TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CD. GUZMÁN

TITULACIÓN INTEGRAL
TESIS

TEMA:

TÉCNICAS DE ACCESO A DATOS A TRAVÉS DE UN FRAMEWORK EN UN ENTORNO DE DESARROLLO INTEGRADO.

QUE PARA OBTENER EL TÍTULO DE:

INGENIERA INFORMÁTICA

PRESENTA:

TERESA LIZETH MORÁN ACEVEDO

Cd. Guzmán, Municipio de Zapotlán el Grande, Jal, 13/febrero/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:	Teresa Lizeth Morán Acevedo
Carrera:	Ingeniería Informática
No. De Control:	**12290698**
Nombre del Proyecto:	Técnicas de acceso a datos a través de un Framework en 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

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

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

M.C. RAQUEL OCHOA ORNELAS	DRA. MARÍA GUADALUPE SÁNCHEZ CERVANTES	LI. JOEL OCHOA VÁZQUEZ
Nombre y Firma del Asesor	Nombre y Firma del Revisor	Nombre y Firma del Revisor

c.c.p. Archivo
JRGV/JMTS/RZG/Ejss

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

SIGLAS

UML:	Lenguaje Unificado de Modelado (Unified Modeling Language)
OMG	Object Management Group
FN	Formas Normales
FNBC	Forma Normal Boyce-Codd
1FN	Primera Forma Normal
2FN	Segunda Forma Normal
3FN	Tercera Forma Normal
4FN	Cuarta Forma Normal
5FN	Quinta Forma Normal
DF	Dependencias Funcionales
DFD	Diagrama de Flujo de Datos
ESS	Sistemas de Apoyo a Ejecutivos (Employee Self-Service)
MIS	Sistema de Información Gerencial (Management Information System)
DSS	Sistema de apoyo a toma de decisiones (Decision Support System)
KWS	Sistemas del Trabajo del Conocimiento (Knowledge Work System)

TPS	Sistema de Procesamiento de Transacciones
	(Transaction Processing System)
SI	Sistemas de Información
SGBD	Sistema Gestor de Bases de Datos
SQL	Lenguaje de consulta estructurado
	(Structured Query Language)
LDD	Lenguaje de definición de datos
	(Data Definition Language)
LMD	Lenguaje de manipulación de datos
	(Data Manipulation Language)
MB	MegaBytes
KB	KiloBytes
OleDB	Enlace a incrustación de objetos para bases de datos
	(Object Linking and Embedding for Databases)
ODBC	Open DataBase Connectivity
LINQ	Language Integrated Query
CLR	Entorno en tiempo de ejecución de lenguaje común
	(Commun Language Runtime)

CAPÍTULO 1. INTRODUCCIÓN

En el área de Informática, es importante subrayar que las tecnologías computacionales se encuentran en constante evolución; esto significa que los cambios ocurren constantemente sin esperar a que nos acostumbremos a ellos. Desde los primeros pasos de la informática en los años 80 hasta la actualidad, el informático ha enfrentado de manera permanente herramientas nuevas para el desarrollo de software que debe apropiar para estar en sintonía con las necesidades del medio.

Además, la inserción de las nuevas tecnologías de la informática y las comunicaciones sigue avanzando y los Ingenieros en Informática tienen como desafío aplicarlas aún así con sus ventajas y desventajas, como toda tecnología que se inserta en alguna esfera para la ciencia.

El entorno de desarrollo integrado de Microsoft Visual Studio 2015 mantiene un ritmo incansable de evolución aumentando el nivel de productividad que la hace ser muy rentable para su utilización en desarrollos profesionales.

El diseño del modelo que se propone está basado en las tecnologías de Microsoft Visual Studio Community 2015, permitiendo el uso adecuado de la herramienta para el desarrollo de interfaces, patrones de programación y arquitectónicos, facilitando la asimilación de conceptos actuales de programación de una manera más directa y sencilla, y asegurando el desarrollo de software empresarial de calidad a través de la gestión del ciclo de vida del software que permitirá, desde un solo enfoque, cubrir las necesidades de todos los puestos y roles relacionados.

El diseño de un modelo base de negocio incluye técnicas orientadas a las transacciones, LINQ to SQL, arquitectura N-capas, para que de esta manera impacte en beneficio de los desarrolladores de sistemas, al proponer un modelo que sirva de guía y referencia para facilitar el desarrollo de software mediante la tecnología de Microsoft Visual Studio Community 2015, generando información confiable y disponible que permita enriquecer y perfeccionar el desarrollo de aplicaciones informáticas empresariales de calidad.

CAPÍTULO 2. PLANTEAMIENTO DE LA INVESTIGACIÓN

2.1 JUSTIFICACIÓN

Las tecnologías computacionales han evolucionado en todos los sentidos, desde la manera de proveer la conexión a Internet por medio de dispositivos móviles hasta el desarrollo de lenguajes y técnicas de programación.

El entorno de desarrollo integrado de Microsoft Visual Studio 2015 ha surgido con muchas ventajas y mejoras a versiones anteriores que la hacen ser una de las plataformas favoritas que ha permanecido en el medio informático a través de los años. El proyecto propone el diseño de un modelo base de negocio que articula la tecnología que ofrece Microsoft Visual Studio 2015 en temas enfocados a técnicas de programación avanzadas con conexión a datos, como lo son transacciones, LINQ to SQL, que permitirá impulsar el desarrollo de sistemas de información.

El modelo de negocio implementa las técnicas anteriores describiendo de manera detallada los componentes involucrados en cada tarea de una manera descriptiva, demostrando a través de diferentes pruebas el correcto funcionamiento de cada rutina y documentando a detalle su implementación.

2.2 PROBLEMA DE LA INVESTIGACIÓN

Actualmente se cuenta con muy poca información sobre las características de la nueva versión de Visual Studio Community 2015 en relación a los nuevos conceptos en la programación de sistemas de información. Por lo anterior, es de suma importancia explorar las nuevas técnicas de programación en el desarrollo de proyectos informáticos. Esta investigación permitirá acceder a la nueva tecnología que ofrece Microsoft y preparar material accesible a través de ejemplos prácticos enfocados a un problema real propuesto: Desarrollar un sistema de información que permita el control de pedidos que realizan los clientes sobre los artículos que tiene a la venta una determinada empresa.

2.3 OBJETIVOS GENERALES Y ESPECÍFICOS

2.3.1 OBJETIVO GENERAL

Impulsar el desarrollo de aplicaciones empresariales con la integración de técnicas de programación avanzada a través de un Framework de desarrollo integrado.

