento a seguir para registrar correctamente una venta.

1. De la pantalla de inicio o menú principal seleccionar <**Procesos**>.
2. De la interfaz de procesos, seleccionar <**Ventas**>.
3. En la primera pestaña <**Registrar venta**> seleccionar los datos correspondientes.
4. En **Datos del cliente**, seleccionar de la comboBox el cliente indicado.
5. En **Datos del personal**, seleccionar de la comboBox el personal que atenderá dicha venta.
6. En el apartado de **Productos** seleccionar los productos correspondientes para la venta.
7. En **Agregar a detalle** proporcionar la cantidad de productos y posteriormente dar clic en el botón.
8. Seleccionar el tipo de venta en **Condición**, si es a crédito seleccionar la fecha de vencimiento.

En la Figura 14 se muestra la pantalla principal del proceso de ventas.

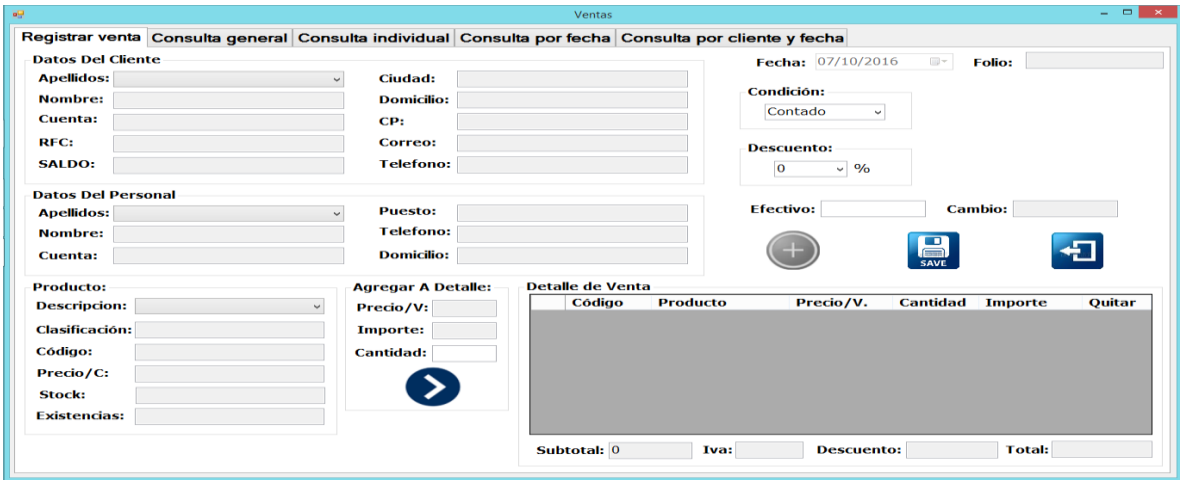


Figura 14. Pantalla principal del proceso de ventas.

Posteriormente se realiza una venta con los siguientes datos:

Entradas:

Cliente: Natalia Reyes Godínez

Personal: Raúl Hernández.

Existencias de productos.

En la Figura 15 se muestran los productos a registrar en la venta, remarcando el código y la existencia de cada uno de ellos.

Código	Descripción	Categoría	Precio costo	Precio venta	Existencias	Stock	Imagen
100	Mesa de sala	Cocina	540.00	600.00	28	5	
101	Mesa de cocina	Sala	600.00	700.00	50	10	
102	Ropero	Recamara	700.00	750.00	30	25	
103	Sofa de Piel	Sala	300.00	400.00	60	7	
104	Silla madera	Otros	240.00	260.00	250	10	
105	Cama matrimonial	Recamara	2800.00	3000.00	65	12	
106	Banco mediano	Cocina	340.00	360.00	40	25	
107	Buró chico	Sala	620.00	680.00	40	10	
108	Cama individual	Recamara	1200.00	1400.00	320	8	

Figura 15. Productos a registrar en la venta.

En la Figura 16 se muestra el detalle de venta a registrar con los siguientes productos: 2 roperos, 8 sillas de madera y 2 camas matrimoniales.

Detalle de Venta

Código	Producto	Precio/V.	Cantidad	Importe	Quitar
102	Ropero	750.00	2	1500	DELETE
104	Silla madera	260.00	8	2080	DELETE
105	Cama matrimonial	3000.00	2	6000	DELETE

Figura 16. Detalle de la venta a registrar.

La condición de venta es: Crédito al (16/02/2017).

Nota: No hay ninguna venta registra, por lo cual esta es la primera venta registrada, el IVA se calcula en base al 16%.

Salidas:

Venta registrada: Folio = 1.

Saldo del cliente: $ 11,112.80.

En la Figura 17 se muestran los clientes registrados, remarcando el ID y el Saldo del cliente al cual se registró la venta a crédito.

ID	Nombre	Apellidos	RFC	Teléfono	Ciudad	Domicilio	Cód. Postal	Email	Saldo
10	Mario	Castellanos	SDYR78E	578-456-6561	Guadalajara	Desconocido	48752	m1234567890@live...	0.00
20	Karla Aguirre	Jimenez Chavez	JCKA343DSD	333-312-4686	Guadalajara	Desconocido	49070	jimenez@gmail.com	0.00
21	Karen Sofia	Chavez Lazaro	CLKS345SDS	3333784686	Guadalajara	Conocido	49070	chavez@gmail.com	0.00
22	Manuel	Ramirez Cruz	RCMS3475SDS	342-378-4681	Tecalitlan	Conocido	59070	ramirez@gmail.com	0.00
34	Jose Daniel	Orozco Martinez	OMJD3475SDS	548-002-3687	Tamazula	Conocido	30210	joseDa@gmail.com	0.00
36	Cristian	Gonzales Morán	GMCS3475SDS	332-561-0058	Mazamitla	Conocido	49500	cris@gmail.com	0.00
40	Mariana	Alcaraz Gama	ASDD09SD14	123-165-1321	Mazamitla	Conocido	49500	marianita@live.co...	0.00
50	Luis	Castellos	HJSKDH67	456-465-1987	Guzman	Conocido	78920	luiscastellos@live.c...	0.00
60	Melisa	Maya	KJHJ23H23	148-756-1132	Guzman	Conocido	87410	meli@hotmail.com	0.00
70	Natalia	Reyes Godínes	48D6S64FFG	365-214-7800	Mazamitla	Conocido	49600	nata@hormail.com	11112.80

Figura 17. Clientes registrados.

En la Figura 18 se muestran los productos registrados en el inventario, remarcando el Código y la Existencia de los productos que fueron registrados anteriormente en la venta, por lo tanto, se puede apreciar que las existencias de los productos remarcados han disminuido.

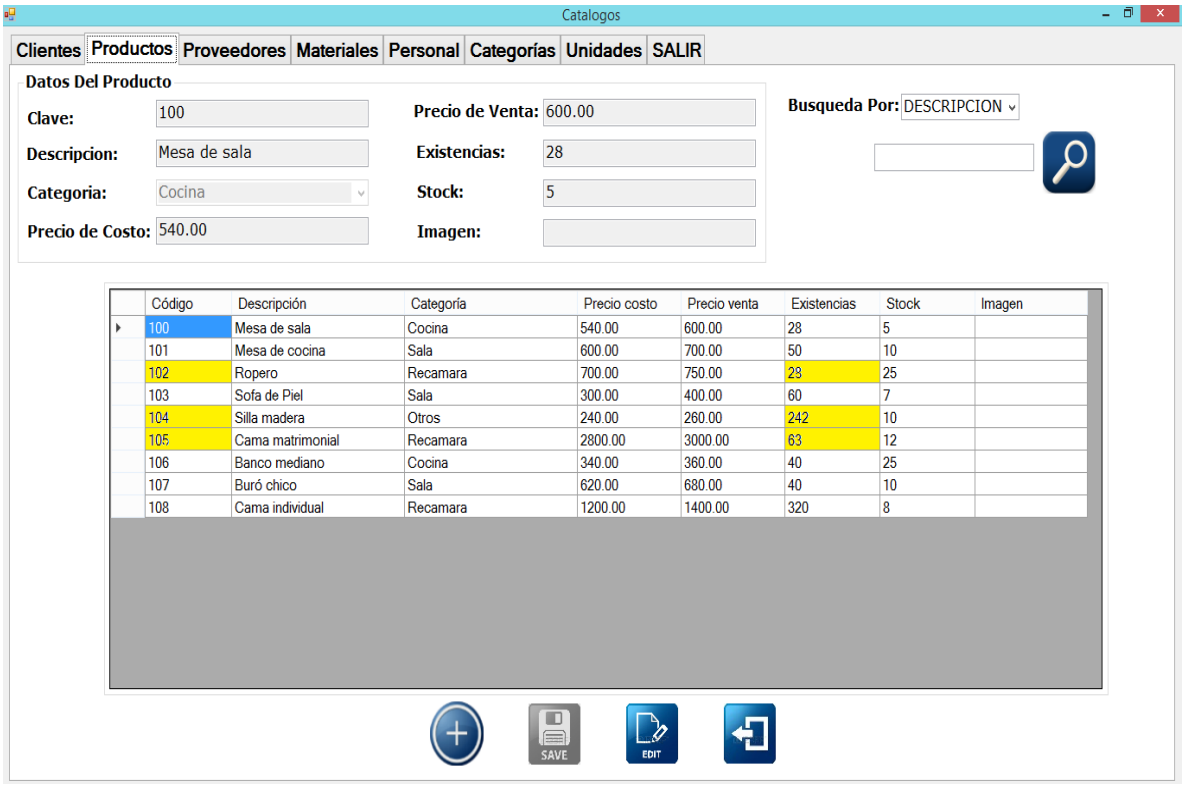


Figura 18. Productos registrados en inventario (Catálogo de Productos).

En la Figura 19 se muestra la venta registrada correctamente.

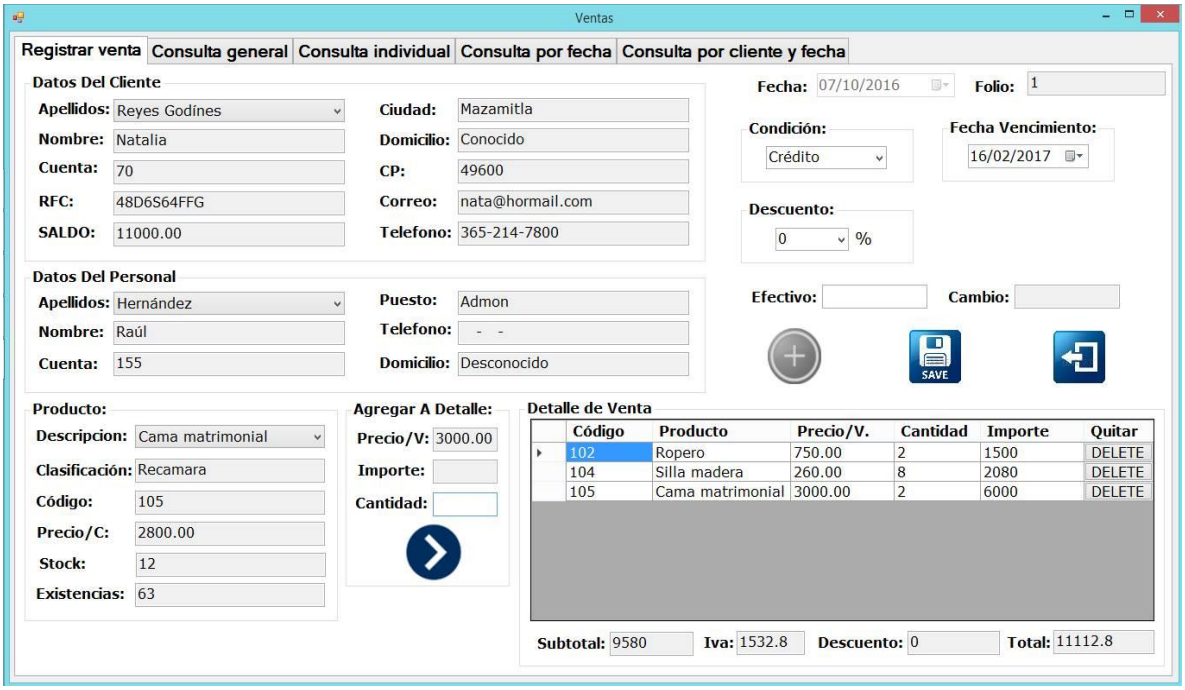


Figura 19. Venta registrada correctamente.