2.3.2 OBJETIVOS ESPECÍFICOS

- Obtener información relacionada con el procesamiento de transacciones.
- Gestionar el control de transacciones en una base de datos SQL Server.
- Examinar la tecnología LINQ to SQL para aplicaciones cliente servidor.
- Implementar aplicaciones cliente servidor usando la tecnología LINQ to SQL.

CAPÍTULO 3. REVISIÓN DE LA LITERATURA

3.1 SISTEMAS DE INFORMACIÓN

3.1.1 CONCEPTO

Un Sistema de información (SI) asociado al modelo empresarial es un conjunto de recursos técnicos, humanos y económicos, interrelacionados dinámicos y organizados en torno al objetivo de satisfacer las necesidades de la información de una organización empresarial, para la gestión y la correcta toma de decisiones (Guerras Martin, 2011).

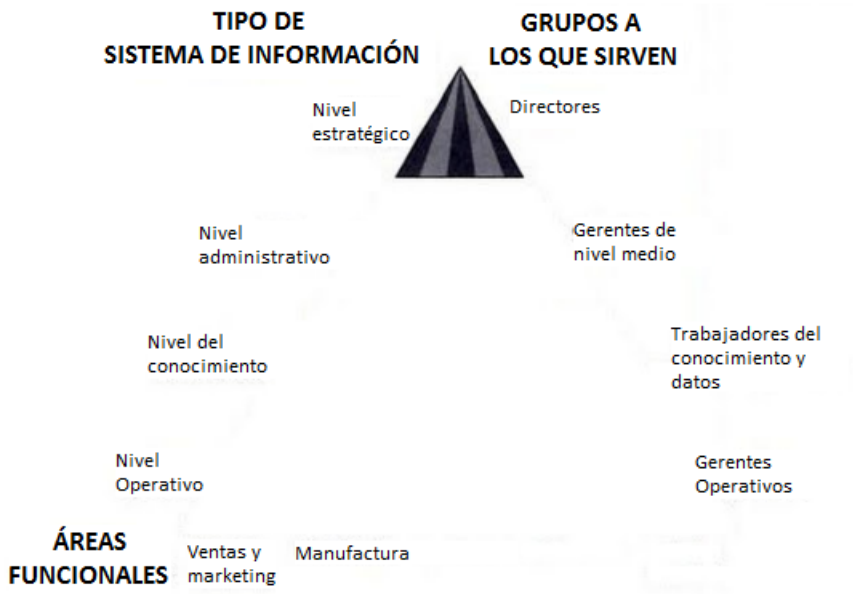
Los SI, integradores de un conjunto amplio de elementos que van más allá de tecnologías, son imprescindibles y fundamentales para cualquier organización, y no sólo por su carácter instrumental, al satisfacer las necesidades de información para la toma de decisiones, si no que se convierten en un vehículo para generar capacidades distintivas constitutivas de fuente de ventajas competitivas al combinar esta tecnología con otros recursos valiosos (Guerras Martin, 2011).

Por otro lado, un SI será eficaz si facilita la información necesaria para la organización y lo hace en el momento oportuno, y será eficiente si lo realiza con los menores recursos tecnológicos, humanos, temporales y económicos posibles (De Pablos Herederos, López, Romo y Medina, 2011).

3.1.2 TIPOS DE SISTEMA

Hay cuatro tipos de SI que dan servicio a los diferentes *niveles de organización*: sistemas a nivel operativo, sistemas a nivel conocimiento, sistemas a nivel administrativo y sistemas a nivel estratégico, los cuales se muestran en la Figura 1 y describen a continuación:

Figura 1. Tipos de sistemas de información.



Fuente: Laudon (2004).

- Sistemas a nivel operativos:

  Laudon y Laudon (2004) aseguran que apoyan a los gerentes operativos en el apoyo a las actividades y transacciones elementales de la organización como ventas, ingresos, depósitos en efectivo, nómina, decisiones de crédito y flujo de materiales en una fábrica. El objetivo principal de los sistemas a éste nivel es responder las preguntas de rutina y seguir el flujo de las transacciones a través de la organización. Preguntas como: ¿Qué pasó con el pago del Señor Gutiérrez?, ¿Cuántas partes hay en el inventario?... En general, para contestar este tipo de preguntas la información debe estar a la mano y ser actual y precisa. Un ejemplo de este tipo está en un sistema para registrar los depósitos realizados e un cajero automático o uno que lleve el registro del número de horas trabajadas cada día por los empleados de una fábrica.

- Sistemas a nivel conocimiento:

  Laudon y Laudon (2004) mencionan que se apoya a los trabajadores del conocimiento y de datos de una organización. El propósito de estos sistemas es ayudar a las empresas comerciales a integrar el nuevo conocimiento en los negocios y ayudar a la organización a controlar el flujo del trabajo de oficina. Los sistemas a nivel de conocimiento, especialmente en forma de estaciones de trabajo y sistemas de oficina, están entre las aplicaciones de crecimiento más rápido en los negocios actuales.

- Sistemas a nivel administrativos:

  Laudon y Laudon (2004) aseguran que sirven a las actividades de supervisión, control, toma de decisiones y administrativas de los gerentes de nivel medio. La pregunta principal que plantean estos sistemas es: ¿Van bien las cosas?, ya que proporciona informes periódicos más que información instantánea de operaciones. Un ejemplo es un sistema de control de reubicación que informe los costos totales de una mudanza, búsqueda de vivienda y financiamiento de vivienda para empleados de todas las divisiones de la compañía y notifique cualquier costo actual que exceda los presupuestos.

- Sistemas a nivel estratégico:

  Laudon y Laudon (2004) mencionan que estos sistemas ayudan a los directores a enfrentar y resolver aspectos estratégicos y tendencias a largo plazo, tanto en la empresa como en el entorno externo. Su función principal es compaginar los cambios en el entorno externo con la capacidad organizacional existente por ejemplo, ¿Cuáles son los niveles de empleo en cinco años?, ¿Cuáles son las tendencias a largo plazo de los costos de la industria y dónde encaja nuestra empresa?, ¿Qué productos debemos estar elaborando dentro de cinco años?