En la Figura 20 se muestra una consulta general de las ventas registradas, por lo tanto, se corrobora que la venta se grabó correctamente, ya que se puede observar en sus datos que es la venta que se registró anteriormente.

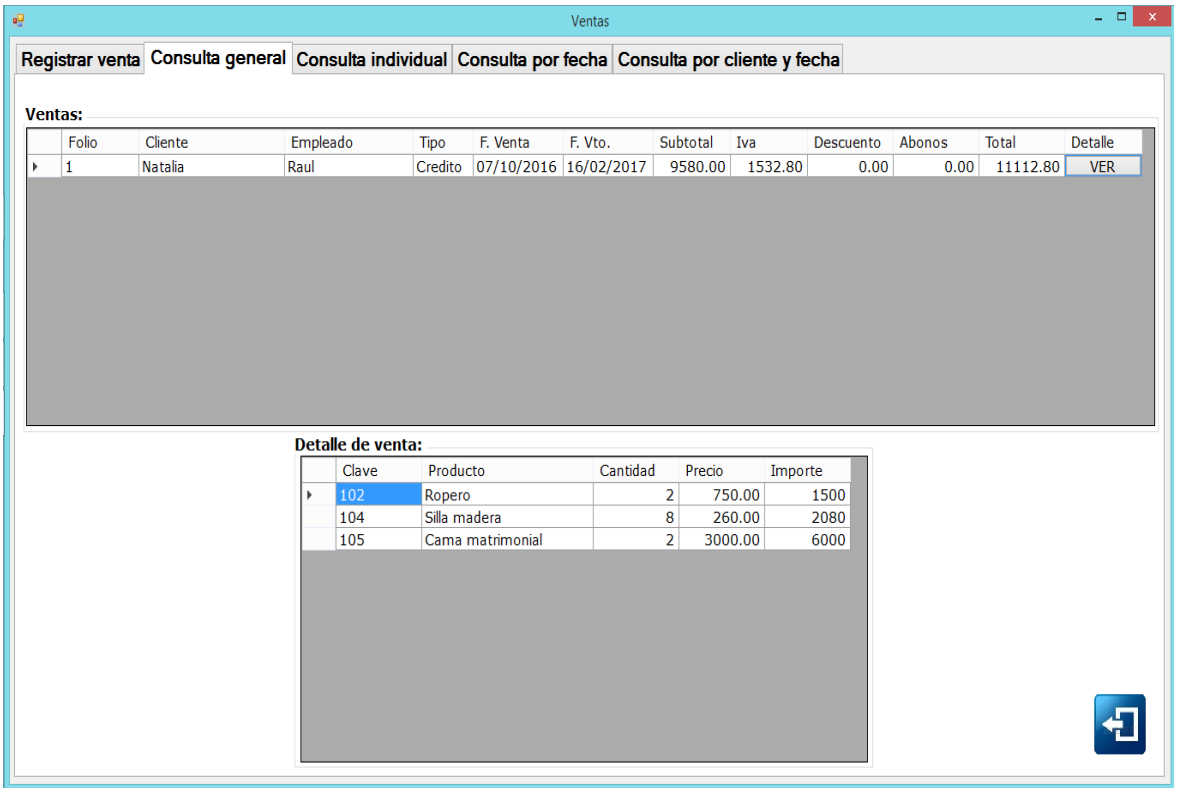


Figura 20. Consulta general de ventas registradas.

4.2.2. Caso de prueba: Registrar abono.

Procedimiento a seguir para registrar correctamente un abono.

1. De la pantalla de inicio o menú principal seleccionar <**Procesos**>.
2. De la interfaz de procesos, seleccionar <**Abonos**>.
3. En la primera pestaña <**Registrar abono**> seleccionar los datos correspondientes.
4. En **Datos del cliente**, seleccionar de la comboBox el cliente indicado, posteriormente de haber seleccionado el cliente, en la rejilla de Ventas se cargarán sus ventas realizadas a crédito.

5. En el apartado de **Realizar abono**, seleccionar el folio de venta a la que se le hará el respectivo abono y dar clic en el botón para habilitar la caja de texto del importe.
6. En seguida introducir el monto del abono y dar clic en el botón de grabar.

En la Figura 21 se muestra la pantalla principal del proceso de abonos.

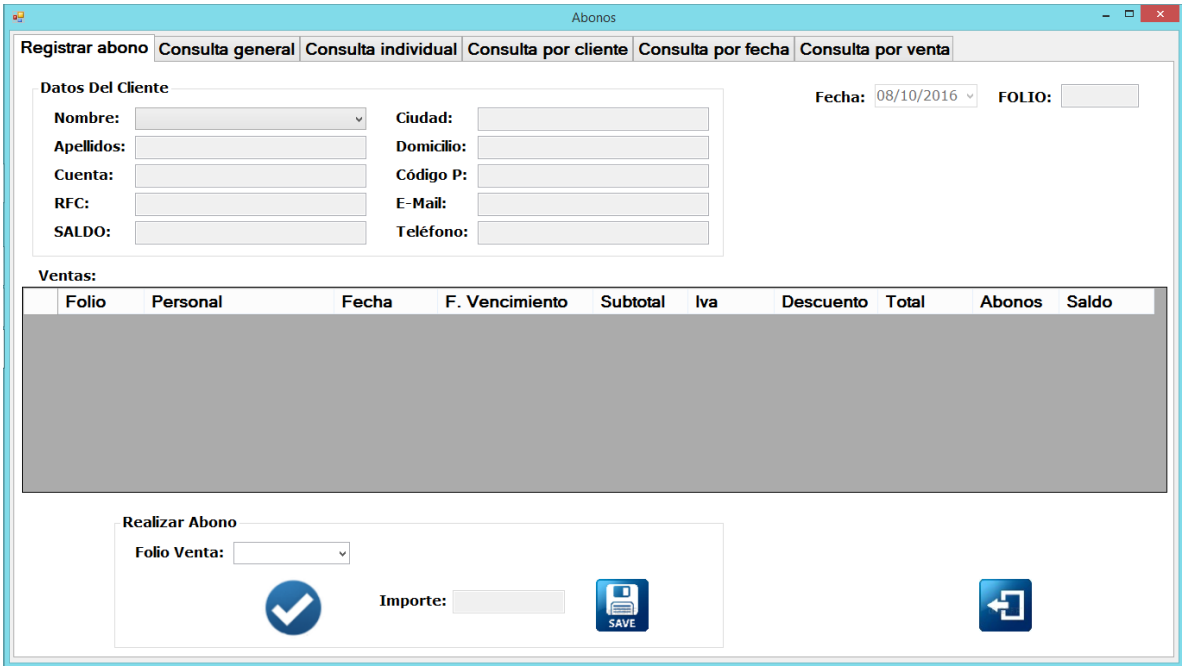


Figura 21. Pantalla principal del proceso de abonos.

Posteriormente se realiza un abono con los siguientes datos:

Entradas:

Cliente: Natalia Reyes Godínez

Folio venta: 1.

Importe del abono: $112.80.

Nota: No hay ningún abono registrado hasta este momento.

Salidas:

Folio de abono: 1.

Saldo del cliente: $ 11,000.00

En la Figura 22 se muestran los clientes registrados, seleccionando el cliente al cual se registró el abono, observando que su saldo ahora es de $11,000.00.

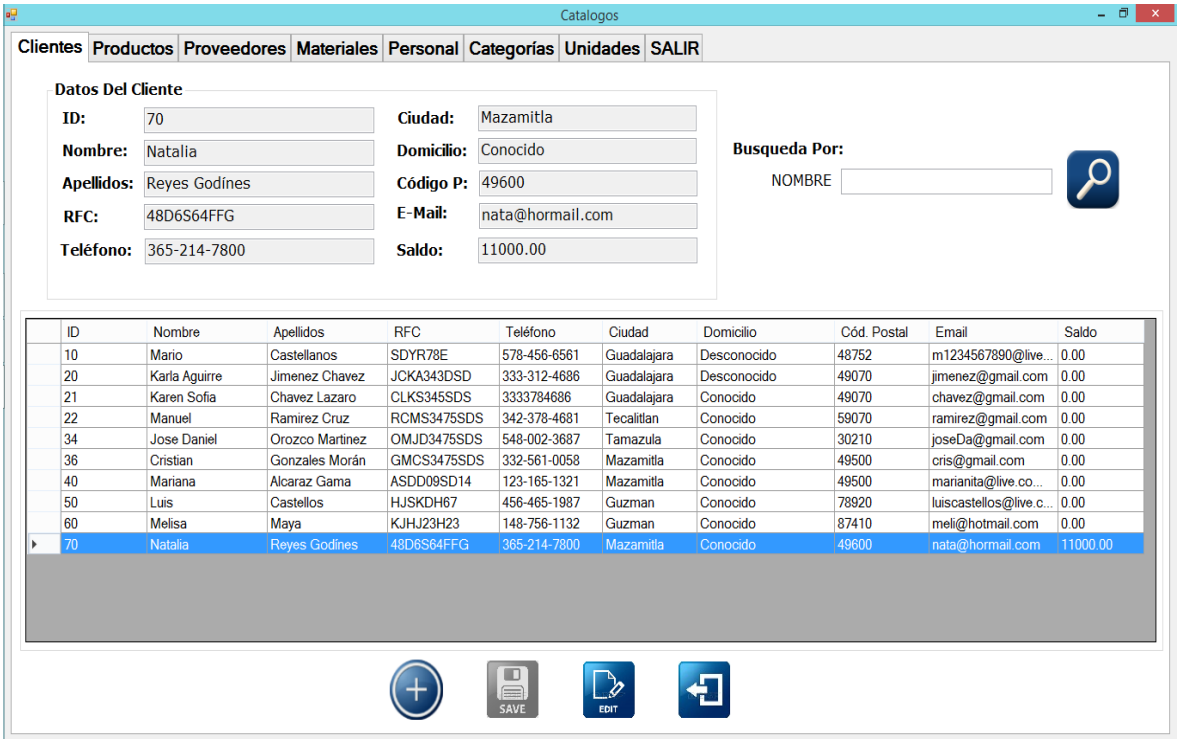


Figura 22. Clientes registrados en catálogo.

En la Figura 23 se muestra el abono registrado correctamente.

Figura 23. Abono registrado correctamente.

En la Figura 24 se muestra una consulta de abonos por venta, por lo tanto, se verifica que el abono se grabó correctamente, ya que se puede observar en sus datos que es el abono que se registró anteriormente.