De igual manera, existen sistemas de información que corresponden a cada nivel organizacional los cuales se clasifican de la siguiente manera:

- Sistemas de apoyo a ejecutivos (ESS) en el nivel estratégico:

  Laudon y Laudon (2004) aseguran que los directores son quienes utilizan estos sistemas para tomar decisiones, ya que dan un servicio a nivel estratégico de la organización y auxilian en las decisiones no rutinarias que requieren juicio,

evaluación y comprensión porque no hay un procedimiento convenido para llegar a una solución. Crean un entorno de cómputo y comunicaciones en vez de proporcionar cualquier aplicación fija o habilidad específica. Éstos están diseñados para incorporar datos sobre eventos externos como leyes impositivas nuevas o competidoras pero incluso extraen información resumida de los MIS y DSS internos. Filtran, comprimen y dan seguimiento a datos críticos, y destaca la reducción del tiempo y esfuerzo que se requiere para que los ejecutivos obtengan información útil. Emplean software de gráficos más avanzado y tienen capacidad, por ejemplo, para entregar inmediatamente gráficos y datos provenientes de muchas fuentes a un director o una junta de directores.

- Sistemas de información gerencial (MIS) en el nivel administrativo:

  Laudon y Laudon (2004) mencionan que estos sistemas apoyan al nivel administrativo de la organización, proveyendo de informes a los gerentes y, en algunos casos, de acceso en línea al desempeño real y los registros históricos de la organización. Por lo general, se orientan casi exclusivamente a eventos internos, no a eventos externos ni del entorno. Éstos dan servicio principalmente a las funciones de planeación, control y toma de decisiones a nivel administrativo. Por lo general, para sus datos dependen de sistemas de procesamiento de transacciones subyacentes.

- Sistemas de apoyo a toma de decisiones (DSS) en el nivel administrativo:

  Laudon y Laudon (2004) aseguran que estos sistemas ayudan a los gerentes a tomar decisiones que son exclusivas, rápidamente cambiantes y no especificadas fácilmente con anticipación. Abordan problemas donde el procedimiento para llegar a una solución podría no estar predefinido con anterioridad.

  Estos sistemas pueden apoyarse en información de los TPS y de los MIS.

  Por el diseño, los DSS tienen más poder analítico que los demás sistemas ya que contienen explícitamente una variedad de modelos para el análisis de datos, o bien condensan grandes cantidades de datos de tal forma que su análisis sea sencillo para los encargados de tomar las decisiones. Están diseñados de modo que los usuarios puedan trabajar directamente con ellos por la facilidad de manejo del software y su interactividad.

- Sistemas de trabajo del conocimiento (KWS) y sistemas de oficina en el nivel conocimiento:

  Laudon y Laudon (2004) mencionan que éstos satisfacen las necesidades de información al nivel de conocimiento de la organización. Los *sistemas de trabajo* del conocimiento auxilian a los trabajadores del conocimiento, mientras que los sistemas de oficina auxilian principalmente a los trabajadores de datos (aunque también los utilizan ampliamente los trabajadores del conocimiento). Cabe destacar que los *trabajadores del conocimiento* son aquellas personas con títulos universitarios formales y que suelen ser parte de profesiones reconocidos como ingenieros, médicos, abogados y científicos los cuales se emplean principalmente en crear información y conocimientos nuevos y los trabajadores de datos tienen niveles de educación avanzada menos formales y tienden más a procesar información que a crearla como lo son las secretarias, tenedores de libros, archivistas o gerentes cuyo trabajo consiste principalmente en utilizar, manejar o distribuir la información.

  Los sistemas de oficina típicos manejan y administran documentos mediante procesamiento de texto, autoedición, digitalización de documentos y archivo digital, programación a través de calendarios electrónicos y comunicación mediante correo electrónico, correo de voz o videoconferencia.

- Sistemas de procesamiento de transacciones (TPS) en el nivel operativo:

  Laudon y Laudon (2004) mencionan que éstos sistemas se efectúan de manera computarizada y registran las transacciones diarias necesarias para dirigir negocios dando el servicio a nivel operativo de la organización. Ejemplos de ello son las entradas de pedidos de ventas, los sistemas de reservaciones en hoteles, la nómina, el registro de empleados y los embarques. Cabe destacar que hay cinco categorías funcionales de TPS: ventas y marketing, manufactura y producción, finanzas y contabilidad, recursos humanos u otros tipos de TPS que son exclusivos de una industria en particular.

  Para resumir lo anterior, se muestra en la Tabla 1 dicho contenido.

Tabla 1. Resumen de los tipos de sistemas de información.

Tipo de sistema	Entradas de información	Procesamiento	Salidas de información	Usuarios
ESS	Datos acumulados: externos, internos	Gráficas, simulaciones, interactivo	Proyecciones, respuestas a consultas	Directores
DSS	Datos de bajo volumen o bases de datos masivas optimizadas para el análisis de datos, modelos analíticos y herramientas de análisis de datos	Interactivo, simulaciones, análisis	Informes especiales, análisis de decisiones, respuestas a consultas	Profesionales, gerentes de personal
MIS	Datos resumidos de transacciones, datos de alto volumen, modelos simples	Informes de rutina, modelos simples, análisis de bajo nivel	Informe resumidos y excepciones	Gerentes de nivel medio
KWS	Especificaciones de diseño, base del conocimiento	Modelado, simulaciones	Modelos, gráficos	Profesionales, personal técnico
Sistema de oficina	Documentos, programas	Administración de documentos, programación, comunicación	Documentos, programas, correo	Oficinistas
TPS	Transacciones; eventos	Clasificación, listado, fusión, actualización	Informes detallados, listas, resúmenes	Personal de operaciones, supervisores

Fuente: Laudon (2004).

3.1.3 DIAGRAMA DE WARNIER

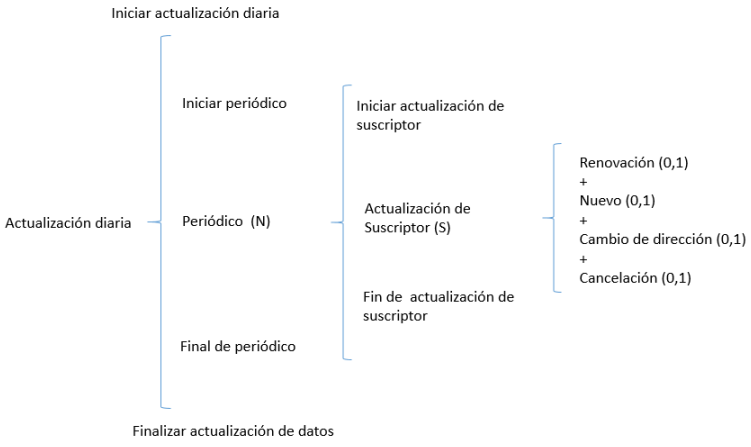
El *Desarrollo de Sistemas Estructurados de Datos* (DSED), también llamado metodología de Warnier/Orr, se basa en el trabajo pionero sobre el análisis del dominio de la información, realizado por J.D. Warnier, que consiste en representar la jerarquía de la información usando las tres construcciones de secuencia, selección y repetición, demostrando que la estructura del software puede derivarse directamente de la estructura de datos (Barranca de Areba, 2001).