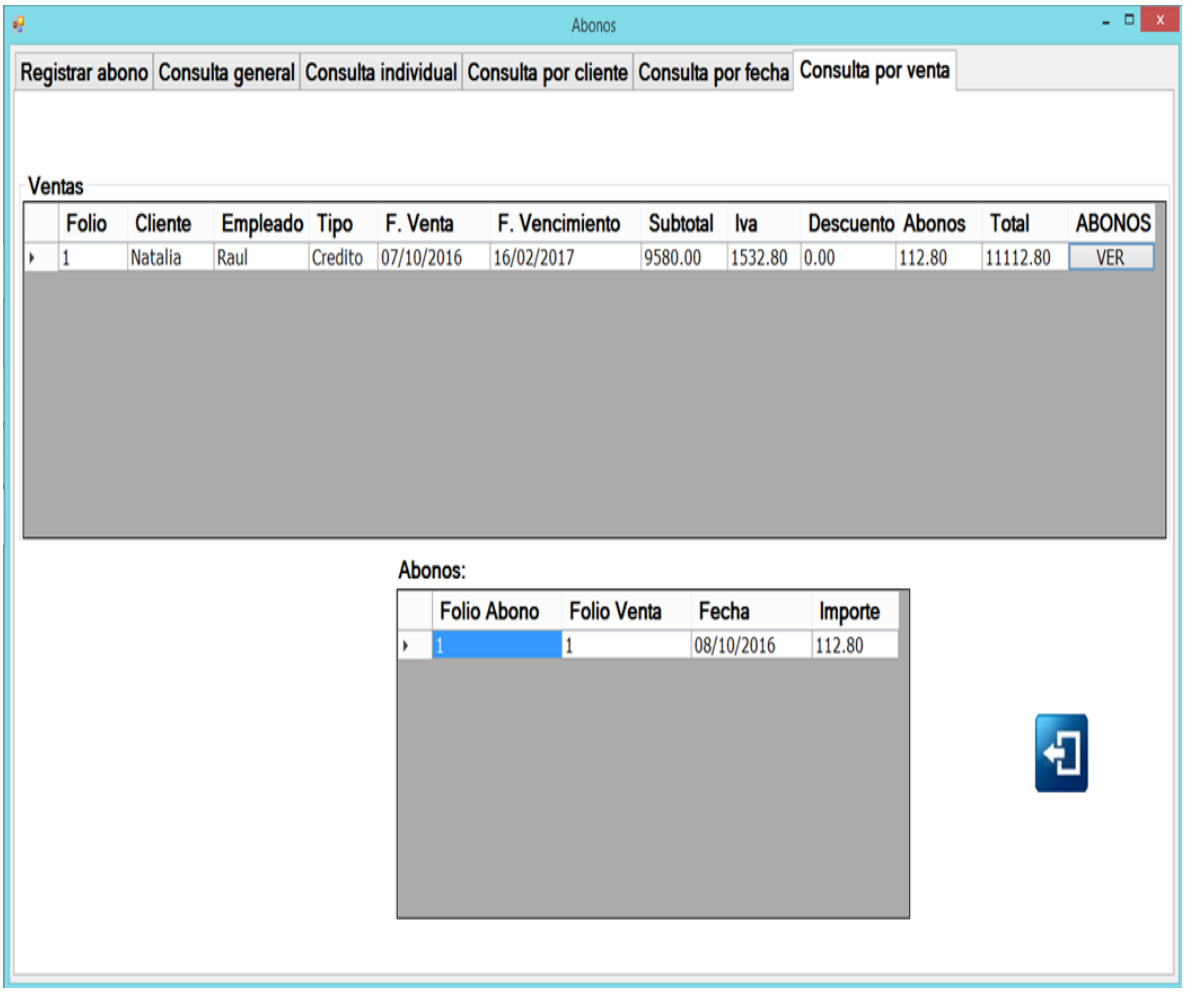


Figura 24. Consulta de abonos por venta.

4.3. Implantación.

Para lograr una implantación exitosa fue necesario llevar un seguimiento detallado de algunas actividades las cuales tendrían como finalidad familiarizarse y cargar la información a la plataforma, así como también enseñar cómo se debe de abastecer de una forma correcta el sistema en cuanto a información y operatividad.

En la Tabla 1 se muestra el cronograma de actividades diseñado para la liberación del sistema.

Tabla 1. Cronograma de Actividades.

ACTIVIDAD	**Diciembre 2016**				**Enero 2017**				**Febrero 2017**			
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Capacitación	X	X	X	X								
Alimentación de catálogos con documentos con saldo actual.					X	X	X					
Verificación de resultados								X	X	X		
Liberación del sistema.											X	X

Capítulo 5. Resultados.

5.1. Diseño de pantallas.

Consulta de ventas.

En la Figura 25 se muestra una consulta general de ventas, así como su respectivo detalle, donde se visualizan los productos de cada una de las ventas.

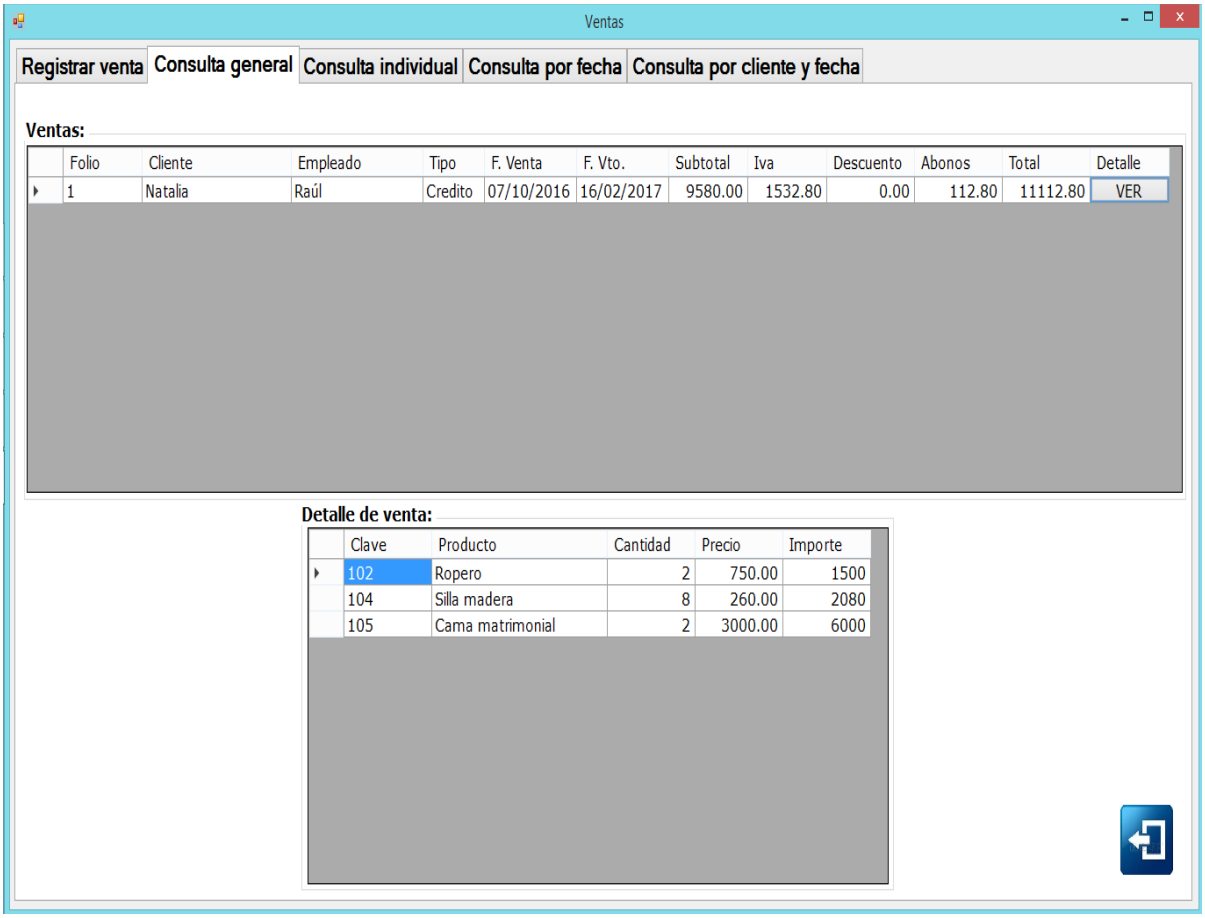


Figura 25. Consulta general de ventas.

En la Figura 26 se muestra una consulta individual de ventas, donde se presentan los datos del cliente, información de la venta, del personal que atendido la venta y su detalle.

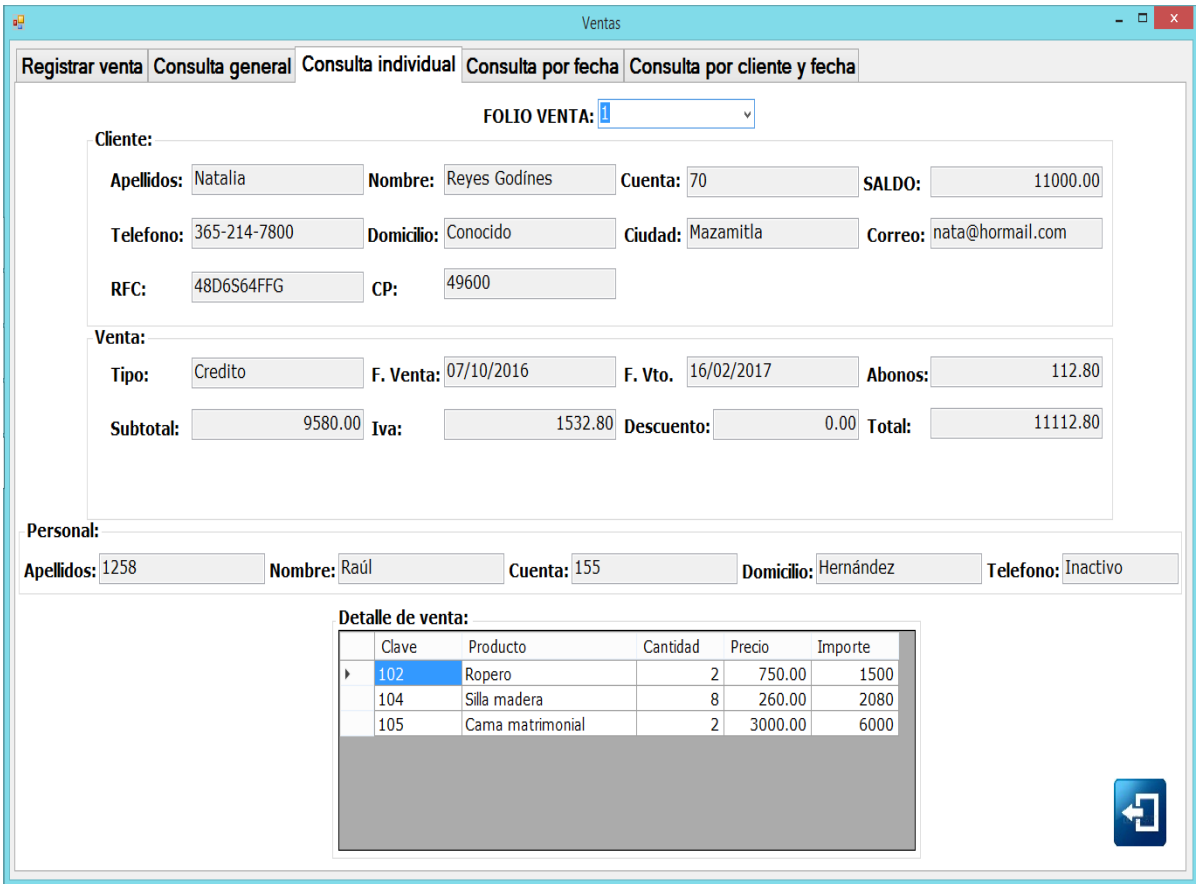


Figura 26. Consulta individual de ventas.

En la Figura 27 se muestra el filtro de ventas realizadas entre un lapso de dos fechas, añadiendo la opción de poder ver el respectivo detalle de venta de cualquiera de éstas.