El método consiste en comenzar el análisis examinando la jerarquía e información; esto es cómo se mueven los datos entre productores y consumidores de la información (Barranca de Areba, 2001).

El diagrama de Warnier es creado por Jean Dominique Warnier el cual utiliza como símbolo principal la "llave" en la cual se describen los conjuntos involucrados; en la parte izquierda se coloca el nombre del conjunto y la parte derecha los elementos que componen el conjunto. Warnier aplicó esos conceptos a las secuencias lógicas de programación, formando una técnica algorítmica, la cual es considerada como un conjunto y sus acciones son los elementos que se plasman en el mencionado diagrama (López Román, 2011).

Un ejemplo de diagrama de Warnier/Orr es el que se observa en la Figura 2.

Figura 2. Ejemplo de diagrama de Warnier/Orr.



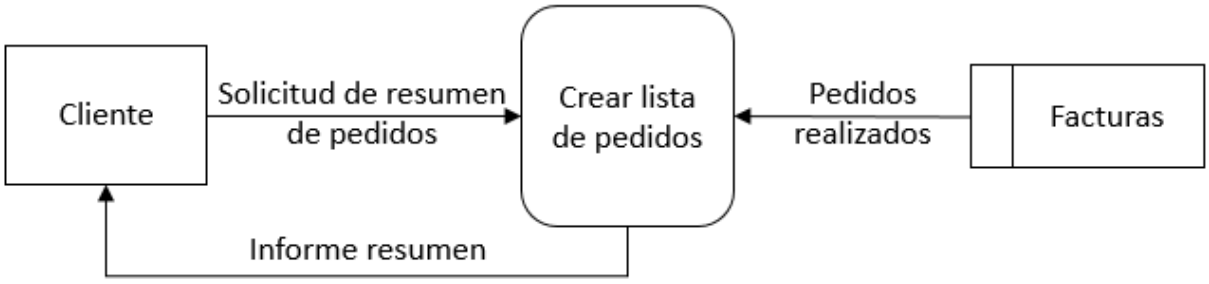
Fuente: Adaptado de Viso, E. y Pelaez, C. (2007).

3.1.4 DIAGRAMA DE FLUJO DE DATOS

El modelado de datos orientado al flujo es una de las notaciones de análisis utilizadas con mayor amplitud en la actualidad. Aunque el diagrama de flujo de datos (DFD) y los diagramas y la información relacionados no son una parte formal de UML, pueden utilizarse para complementar los diagramas de UML y proporcionar un conocimiento adicional de los requisitos y el flujo del sistema (Pressman, 2005).

El DFD tiene una visión del sistema del tipo entrada-proceso-salida. Esto es, los objetos de datos fluyen hacia el interior del software, se transforman mediante elementos de procesamiento, y los objetos de datos resultantes fluyen al exterior del software. Los objetos de datos se representan mediante flechas rotuladas y las transformaciones se representan por medio de círculos (también llamas burbujas). El DFD se representa en una forma jerárquica. Esto es, el primer modelo de flujo de datos que representa el sistema como un todo. Los diagramas de flujo subsecuentes refinan el diagrama de contexto, ya que proporcionan una cantidad creciente de detalles con cada nivel subsiguiente. Véase la Figura 3 para ejemplificar el diagrama de flujo de datos (Pressman, 2005).

Figura 3. Ejemplo de Diagrama de Flujo de datos.



Fuente: Adaptado de Fernandez Alarcón, V. (2006).

3.2 UML

3.2.1 ¿QUÉ ES?

UML es el acrónimo en inglés para *lenguaje unificado de modelado,* lo cual es una notación de modelado visual, que utiliza diagramas para mostrar distintos aspectos de un sistema. Si bien muchos destacan que UML es apto para modelar cualquier sistema, su mayor difusión y sus principales virtudes se advierten en el campo de los sistemas de software (Fontela, 2011).

Se habla de lenguaje, en cuanto a que es una herramienta de comunicación formal, con una serie de construcciones, una sintaxis y una semántica definidas. Así los elementos constructivos son diagramas y sus partes, la sintaxis es la descripción de cómo deben realizarse esos diagramas y la semántica define el significado de cada diagrama y elemento de los mismos (Fontela, 2011).

3.2.2 ANTECEDENTES

UML surgió en 1995, por iniciativa de Grady Booch, James Rumbaugh e Ivar Jacobson, tres conocidos ingenieros de software que ya habían avanzado con sus propias notaciones de modelado. Precisamente, UML se define como *unificado*, porque surgió como síntesis de los mejores elementos de las notaciones previas (Fontela, 2011).

A mediados de la década de 1990, ya se estaba inmerso en la falta de un estándar por lo que se marcó el camino para la modernización de software orientado a objetos (Fontela, 2011).

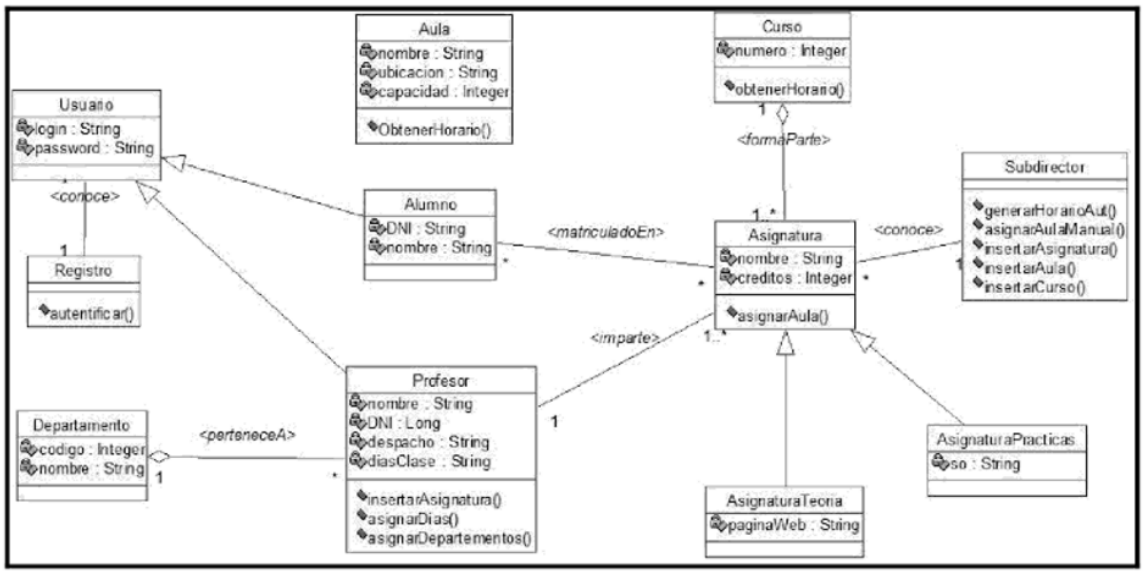
Luego UML se especificó con más rigurosidad y en 1997 se presentó la versión 1.0, que fue fue aprobada y establecida como estándar por el OMG (*Object Management Group*). De allí en más siguió evolucionando, formalizándose, creciendo y complejizándose (Fontela, 2011).