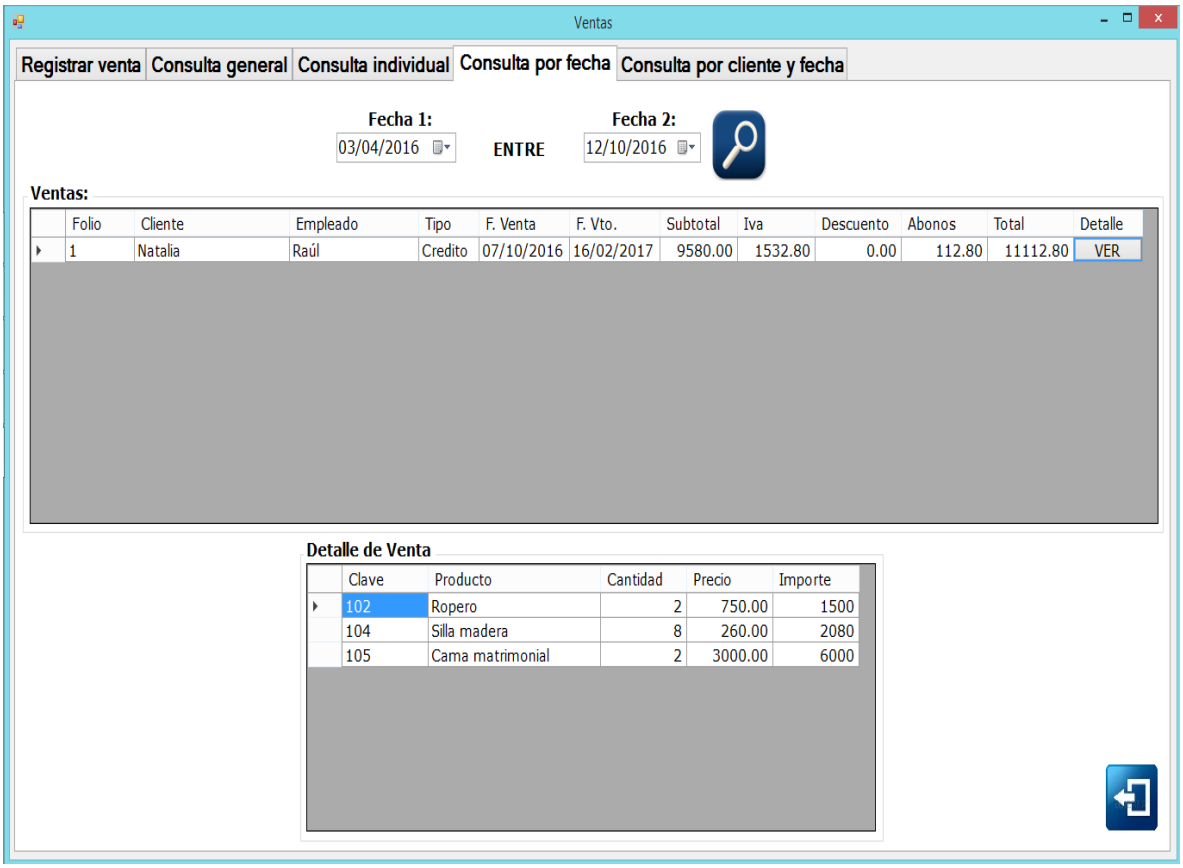


Figura 27. Consulta de ventas entre fechas.

En la Figura 28 se muestra el filtro de ventas realizadas entre un lapso de dos fechas y de un respectivo cliente, añadiendo la opción de poder ver el respectivo detalle de venta de cualquiera de éstas.

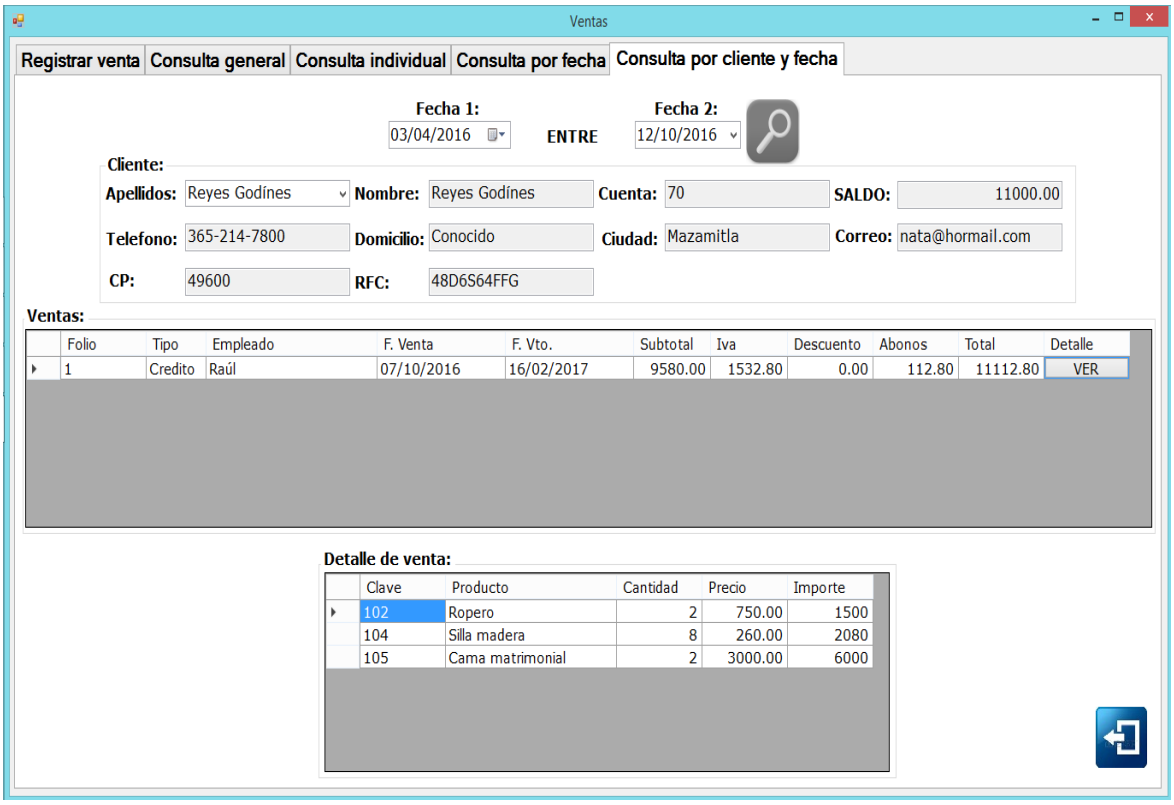


Figura 28. Consulta de ventas entre fechas por cliente.

Consulta de abonos.

En la Figura 29 se muestra una consulta general de abonos, donde se pueden apreciar los datos de la venta y del cliente.

Figura 29. Consulta general de abonos.

En la Figura 30 se muestra una consulta individual de abonos, presentando también datos de la venta y del cliente.

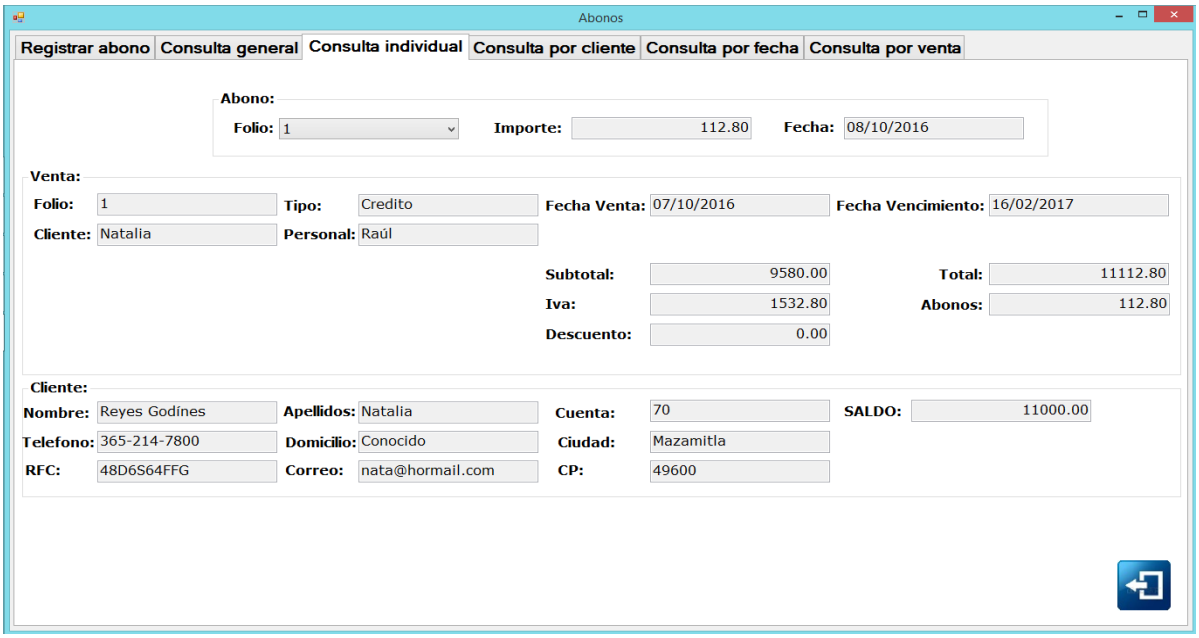


Figura 30. Consulta individual de abonos.

En la Figura 31 se muestra una consulta de abonos por cliente, añadiendo la opción de ver datos de la venta.

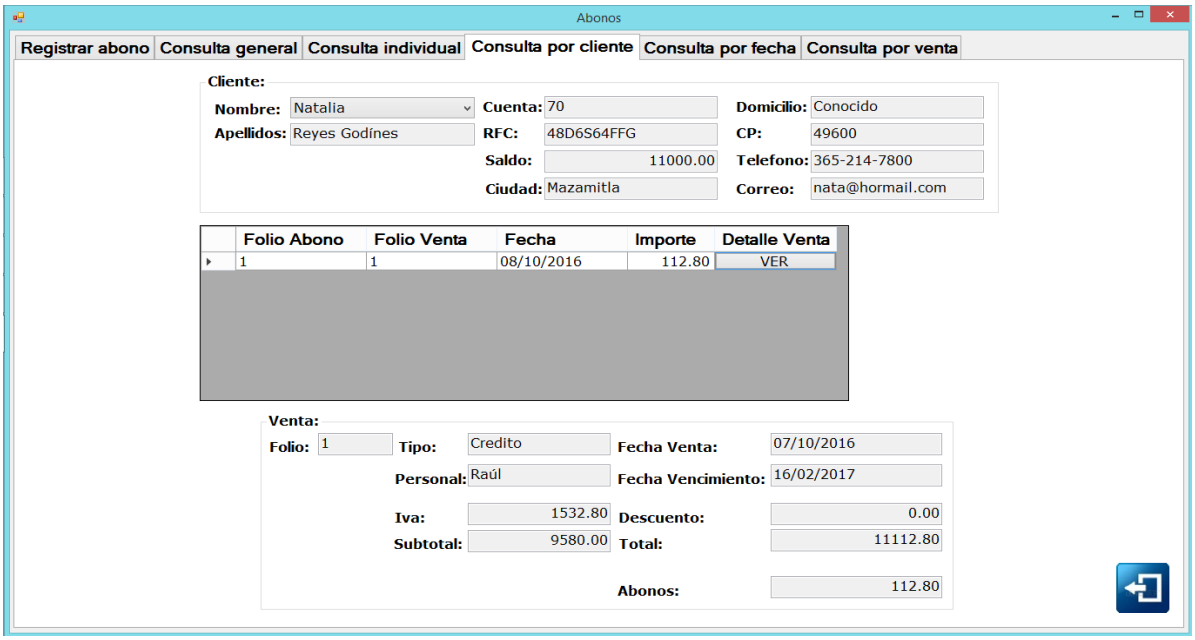


Figura 31. Consulta de abonos por cliente.

En la Figura 32 se muestra una consulta de abonos realizados entre el lapso de dos fechas, mas la información de la venta y del cliente.

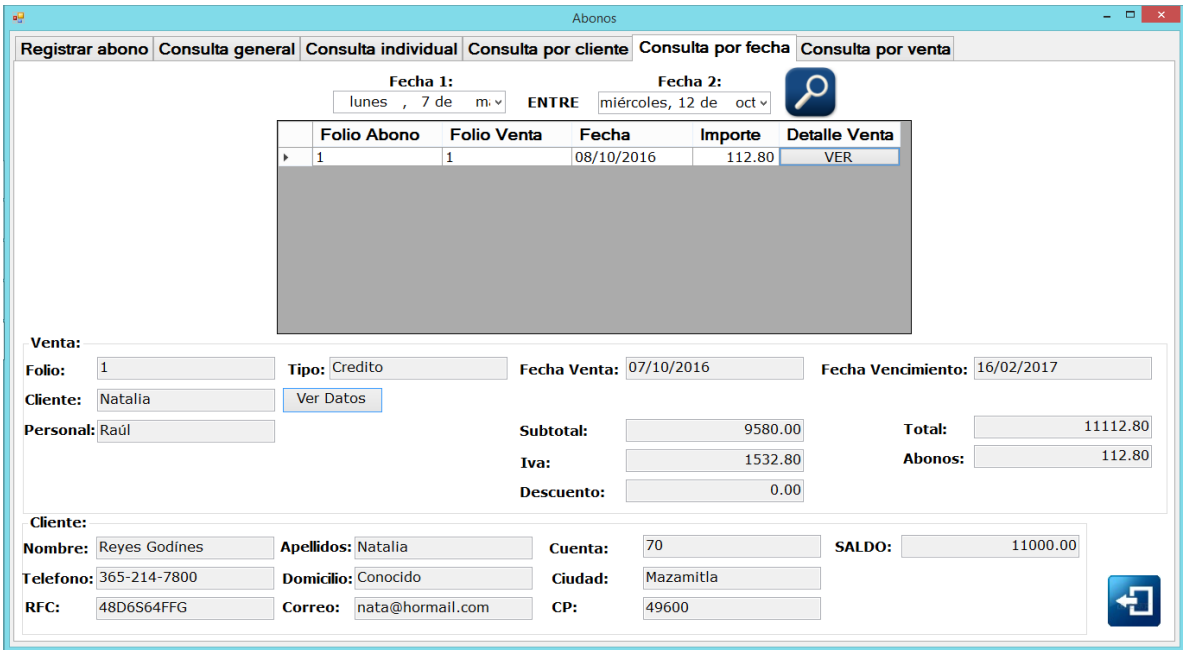


Figura 32. Consulta de abonos entre fechas por cliente.