3.2.3 DIAGRAMAS DE CLASE

Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. Gráficamente, una clase se representa como un rectángulo (Booch, Rumbaugh y Jacobon, 2006).

UML también proporciona una representación gráfica de las clases como se muestra en la Figura 4. Ésta notación permite visualizar una abstracción independientemente de cualquier lenguaje de programación específico y de forma que permite resaltar las partes más importantes de una abstracción: su nombre, sus atributos y sus operaciones (Booch, Rumbaugh y Jacobon, 2006).

Figura 4. Diagrama de clases



Fuente: Coral, C., Moraga, M. A. y Piattini, M. (2010).

Booch, Rumbaugh y Jacobon (2006), mencionan como características de los diagramas de clases las siguientes:

- Nombres: Cada clase ha de tener un nombre que las distingue de las demás clases, la cual está conformada por una cadena de texto (texto formado por cualquier número de letras, números y ciertos signos de puntuación, excepto dos puntos ya que ese se utiliza para separar el nombre de la clase y el paquete en el que se encuentra) a la cual

se le denomina *cadena simple*, pero si se conforma de un nombre de la clase precedido por el nombre del paquete en el que se encuentra se le denomina *nombre calificado*.

- Atributos: Es una propiedad de una clase identificada por un nombre, la cual representa alguna propiedad del elemento que se está modelando que es compartida por todos los objetos de esa clase. Una clase puede tener un número de atributos o no tener ninguno.
- Operaciones: Es la implementación de un servicio que puede ser requerido a cualquier objeto de la clase para que muestre su comportamiento. En otras palabras, es una abstracción de algo que se puede hacer a un objeto y que es compartido por todos los objetos de la clase.
- Relaciones: Es una conexión entre elementos. En el modelado orientado a objetos, las tres relaciones más importantes son las *dependencias, las generalizaciones y las asociaciones*. Gráficamente, una relación se representa como una línea usándose diferentes tipos de línea para diferencias los tipos de relaciones.
    - *Dependencia*: Es una relación de uso que declara que un elemento utiliza la información y los servicios de otro elemento, pero no necesariamente a la inversa. Gráficamente, una dependencia se representa como una línea discontinua dirigida hacia el elemento del cual depende. Éstas se usan cuando se quiera indicar que un elemento utiliza a otro.
    - *Generalización*: Es una relación entre un elemento general (llamado superclase o padre) y un caso más específico de ese elemento (llamado subclase o hijo) la cual se utiliza cuando se quiera mostrar relaciones padre/hijo. Un objeto de la clase hija se puede asociar a una variable o un parámetro cuyo tipo venga dado por el padre, pero no a la inversa. En otras palabras, la generalización significa que el hijo puede sustituir la declaración del padre. Un hijo hereda las propiedades de sus padres, especialmente atributos y operaciones, y en ocasiones el hijo añade atributos y operaciones. Una implementación de operación por parte del hijo redefine la implementación de la misma operación en el padre, lo cual se conoce como *Polimorfismo*.

Gráficamente, la generalización se representa como una línea dirigida continua, con una gran punta de flecha vacía, apuntando al padre.

- *Asociación*: Es una relación estructura que especifica que los objetos de un elemento están conectados con los objetos de otro. Dada una asociación entre dos clases, se puede establecer una relación desde un objeto de una clase hasta algunos objetos de la otra clase. Se pueden dar que en las asociaciones un objeto se pueda conectar con otros objetos de la misma clase, por lo que si se conecta con dos clases se dice *asociación binaria* y si son a más de dos clases (que no es muy común) se denomina *asociaciones n-arias*.

  Gráficamente una asociación se representa como una línea continua que conecta la misma o diferentes clases.

  Además de la forma básica hay cuatro elementos que se aplican a las asociaciones, las cuales son:

  - Nombre: La asociación puede tener un nombre, que se utiliza para describir la naturaleza de la relación y para que no haya ambigüedad en su significado, se puede dar una dirección al nombre por medio de una flecha que apunte en la dirección en la que se pretende que se lea el nombre.
  - Rol: Cuando una relación participa en una relación, tiene un rol específico que juega en esa relación, el cual es simplemente la cara que la clase de un extremo de la asociación presenta a la clase del otro extremo.

    Cabe destacar que la misma clase puede jugar el mismo o diferentes roles en otras asociaciones.
  - Multiplicidad: En muchas situaciones del modelado, es importante señalar cuantos objetos pueden conectarse a través de una instancia de una asociación. Éste *cuantos* se denomina multiplicidad del rol de la asociación, y representa un rango de enteros que especifican el tamaño posible del conjunto de objetos relacionados. Ésta se describe como una expresión con un valor mínimo y un valor máximo, que pueden ser iguales utilizándose sólo dos puntos para

separar ambos valores. Cuando se indica una multiplicidad en un extremo de una asociación, se está especificando cuántos objetos de la clase de ese extremo puede haber para cada objeto de la clase en el otro extremo. La multiplicidad se puede indicar con exactamente uno (1), cero o uno (0..1), muchos (0..*), o uno o más (1..*). De igual manera se puede dar un rango de enteros (como 2..5) incluso con un número exacto (por ejemplo 3, lo cual equivale a 3..3).

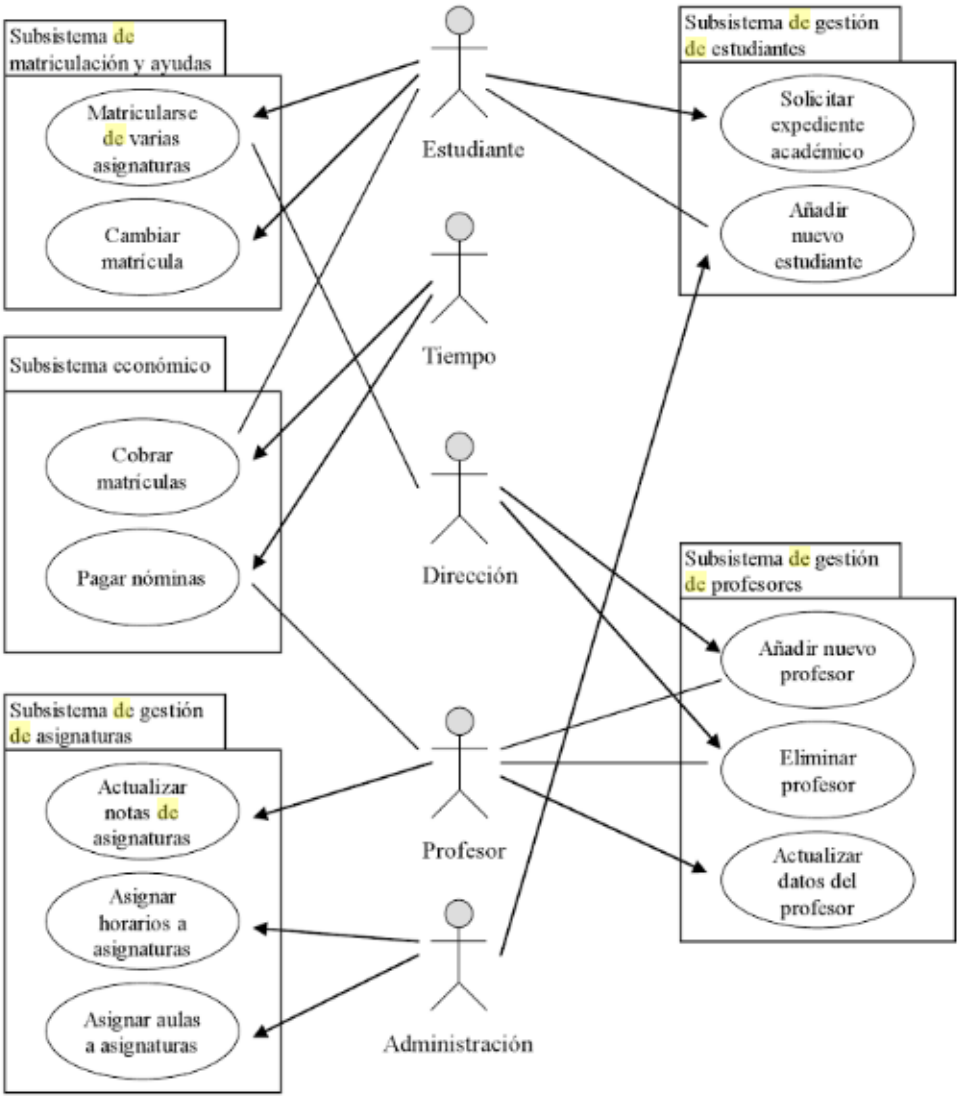
- Agregación: Una asociación normal entre dos clases representa una relación estructural entre iguales, es decir, ambas clases están conceptualmente en el mismo nivel, sin ser ninguna más importante que la otra. A veces, se desea modelar una relación *todo/parte*, en la cual un clase representa una cosa grande (el todo), que consta de elementos más pequeños (las partes). Éste tipo de relación se le denomina agregación, el cual, gráficamente, se especifica añadiendo una asociación normal con un rombo vacío en la parte del todo.

3.2.4 DIAGRAMA DE CASOS DE USO

El modelo de los casos de uso suele servir, entre otras cosas, para delimitar el alcance del sistema, esbozar quiénes interactuarán con el sistema, a modo de actores, cuáles son las funcionalidades esperadas y capturar el primer glosario de términos de dominio. Y, sobre todo, para validar los requisitos del cliente (Fontela, 2011).

Estos diagramas no especifican el comportamiento de los casos de uso, sino solamente relaciones entre distintos casos de uso y viceversa, por lo que son clasificados como diagramas estructurales y no de comportamiento. En la Figura 5 se muestra un ejemplo de un diagrama de casos de uso (Fontela, 2011).

Figura 5. Ejemplo de diagrama de casos de uso.



Fuente: Fernandez Alarcón, V. (2006).

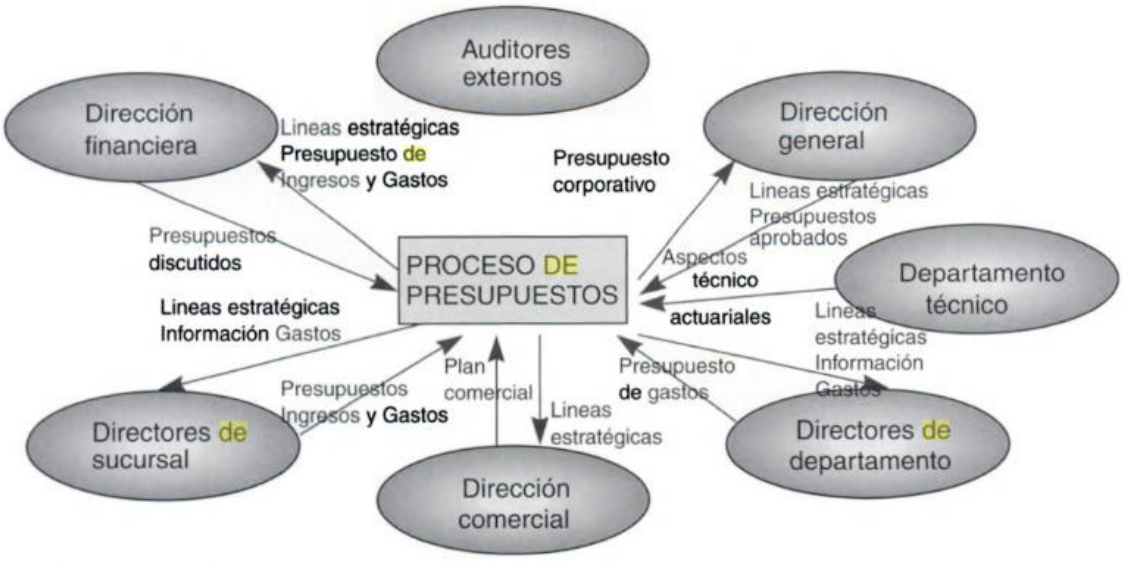
3.2.5 DIAGRAMA DE CONTEXTO

Un diagrama de contexto refleja las entidades externas que interactúan con un sistema o proceso y los flujos (de datos, información, etc.) entre las entidades y el sistema o proceso (Arjona, 1999).