En la Figura 33 se muestra una consulta general de ventas, más la opción de ver los abonos realizados a una venta específica.

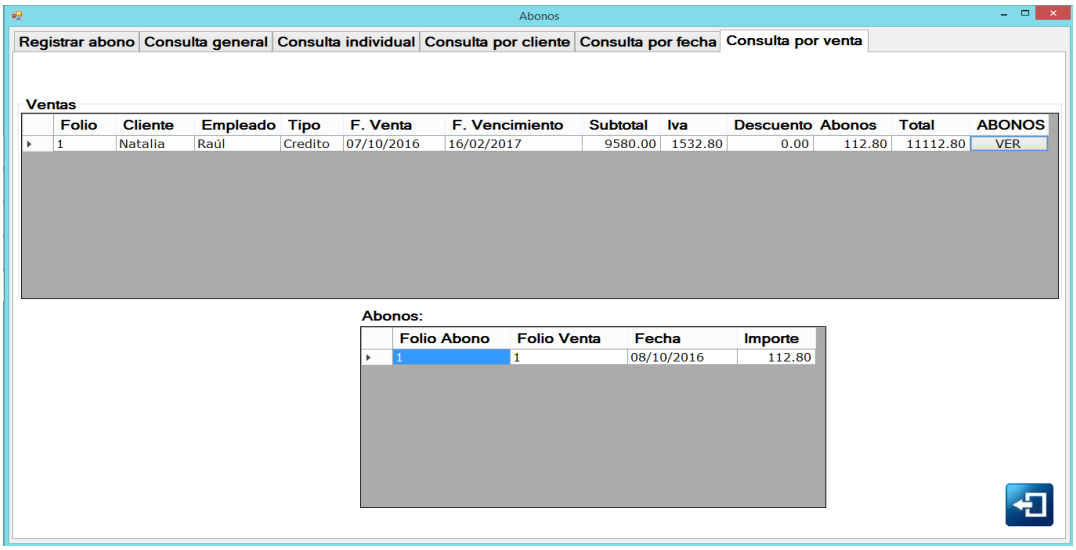


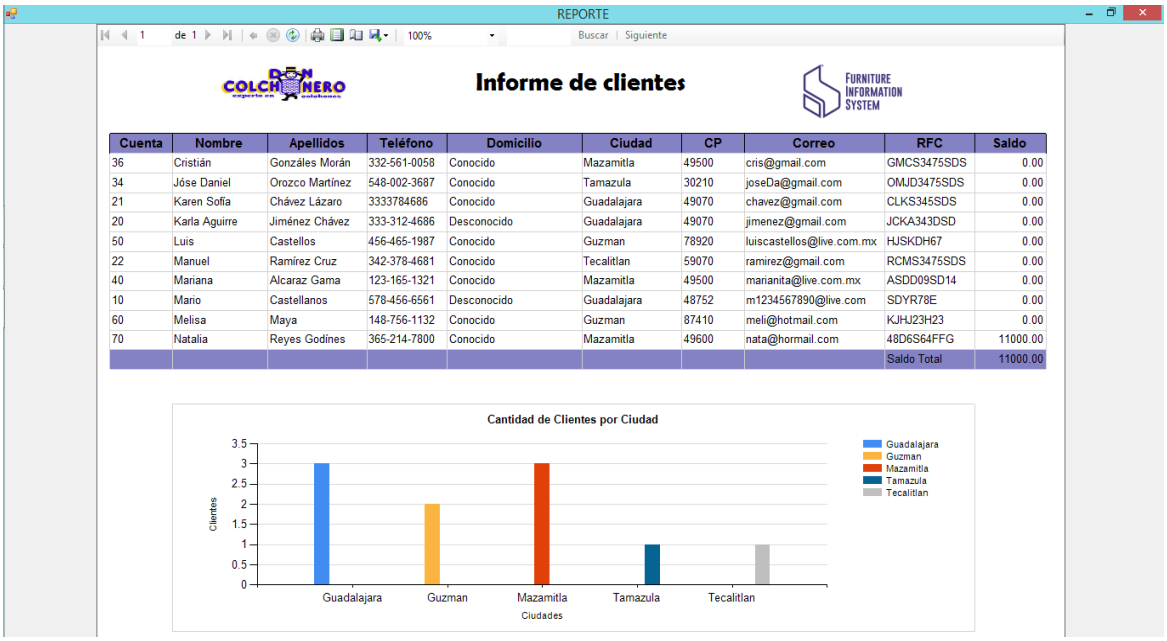
Figura 33. Consulta de abonos por venta.

5.2. Diseño de reportes.

En la Figura 34 se muestra un reporte general de personal, más dos gráficos; uno donde se aprecian el total de personal para cada puesto, y dentro del otro la representación del personal activo e inactivo.

Figura 34. Reporte general de personal.

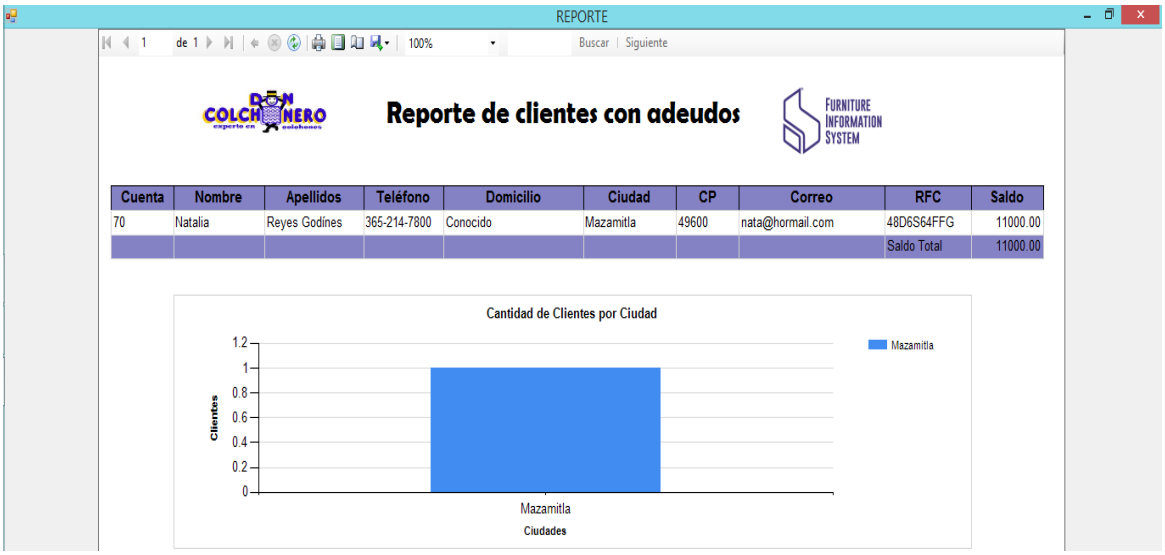
En la Figura 35 se muestra un reporte general de clientes y un gráfico que representa las ciudades de donde pertenecen los clientes.



Cuenta	Nombre	Apellidos	Teléfono	Domicilio	Ciudad	CP	Correo	RFC	Saldo
36	Cristián	Gonzáles Morán	332-561-0058	Conocido	Mazamitla	49500	cris@gmail.com	GMCS3475SDS	0.00
34	Jóse Daniel	Orozco Martínez	548-002-3687	Conocido	Tamazula	30210	joseDa@gmail.com	OMJD3475SDS	0.00
21	Karen Sofía	Chávez Lázaro	3333784686	Conocido	Guadalajara	49070	chavez@gmail.com	CLKS345SDS	0.00
20	Karla Aguirre	Jiménez Chávez	333-312-4686	Desconocido	Guadalajara	49070	jimenez@gmail.com	JCKA343DSD	0.00
50	Luis	Castellos	456-465-1987	Conocido	Guzman	78920	luiscastellos@live.com.mx	HJSKDH67	0.00
22	Manuel	Ramírez Cruz	342-378-4681	Conocido	Tecalitlan	59070	ramirez@gmail.com	RCMS3475SDS	0.00
40	Mariana	Alcaraz Gama	123-165-1321	Conocido	Mazamitla	49500	marianita@live.com.mx	ASDD09SD14	0.00
10	Mario	Castellanos	578-456-6561	Desconocido	Guadalajara	48752	m1234567890@live.com	SDYR78E	0.00
60	Melisa	Maya	148-756-1132	Conocido	Guzman	87410	meli@hotmail.com	KJHJ23H23	0.00
70	Natalia	Reyes Godínes	365-214-7800	Conocido	Mazamitla	49600	nata@hormail.com	48D6S64FFG	11000.00
								Saldo Total	11000.00

Figura 35. Reporte general de clientes.

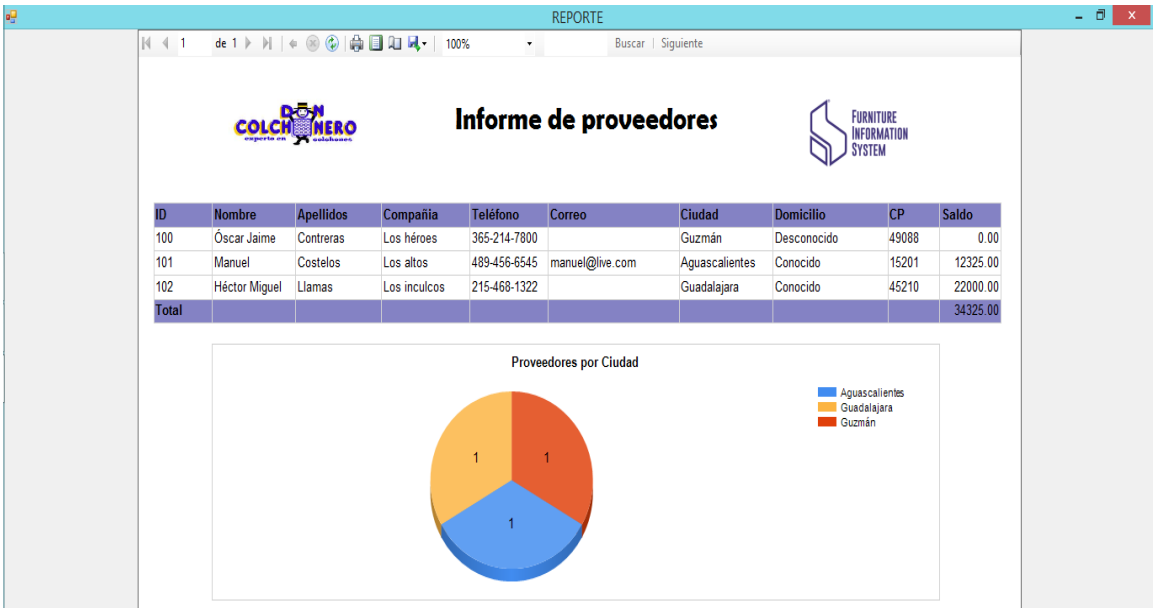
En la Figura 36 se muestra un reporte general de clientes con adeudos y un gráfico que representa las ciudades de donde pertenecen los clientes.



Cuenta	Nombre	Apellidos	Teléfono	Domicilio	Ciudad	CP	Correo	RFC	Saldo
70	Natalia	Reyes Godínes	365-214-7800	Conocido	Mazamitla	49600	nata@hormail.com	48D6S64FFG	11000.00
								Saldo Total	11000.00

Figura 36. Reporte general de clientes con adeudos.