Alonso, Martínez y Segovia Perez (2005) mencionan que el diagrama de contexto es una herramienta importante porque enfatiza:

- Las entidades externas (personas, empresas y otros sistemas) con los que se comunica el sistema.
- Los datos que el sistema recibe desde el exterior para su proceso y los datos que genera y envía al exterior.
- Los almacenes de datos externos del sistema, que se crean fuera del sistema y el los utiliza o que los crea el propio sistema y son usados fuera.
- Las fronteras entre el sistema a desarrollar y el resto del mundo. Como ejemplo véase la Figura 6.

Figura 6. Ejemplo de diagrama de contexto.



Fuente: Arjona Torres, M (1999).

3.2.6 DIAGRAMAS DE COLABORACIÓN O COMUNICACIÓN

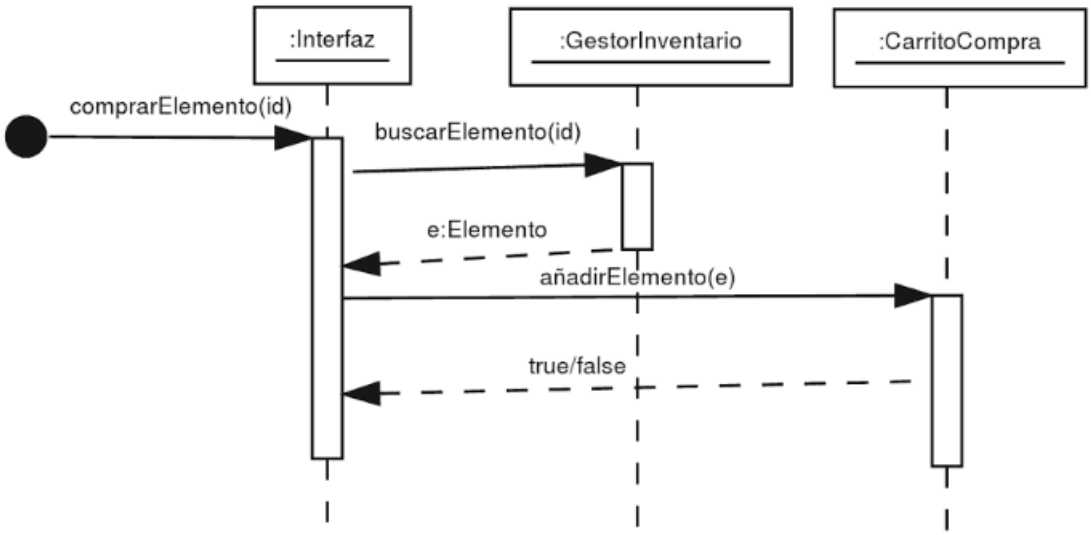
Es un diagrama de interacción que resalta la estructuración de los objetos que envían y reciben mensajes, por lo que describen la vista dinámica de un sistema. Éste diagrama muestra un conjunto de roles, enlaces entre ellos y los mensajes enviados y recibidos por las instancias que interpretan esos roles (Booch, Rumbaugh y Jacobon, 2006).

Booch, Rumbaugh y Jacobon (2006) comentan que los diagramas de comunicación tienen dos características que los distinguen de los diagramas de secuencia:

1. El camino: El camino se dibuja haciéndolo corresponder con una asociación, variables locales, parámetros, variables globales o accesos propios. Un camino representa una fuente de conocimiento de un objeto.
2. Número de secuencia: Se utiliza para indicar la ordenación temporal de un mensaje, se precede de un número (comenzando por el mensaje número 1), que se incrementa secuencialmente por cada nuevo mensaje en el flujo de control (2, 3 etc.). Para representar el anidamiento, se utiliza la numeración decimal de Dewey (1 es el primer mensaje; 1.1 es el primer mensaje entro de 1). También hay que tener en cuenta que se pueden mostrar varios mensajes a través del mismo enlace (posiblemente enviados desde distintas direcciones), y cada uno tendrá un número de secuencia único.

Como ejemplo del diagrama de comunicación véase la Figura 7.

Figura 7. Ejemplo del diagrama de colaboración o comunicación.



Fuente: Berengel Gómez, J.L. (2016).

3.2.7 DIAGRAMA DE ESTADOS

El diagrama de estados de UML es una herramienta que sirve para modelar cómo afecta un escenario a los estados que un objeto toma, en conjunto con los eventos que provocan las transiciones de estado. Este muestra los cambios de estado que sufre un objeto a través del tiempo y se puede modelar toda la vida del objeto o su existencia dentro de un escenario particular. También es habitual denominarlo *diagrama de máquina de estados*, o *máquina de estados* (Fontela, 2011).

Se pueden utilizar para modelar objetos reactivos, es decir, aquellos objetos para los cuales la mejor forma de caracterizar su comportamiento sea señalar cuál es su comportamiento frente a estímulos provenientes desde fuera de su contexto, o aquellos que están ociosos hasta el momento en que reciben un evento (Fontela, 2011).

Los cambios de estados que sufren los objetos se deben a estímulos o mensajes recibidos de otros objetos, cuyos efectos UML llama *eventos* (Fontela, 2011).

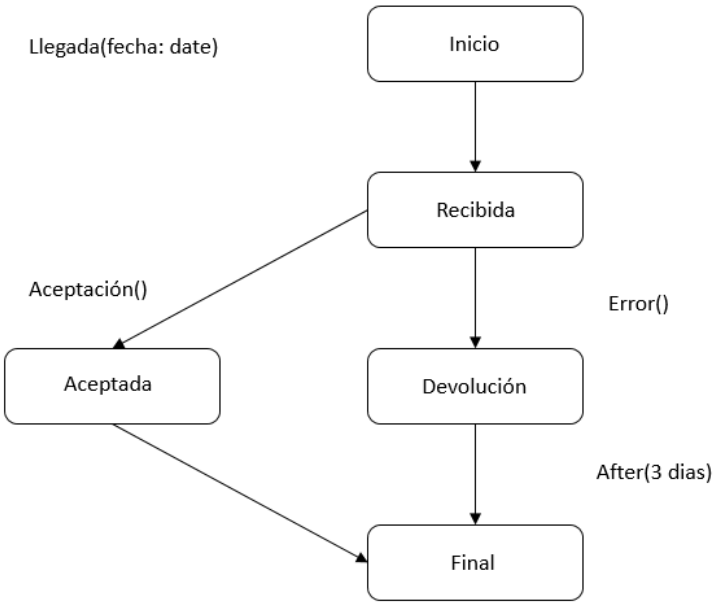
Un evento indica la aparición de un estímulo que puede disparar una transición de estados. Es la especificación de un acontecimiento significativo, como una señal recibida, un cambio de estado o el paso de un intervalo de tiempo (Fontela, 2011).