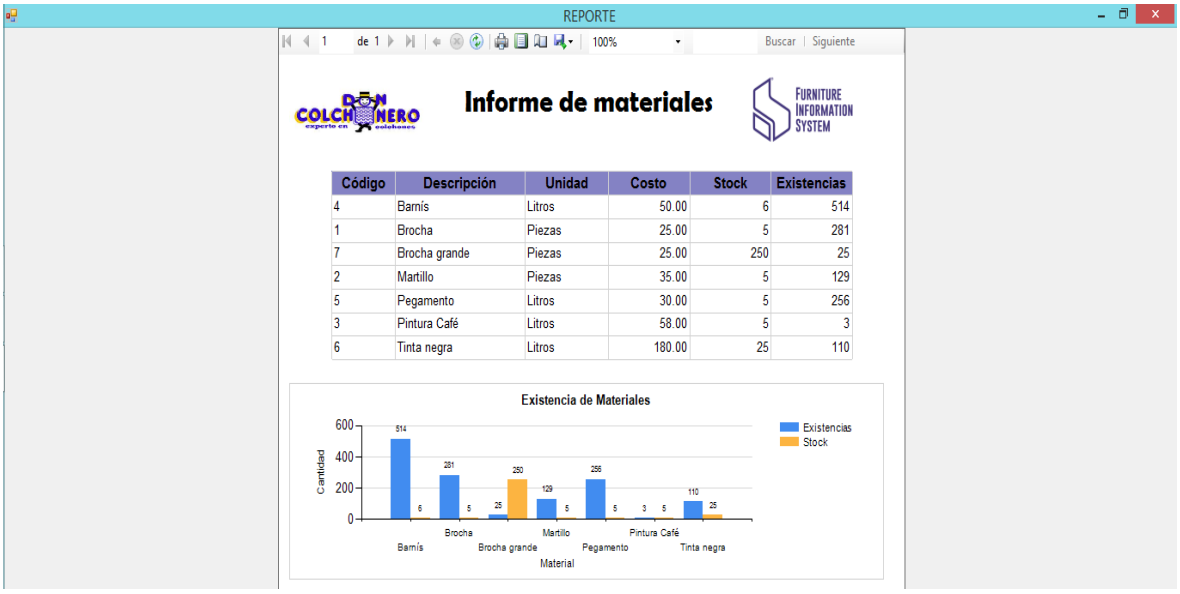
En la Figura 37 se muestra un reporte general de proveedores y un gráfico que representa las ciudades de donde pertenecen los proveedores.



ID	Nombre	Apellidos	Compañía	Teléfono	Correo	Ciudad	Domicilio	CP	Saldo
100	Óscar Jaime	Contreras	Los héroes	365-214-7800		Guzmán	Desconocido	49088	0.00
101	Manuel	Costelos	Los altos	489-456-6545	manuel@live.com	Aguascalientes	Conocido	15201	12325.00
102	Héctor Miguel	Llamas	Los inculcos	215-468-1322		Guadalajara	Conocido	45210	22000.00
Total									34325.00

Figura 37. Reporte general de proveedores.

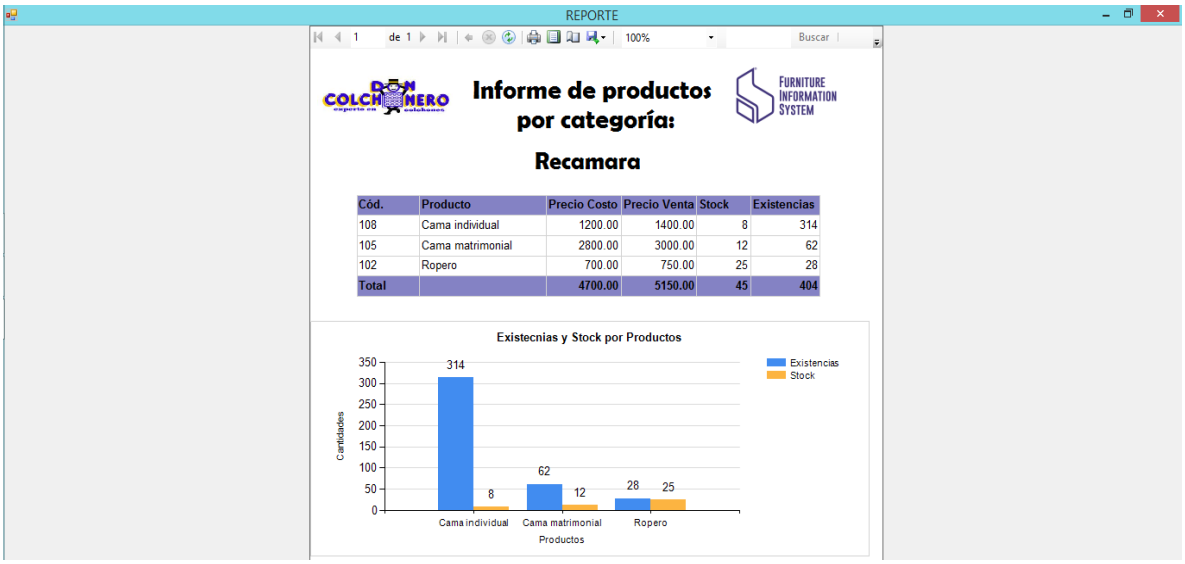
En la Figura 38 se muestra un reporte general de materiales y un gráfico que representa las existencias y stock por material.



Código	Descripción	Unidad	Costo	Stock	Existencias
4	Barnís	Litros	50.00	6	514
1	Brocha	Piezas	25.00	5	281
7	Brocha grande	Piezas	25.00	250	25
2	Martillo	Piezas	35.00	5	129
5	Pegamento	Litros	30.00	5	256
3	Pintura Café	Litros	58.00	5	3
6	Tinta negra	Litros	180.00	25	110

Figura 38. Reporte general de materiales.

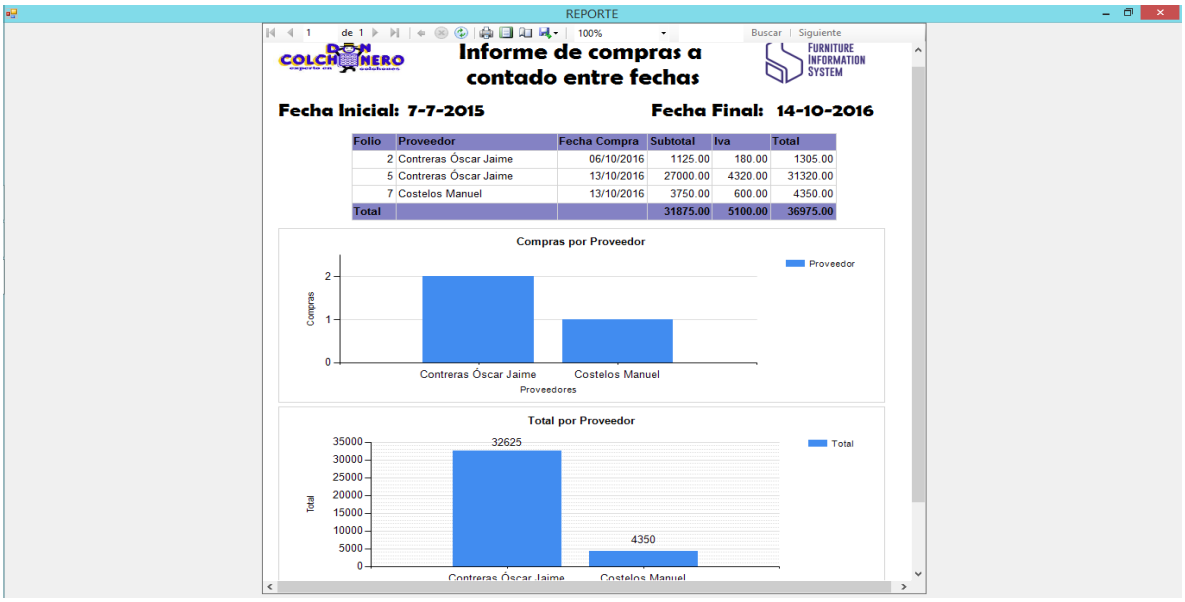
En la Figura 39 se muestra un reporte de productos por categoría y un gráfico que representa las existencias y stock por producto.



Cód.	Producto	Precio Costo	Precio Venta	Stock	Existencias
108	Cama individual	1200.00	1400.00	8	314
105	Cama matrimonial	2800.00	3000.00	12	62
102	Ropero	700.00	750.00	25	28
Total		**4700.00**	**5150.00**	**45**	**404**

Figura 39. Reporte de productos por categoría.

En la Figura 40 se muestra un reporte de compras de contado entre dos fechas y un gráfico que representa la cantidad de compras y monto total por proveedor.



Folio	Proveedor	Fecha Compra	Subtotal	Iva	Total
2	Contreras Óscar Jaime	06/10/2016	1125.00	180.00	1305.00
5	Contreras Óscar Jaime	13/10/2016	27000.00	4320.00	31320.00
7	Costelos Manuel	13/10/2016	3750.00	600.00	4350.00
Total			**31875.00**	**5100.00**	**36975.00**

Figura 40. Reporte de compras de contado entre dos fechas.

En la Figura 41 se muestra un reporte de ventas a crédito entre fechas y dos gráficos que representan las ventas y el monto por venta del cliente.

Folio	Cliente	Personal	Fecha Venta	Fecha Vencimiento	Subtotal	IVA	Descuento	Total	Abonos	Saldo
1	Natalia Reyes Godínes	Raúl Hernández	10/7/2016	2/16/2017	9580.00	1532.80	0.00	11112.80	112.80	11000.00
3	Karla Aguirre Jiménez Chávez	Esteban Miguel Pérez Chávez	10/13/2016	4/6/2017	12960.00	2073.60	0.00	15033.60	0.00	15033.60
4	Luis Castellos	Ohmar López Cruz	10/13/2016	12/7/2016	3200.00	512.00	0.00	3712.00	0.00	3712.00
				Totales:	25740.00	4118.40	0.00	29858.40	112.80	29745.60

Figura 41. Reporte de ventas a crédito entre fechas.

Conclusiones.

Los sistemas de información facilitan la administración de las empresas y hoy en día resultan indispensables para sobresalir en un mercado competitivo.

Mediante el análisis y diseño de un sistema de información se puede detectar la verdadera importancia de desarrollar un producto de buena calidad, del cual es de suma importancia la etapa de implementación y el buen funcionamiento del sistema durante las pruebas.

Esta investigación fue realizada y fundamentada en la investigación de nuevos conceptos, puntos innovadores y nuevas características del IDE Visual Studio en su versión 2015 para conocer y entender la programación en su contexto teórico. De esta forma se reforzaron más los conocimientos de programación en el lenguaje Visual Basic.

Durante el desarrollo de este trabajo de investigación se conocieron nuevas teorías y técnicas de programación las cuales facilitan y hacen más eficiente la manera de programar, ya que se optimiza código de programación.

Se profundizó en el uso de ADO .NET y sus características, así como la arquitectura, proveedores de datos y objetos como DataSet, DataAdapter y DataReader.

Cabe mencionar que se adquirió experiencia al gestionar conexiones a bases de datos de Access, MySQL y SQL Server mediante el IDE, lo cual es interesante y una muy buena alternativa para el desarrollo de aplicaciones y la gestión de orígenes de datos. Por otra parte, los archivos de configuración App.config son necesarios para gestionar las conexiones a los orígenes de datos en aplicaciones que ya han sido empaquetadas, ya que facilitan el uso de parámetros en casos donde es necesario modificar datos de conexión.

Bibliografía.

Briceño, E. A. (2005). *Sistemas de información y su importancia para la empresa*. Recuperado el 18 de 04 de 2017, de https://www.gestiopolis.com/sistemas-informacion-importancia-empresa/

Caminos, G. (2015). *Visual Studio 2015*. Recuperado el 10 de 03 de 2017, de https://hipertextual.com/analisis/visual-studio-2015

CassiaNet. (2017). *Stored Procedure en My SQL*. Recuperado el 11 de 05 de 2017, de https://cassianinet.blogspot.mx/2011/05/stored-procedure-en-mysql-parte-1.html?showComment=1365116180595#c2288059016891351564

Danysoft. (2017). *VISUAL STUDIO 2015*. Recuperado el 14 de 06 de 2007, de http://www.danysoft.com/visual-studio-2015-enterprise-with-msdn/

Demczuk, H. (2017). *Funciones y Procedimientos Almacenados en SQL Server*. Recuperado el 11 de 05 de 2017, de https://hernandemczuk.com/procedimientos-almacenados-en-sql-server/

Goette, E. (2011). *Procedimientos almacenados y funciones en MySQL*. Recuperado el 11 de 05 de 2017, de http://emanuelpeg.blogspot.mx/2011/05/procedimientos-almacenados-y-funciones.html

Jiménez, E. G. (2010). *Aplicaciones con Visual Basic .NET.* México: Alfaomega.

Microsoft. (2016). *MSDN*. Recuperado el 01 de Enero de 2017, de MSDN: https://msdn.microsoft.com/es-es/library/ms254494(v=vs.110).aspx

Microsoft. (2017). *MSDN*. Recuperado el 13 de 06 de 2017, de https://code.msdn.microsoft.com/

Pinzón, Y. (2014). Recuperado el 21 de 04 de 2017, de https://informaticoypf.wordpress.com/2014/04/14/conectar-base-de-datos-mysql-en-visual-studio-2013/

Remon, M. A. (2012). *Programación Orientada a Objetos con Visual Basic 2012.* Lima - Perú: Empresa Editora Macro. Recuperado el 01 de Enero de 2017

Rodriguez, A. (2017). *Conexión a Base de datos con Visual Basic .Net*. Recuperado el 20 de 04 de 2017, de http://www.vb-mundo.com/conexion-a-base-de-datos-con-visual-basic-net/

Solé, J. N. (2014). *Conexión a una base de datos SqlServer con VB.net*. Recuperado el 21 de 04 de 2017, de http://www.jordinaves.com/connexion-base-datos-sqlserver-con-vb-net/

ANEXOS.

Código.

A continuación, se presentan y explican algunos bloques de código utilizados para la programación del sistema de control administrativo para la empresa Don Colchonero Muebles.

Archivo App.config.

En este bloque de código se presenta la configuración del archivo App.config para la implementación y el uso estandarizado del archivo de conexión a la base de datos, el cual brinda la facilidad para la manipulación de la cadena de conexión si se requiere trasladar a otra computadora o se cambia el servidor y/o base de datos.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
  </configSections>
  <connectionStrings>
    <add     name="cn"     connectionString="Data     Source=LAPTOP-
    CDM4D96N\CARDENAS;  Initial  Catalog=TallerMuebles;  Integrated
    Security=True"
    providerName="System.Data.SqlClient" />
  </connectionStrings>
  <startup>
  <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5"
  />
  </startup>
</configuration>
```

Función para llenar un DataGridView.

El siguiente bloque de código hace referencia a una función en Visual Basic la cual hace el llamado a un procedimiento almacenado de SQL Server

“LISTAMATERIALES” para hacer la carga de los registros de la tabla materia prima o materiales a un DataGridView.

```
Sub llenarRejillaMateriales()
  Using cn As New SqlConnection(ConfigurationManager.ConnectionStrings("cn").Conne
  ctionString)
    Using cmd As New SqlCommand
      cmd.Connection = cn
      cmd.CommandType = CommandType.StoredProcedure
      cmd.CommandText = "LISTAMATERIALES"
      Using da As New SqlDataAdapter
        da.SelectCommand = cmd
        Dim ds As New DataSet
        da.Fill(ds, "Materiales")
        dgvMateriales.DataSource = ds.Tables("Materiales")
        dgvMateriales.Columns(0).Width = 100
        dgvMateriales.Columns(1).Width = 200
        dgvMateriales.Columns(2).Width = 150
        dgvMateriales.Columns(3).Width = 120
        dgvMateriales.Columns(3).DefaultCellStyle.Alignment      =
        DataGridViewContentAlignment.TopRight
        dgvMateriales.Columns(4).Width = 120
        dgvMateriales.Columns(4).DefaultCellStyle.Alignment      =
        DataGridViewContentAlignment.TopRight
        dgvMateriales.Columns(5).Width = 120
        dgvMateriales.Columns(5).DefaultCellStyle.Alignment      =
        DataGridViewContentAlignment.TopRight
        ds.Dispose()
      End Using
    End Using
  End Using
End Sub
```

Función para grabar un nuevo cliente.

El siguiente bloque de código representa el registro de un nuevo cliente en la base de datos en una función de Visual Basic, para ello se hace uso de un contenedor Using para establecer la conexión al servidor y base de datos, el cual permite la liberación de recursos al finalizar su uso.

Como primer paso se verifica en la base de datos que no exista el registro por el nombre y apellido del cliente, en dado caso de existir ese cliente, se manda una alerta y el puntero se posiciona en ese registro para verificar que es correcto el resultado de devuelto, de lo contrario, se ejecuta una nueva sentencia de SQL para verificar que el id del cliente este libre y se pueda grabar con dicho id, en caso de ya existir ese id se manda una alerta y se posiciona el puntero en el registro existente.

Por lo tanto si el id está disponible para el registro se manda llamar el procedimiento almacenado de SQL Server “REGISTRARCLIENTE” y se le mandan los parámetros requeridos para dar de alta al nuevo cliente, por último se manda un mensaje de satisfacción y se actualizan los datos del DataGridView mandando el apuntador al registro insertado actualmente para verificar que el registro se ha dado de alta en el sistema exitosamente.

```
Sub registrarCliente()
  Dim res As String
  Try
    Using cn As New SqlConnection(ConfigurationManager.ConnectionStrings("cn").Con
    nectionString)
      Using cmd As New SqlCommand
        cn.Open()
        cmd.Connection = cn
        'Realiza la búsqueda en la base de datos por descripción
        devolviendo el id antes de insertar el producto
```

```
cmd.CommandText = "SELECT IdCliente From Clientes Where
Nombre = '" & txtNombre.Text & "' and Apellidos = '" &
txtApellidos.Text & "'"

res = cmd.ExecuteScalar()

If res <> "" Then

  MsgBox("Ya existe el Usuario:   " & txtNombre.Text & " "
  & txtApellidos.Text)

  idCliente = res

Else

  'Realiza la búsqueda en la base de datos por id
  devolviendo el id y el nombre del producto antes de
  insertar el producto

  cmd.CommandText = "SELECT IdCliente, Nombre, Apellidos
  From Clientes Where IdCliente = '" & txtID.Text & "'"

  lector = cmd.ExecuteReader

  If lector.Read() Then

    MsgBox("Ya existe el Cliente con ID: " & lector(0))

    idCliente = lector(0)

  Else

    lector.Close()

    'cmd.CommandText   =   "insert   into   Clasificacion
    (Descripcion) values ('" & txtClasificacion.Text & "')
    select scope_identity()"

    cmd.CommandType = CommandType.StoredProcedure

    cmd.CommandText = "REGISTRARCLIENTE"

    With cmd.Parameters

      .Add("@IDCLI", SqlDbType.VarChar).Value = txtID.Text

      .Add("@NOM",       SqlDbType.VarChar).Value       =
      txtNombre.Text

      .Add("@APE",       SqlDbType.VarChar).Value       =
      txtApellidos.Text

      .Add("@RFC", SqlDbType.VarChar).Value = txtRFC.Text

      .Add("@TEL",       SqlDbType.VarChar).Value       =
      mtbTelefono.Text
```

```
                .Add("@CD",        SqlDbType.VarChar).Value        =
                txtCiudad.Text
                .Add("@DOM",       SqlDbType.VarChar).Value        =
                txtDomicilio.Text
                .Add("@CP", SqlDbType.VarChar).Value = txtCP.Text
                .Add("@EMAIL",     SqlDbType.VarChar).Value        =
                txtCorreoE.Text
            End With
            cmd.ExecuteNonQuery()
            ' Recupera el id insertado para ubicar el apuntador en
            el registro actualmente insertado
            idCliente = txtID.Text
            'MsgBox("Producto   agregado   correctamente!   "   &
            txtId.Text & " - " & txtDes.Text)
            MessageBox.Show("Cliente registrado correctamente!!! "
            &  txtID.Text  &  "  -  "  &  txtNombre.Text  &  "  "  &
            txtApellidos.Text,  "Mensaje",  MessageBoxButtons.OK,
            MessageBoxIcon.Information)
          End If
        End If
      End Using
    End Using
    actualizarDatosClientes()
  Catch ex As Exception
    operacionFallidaClientes()
  End Try
End Sub
```

Carga de datos a ComboBox.

De tal forma en el siguiente bloque de código se representa la asignación de unidades de medida a un objeto de tipo ComboBox, se hace uso de un contenedor Using para la liberación de recursos y establecer la conexión a la base de datos, también se manda llamar al procedimiento almacenado de SQL Server “LISTAUNIDADES”, así como también se crea un objeto de tipo SqlDataAdapter

para establecer la conexión con un objeto de tipo DataSet que representa el conjunto de datos.

Por último en la propiedad DataSource del ComboBox asigna el conjunto de datos al objeto ComboBox, así como también en sus propiedad DisplayMember se asigna la columna “Descripción” y en la propiedad ValueMember el “ID”, para a la hora de la ejecución se puedan visualizar las unidades y paralelamente se identifique su id respecto a la unidad seleccionada en la ComboBox.

```
Sub llenarUnidades()
  Using cn As New SqlConnection(ConfigurationManager.ConnectionStrings("cn").Conne
  ctionString)
    Using cmd As New SqlCommand
      cmd.Connection = cn
      cmd.CommandType = CommandType.StoredProcedure
      cmd.CommandText = "LISTAUNIDADES"
      Using da As New SqlDataAdapter
        da.SelectCommand = cmd
        Dim ds As New DataSet
        da.Fill(ds, "Unidades")
        cboUnidadesMat.DataSource = ds.Tables("Unidades")
        cboUnidadesMat.DisplayMember = "Descripción"
        cboUnidadesMat.ValueMember = "ID"
        ds.Dispose()
      End Using
    End Using
  End Using
End Sub
```

Formulario de categorías.

Esta parte de código muestra la visualización de un reporte de productos por categoría. Al inicio de la clase se muestran las referencias para importar las clases

u objetos de SQL Server, así como los objetos para reportes y la importación de configuración del archivo App.config para establecer la conexión.

Enseguida está la clase principal del formulario y la recuperación de la conexión mediante un objeto de tipo SqlConnection llamado cn, así como también un objeto de SqlCommand y SqlDataReader, comando y Lector respectivamente.

Después se presenta el evento Load o método principal el cual se ejecuta al lanzar el formulario, donde por consiguiente se abre la conexión y se ejecuta una consulta para posteriormente ejecutar dicho comando con ExecuteReader y recibir los datos en el objeto Lector, y con el apoyo de un ciclo while se cargan los registros de la anterior consulta a un objeto de tipo ComboBox. Después de terminar de iterar en el bucle se cierra la conexión.

```
Imports System.Data.SqlClient
Imports Microsoft.Reporting.WinForms
Imports System.Configuration
Public Class Productos_por_Categoria
  Protected configuracion As ConnectionStringSettings = ConfigurationManager.ConnectionStrings("cn")
  Dim cn As New SqlConnection(configuracion.ConnectionString)
  Dim comando As SqlCommand = cn.CreateCommand()
  Dim Lector As SqlDataReader
  Private Sub Productos_por_Categoria_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    cn.Open()
    comando.CommandText = "Select Descripcion from Clasificacion"
    Lector = comando.ExecuteReader()
    While Lector.Read()
      cboClasificacion.Items.Add(Lector(0))
    End While
    cn.Close()
    cboClasificacion.SelectedIndex = 0 'Determina el primer elemento en la combo para ser mostrado a la hora de la ejecución.
```

```
End Sub
```

Visualización del reporte productos por categorías.

El siguiente método está ligado al botón aceptar, dicho botón se encarga de realizar el llamado para mostrar el reporte en pantalla.

Se abre la conexión para establecer comunicación con la base de datos, después se recupera la categoría seleccionada en la ComboBox para hacer el filtro de los productos, se define una variable de la clase SqlCommand referenciando el procedimiento almacenado, para este caso ProductosPorCategoria, se hace la referencia de la variable anteriormente creada que es de tipo procedimiento almacenado.

Enseguida se crea un objeto de la clase SqlDataAdapter para definir la conexión al origen de datos. Se crean los parámetros necesarios en variables de la clase SqlParameter, estos parámetros serán filtrados y mostrados en el reporte. A continuación se crea un objeto de la clase DataSet para cargar los datos de la consulta en el elemento ReportView y se envían los parámetros para visualizarlos también.

```
Private Sub cmdAceptar_Click(sender As Object, e As EventArgs)
Handles cmdAceptar.Click
  'Determina la posición del formulario
  Reporte.ReportViewer1.Size = New System.Drawing.Size(665, 740)
  Reporte.ReportViewer1.Location = New Point(355, 0)
  cn.Open()
  Dim desc As String = cboClasificacion.Text
  Dim cmd As New SqlCommand("ProductosPorCategoria", cn)
  cmd.CommandType = CommandType.StoredProcedure
  Dim adaptador As New SqlDataAdapter
  adaptador.SelectCommand = New SqlCommand
  adaptador.SelectCommand.Connection = cn
  adaptador.SelectCommand.CommandText = "ProductosPorCategoria"
```

```
        adaptador.SelectCommand.CommandType                                    =
        CommandType.StoredProcedure
        Dim param1 = New SqlParameter("@cat", SqlDbType.Text)
        param1.Direction = ParameterDirection.Input
        param1.Value = CStr(desc)
        adaptador.SelectCommand.Parameters.Add(param1)
        Dim dataset As New DataSet
        adaptador.Fill(dataset)
        dataset.DataSetName = "DataSet1"
        Dim    datasource    As    New    ReportDataSource("DataSet1",
        dataset.Tables(0))
        datasource.Name = "DataSet1"
        datasource.Value = dataset.Tables(0)
        Dim p1 As New ReportParameter("P1", desc)
        Reporte.ReportViewer1.LocalReport.DataSources.Clear()
        Reporte.ReportViewer1.LocalReport.DataSources.Add(datasource)
        Reporte.ReportViewer1.LocalReport.ReportPath                           =
        "C:\Users\Carlos\Documents\Visual                              Studio
        2015\Projects\TallerMuebles\Taller De Muebles\Reporte Productos
        por Categoria.rdlc"
        Reporte.ReportViewer1.LocalReport.SetParameters(New
        ReportParameter() {p1})
        Reporte.ReportViewer1.RefreshReport()
        Reporte.Show()
        cn.Close()
    End Sub
    Private  Sub  cmdSalir_Click(sender  As  Object,  e  As  EventArgs)
    Handles cmdSalir.Click
    Me.Close()
    End Sub
End Class
```

Procedimientos almacenados.

A continuación, se presentan y explican los procedimientos almacenados en SQL Server implementados para la programación del sistema de control administrativo para la empresa Don Colchonero Muebles.

Procedimiento almacenado ReportePersonal.

Este procedimiento almacenado muestra una relación general de personal ordenados por nombre alfabéticamente.

```
-- Relación de personal.
Create Procedure ReportePersonal
As
    select IdPersonal, Nombre, Apellidos, Telefono, Domicilio,
    Puesto, FechaIngreso, Estado from Personal Order By Nombre
Go
```

Procedimiento almacenado ReporteClientes.

Este procedimiento almacenado muestra una relación general de clientes ordenados por nombre alfabéticamente.

```
-- Relación de clientes.
Create Procedure ReporteClientes
As
    Select IdCliente, Nombre, Apellidos, Telefono, Domicilio,
    Ciudad, CP, CorreoE, RFC, Saldo, Pw from Clientes Order By
    Nombre
Go
```

Procedimiento almacenado ClientesConAdeudos.

Este procedimiento almacenado muestra una relación de aquellos clientes que tienen un saldo mayor a 0 y son ordenados por nombre alfabéticamente.

```
-- Relación de clientes con adeudos.
Create procedure ClientesConAdeudos
AS
```

```
    select * from Clientes where Saldo > 0 order by Nombre
GO
```

Procedimiento almacenado ProveedoresInf.

Este procedimiento almacenado muestra una relación general de proveedores ordenados por apellidos alfabéticamente.

```
Create procedure ProveedoresInf
AS
    select IdProveedor, Apellidos, Nombre, Compañia, Telefono,
    CorreoE, Ciudad, Domicilio, CP, Saldo from Proveedores order
    by Apellidos
GO
```

Procedimiento almacenado ProveedoresConAdeudos.

Este procedimiento almacenado muestra una relación de aquellos proveedores que tienen un saldo mayor a 0 y son ordenados por apellidos alfabéticamente.

```
-- Relación de proveedores con adeudos.
Create procedure ProveedoresConAdeudos
AS
    select * from Proveedores where Saldo > 0 order by Apellidos
GO
```

Procedimiento almacenado ReporteMateriales.

Este procedimiento almacenado muestra una relación general de materiales ordenados por nombre alfabéticamente.

```
-- Relación de materiales.
Create Procedure  ReporteMateriales
As
    select * from Materiales Order By Nombre
Go
```

Procedimiento almacenado ProductosPorCategoria.

Este procedimiento almacenado muestra una relación de productos por categoría, puesto que será fundamental el uso de un parámetro de entrada para recibir la categoría para realizar un filtro de aquellos productos pertenecientes a dicha categoría, el conjunto de datos es ordenado por descripción del producto alfabéticamente.

```
-- Relación de productos por categoría.
Create procedure ProductosPorCategoria
@cat varchar(20)
AS
    select Productos.IdProducto, Productos.Descripcion,
    PrecioCosto, PrecioVenta, Existencias, Stock from Productos
    inner join Clasificacion on Productos.IdClasificacion =
    Clasificacion.IdClasificacion where Clasificacion.Descripcion
    = @cat order by Productos.Descripcion
GO
```

Procedimiento almacenado ComprasDeContadoEntreFechas.

Este procedimiento almacenado muestra una relación de compras de contado entre dos fechas, por lo tanto, es necesario el uso de dos parámetros de entrada para recibir fecha inicio y fecha fin para realizar el filtro de aquellas compras realizadas en el intervalo de tiempo, el conjunto de datos es ordenado por apellidos de proveedor alfabéticamente.

```
-- Relación de compras de contado entre fechas.
Create procedure ComprasDeContadoEntreFechas
@f1 date,
@f2 date
AS
    Select FolioCompra, Proveedores.Apellidos +' '+
    Proveedores.Nombre as Proveedor, Fecha as Fecha_Compra,
    Subtotal, IVA, Subtotal + IVA as Total from Compras inner join
    Proveedores on Compras.IdProveedor = Proveedores.IdProveedor
    where Compras.Condicion = 'Contado' and Fecha between @f1 and
    @f2 order by Proveedores.Apellidos
```

```
GO
```

Procedimiento almacenado ComprasDeCreditoEntreFechas.

Este procedimiento almacenado muestra una relación de compras de crédito entre dos fechas, por lo tanto, es necesario el uso de dos parámetros de entrada para recibir fecha inicio y fecha fin para realizar el filtro de aquellas compras realizadas en el intervalo de tiempo, el conjunto de datos es ordenado por apellidos de proveedor alfabéticamente.

```
-- Relación de compras de crédito entre fechas.
Create procedure ComprasDeCreditoEntreFechas
@f1 date,
@f2 date
AS
    Select Compras.FolioCompra, Proveedores.Apellidos +' '+
    Proveedores.Nombre as Proveedor, Fecha as Fecha_Compra,
    FechaVencimiento, Subtotal, IVA, Subtotal + IVA as Total,
    Materiales.IdMaterial, Materiales.Nombre,
    DetalleCompra.Cantidad from Compras inner join Proveedores on
    Compras.IdProveedor = Proveedores.IdProveedor inner join
    DetalleCompra on Compras.FolioCompra =
    DetalleCompra.FolioCompra inner join Materiales on
    DetalleCompra.IdMaterial = Materiales.IdMaterial  where
    Compras.Condicion = 'Credito' and Fecha between @f1 and @f2
    order by Proveedores.Apellidos asc, Compras.FolioCompra asc,
    Materiales.Nombre asc
GO
```

Procedimiento almacenado VentasContado.

Este procedimiento almacenado muestra una relación general de ventas de contado.

```
-- Relación de ventas de contado.
create procedure VentasContado
AS
    Select FolioVenta, Clientes.Nombre Cliente, Personal.Nombre
    Personal, Condicion, Fecha, Subtotal, IVA, Descuento, Abonos
```

```
    from Ventas inner join Personal
    on Ventas.IdPersonal = Personal.IdPersonal
    inner join Clientes
    on Ventas.IdCliente = Clientes.IdCliente
    where Condicion = 'Contado'
GO
```

Procedimiento almacenado VentasCredito.

Este procedimiento almacenado muestra una relación general de ventas de crédito.

```
-- Relación de ventas de crédito.
create procedure VentasCredito
AS
    Select FolioVenta, Clientes.Nombre Cliente, Personal.Nombre
    Personal, Condicion, Fecha, FechaVencimiento, Subtotal, IVA,
    Descuento, Abonos
    from Ventas inner join Personal
    on Ventas.IdPersonal = Personal.IdPersonal
    inner join Clientes
    on Ventas.IdCliente = Clientes.IdCliente
    where Condicion = 'Credito'
GO
```

Procedimiento almacenado VentasCreditoFechas.

Este procedimiento almacenado muestra una relación de ventas de crédito entre dos fechas, por lo tanto, es necesario el uso de dos parámetros de entrada para recibir fecha inicio y fecha fin para realizar el filtro de aquellas ventas realizadas en el intervalo de tiempo.

```
-- Relación de ventas de crédito por fechas.
create procedure VentasCreditoFechas
@Fecha1 date,
@Fecha2 date
AS
```

```
    Select FolioVenta, Clientes.Nombre Cliente, Personal.Nombre
    Personal, Condicion, Fecha, FechaVencimiento, Subtotal, IVA,
    Descuento, Abonos
    from Ventas inner join Personal
    on Ventas.IdPersonal = Personal.IdPersonal
    inner join Clientes
    on Ventas.IdCliente = Clientes.IdCliente
    where (Fecha Between @Fecha1 and @Fecha2) and Condicion =
    'Credito'
GO
```

Procedimiento almacenado VentasContadoFechas.

Este procedimiento almacenado muestra una relación de ventas de contado entre dos fechas, por lo tanto, es necesario el uso de dos parámetros de entrada para recibir fecha inicio y fecha fin para realizar el filtro de aquellas ventas realizadas en el intervalo de tiempo.

```
-- Relación de ventas de contado en un lapso de fechas.
create procedure VentasContadoFechas
@Fecha1 date,
@Fecha2 date
AS
    Select FolioVenta, Clientes.Nombre Cliente, Personal.Nombre
    Personal, Condicion, Fecha, Subtotal, IVA, Descuento, Abonos
    from Ventas inner join Personal
    on Ventas.IdPersonal = Personal.IdPersonal
    inner join Clientes
    on Ventas.IdCliente = Clientes.IdCliente
    where (Fecha Between @Fecha1 and @Fecha2) and Condicion =
    'Contado'
GO
```

Procedimiento almacenado ProduccionesEntreFechas.

Este procedimiento almacenado muestra una relación de producciones entre dos fechas, por lo tanto, es necesario el uso de dos parámetros de entrada para

recibir fecha inicio y fecha fin para realizar el filtro de las producciones realizadas en el intervalo de tiempo.

```
-- Relación de producciones entre fechas.
Create procedure ProduccionesEntreFechas
@f1 date,
@f2 date
AS
    select Produccion.IdProduccion, FechaInicio, FechaFinal,
    Produccion.Descripcion, DetalleProduccion.IdProducto,
    Productos.Descripcion  as Producto, DetalleProduccion.Cantidad
    from Produccion inner join DetalleProduccion
    on Produccion.IdProduccion = DetalleProduccion.IdProduccion
    inner join Productos on DetalleProduccion.IdProducto =
    Productos.IdProducto
    where FechaInicio Between @f1 and @f2 and FechaFinal Between
    @f1 and @f2 order by Produccion.IdProduccion asc,
    Productos.Descripcion asc
GO
```