A diferencia las transiciones entre estados, es el paso de un estado a otro, es decir, un objeto que esté en primer estado realizará ciertas acciones y entrará en un segundo estado cuando ocurra algún evento especificado y se satisfagan ciertas condiciones (Fontela, 2011).

La representación gráfica consiste en un grafo o red con nodos para los estados y arcos para las transiciones, además de un texto en correspondencia con los arcos que describen eventos condiciones y acciones de las transiciones. En la Figura 8 se pueden observar los elementos distintivos de diagrama de estados:

- Los estados por los que pasa el objeto se representan con rectángulos de puntas redondeadas.
- Los nodos inicial y final.
- Las transiciones entre estados se representan con flechas.

Figura 8. Ejemplo de diagrama de estado.



Fuente: Adaptado de Campderrich Falgueras, B. (2003).

3.3 BASES DE DATOS

3.3.1 SQL SERVER

El nombre de SQL se deriva de *Structured Query Language*. Originalmente, SQL se llamaba SEQUEL (de *Structured English QUEry Language*) y fue diseñado e implementado por IBM Research como interfaz para un sistema experimental de base de datos relacional llamado SYSTEM R. Ahora SQL es el lenguaje estándar de los SGBD relacionales (Elmasri y Navathe, 2002).

SQL es un lenguaje de base de datos global; cuenta con enunciados de definición, consulta, actualización y eliminación de datos. Así pues, es tanto un LDD como un LMD. Además cuenta con mecanismos para definir vistas de la bases de datos, para especificar seguridad y autorización, para definir restricciones de integridad, y para especificar controles de transacciones. También tiene como reglas para insertar sentencias de SQL en lenguajes de programación de propósito general como C o PASCAL (Elmasri y Navathe, 2002).

Características de Sql Server Management Studio (s.f) puntualiza que Sql Server ofrece como características las siguientes:

- Compatibilidad con SQL Server y su mayor parte de sus tareas administrativas.
- Un entorno único integrado para la administración del Motor de base de datos de SQL Server.
- Cuadros de diálogo para administrar objetos de Motor de base de datos de SQL Server, Analysis Services y Reporting Services, permitiendo realizar las acciones en el momento, enviarlas a un editor de código o escribirlas en script para ejecutarlas posteriormente.
- Cuadros de diálogo de tamaño no fijo que permiten obtener acceso a varias herramientas mientras un cuadro de diálogo está abierto.
- Un cuadro de diálogo común de programación que permite realizar acciones de los cuadros de diálogo de administración en otro momento.
- Importación y exportación del registro de servidor de SQL Server Management Studio desde ese entorno a otro.

- Guardado o impresión de archivos de presentación XML o de interbloqueo generados por SQL Server Profiler, así como su revisión posterior o envío a los administradores para su análisis.
- Un nuevo cuadro de mensaje de error e informativo, permite enviar a Microsoft un comentario sobre los mensajes, copiarlos en el Portapapeles y enviar mensajes por correo electrónico al equipo de soporte.
- Un explorador Web integrado para una rápida exploración de MSDN o la Ayuda en pantalla.
- Integración de la Ayuda de comunidades en línea.
- Un tutorial sobre SQL Server Management Studio para aprovechar las ventajas de características nuevas y favorecer su productividad.
- Un nuevo monitor de actividad con filtro y actualización automática.
- Interfaces de Correo electrónico de base de datos integradas.

Sentencias

- CREATE DATABASE

  Gabillaud (2015) comenta que para crear una base de datos se tiene que estar conectado como administrador de sistemas o se debe tener permisos para utilizar la instrucción CREATE DATABASE, así como estar dentro de la base de datos del sistema maestro. Como primer paso se debe crear un objeto DATABASE tomando un nombre único de la base de datos en la instancia SQL Server, el cual está limitado a 128 caracteres y debe respetar las reglas de construcción de identificadores. Ya una vez creada la base de datos, ésta contendrá el resto de los objetos:

  - Catálogo de base de datos.
  - Objetos de usuario (tablas, vistas, triggers y procedimientos).
  - Índices, tipos de datos y restricciones de integridad.
  - Archivo de traza de transacciones.

  La sintaxis completa de la instrucción para crear bases de datos es la siguiente:

  CREATE DATABASE nombreBase [ON [PRIMARY][([NAME = nombreLogico,] FILENAME = 'nombreFísico' [, SIZE = tamaño] [, MAXSIZE = {tamañoMaximo |

UNLIMITED}][, FILEGROWTH = valorIncremento]) [,…]] [LOG ON {archivo}][COLLATE nombreClasificacion] [FOR ATTACH | FOR ATTACH_REBUILD_LOG]

En donde:

NOMBRE: es el nombre lógico del archivo

FILENAME: es la ubicación y nombre físico del archivo

SIZE: es el tamaño inicial del archivo en megabytes (MB) o kilobytes (KB). Cuando no se especifica este valor, se considera por default que es en MB. De cualquier forma, el tamaño se expresa con un valor entero.

MAXSIZE: es el tamaño máximo del archivo en KB o MB. Si éste valor no se especifica, el tamaño del archivo estará limitado por el espacio disponible en disco, el cual, por defecto estará asignado en MB.

UNLIMITED: ilimitado (sólo por el espacio disponible en disco).

FILEGROWTH: indica el paso del incremento para el tamaño del archivo (sin superar el valor máximo).

LOG ON: es la ubicación del archivo de traza de las transacciones, la cual almacena las modificaciones en los datos (con INSERT, UPDATE o DELETE). Este archivo sirve para recuperar los datos en caso de producirse algún error.

COLLATE: es la clasificación (Windows o SQL Server) por defecto de la base de datos la cual es asignada por default a la instancia SQL Server.

FOR ATTACH: permite crear bases de datos utilizando archivos de versiones anteriores de SQL Server con un nivel 90 de compatibilidad (SQL 2005, como mínimo).

FOR ATTACH_REBUILD_LOG: es posible crear la base de datos adjuntándole archivos de datos (mdf y ndf), pero no de archivos de traza, ya que estos se crean vacíos. En caso que se adjunten estos archivos, es necesario hacer una copia de seguridad y planificar los procesos de copia de seguridad. NOTA: No se podrán

utilizar las copias de seguridad que se hayan realizado antes de adjuntar los archivos, ya que las secuencias de traza no corresponderán.

- USE

  Gabillaud (2015) aseguran que es la instrucción que se debe ejecutar para tener permisos de CONTROL dentro de una instancia de SQL Server. En el siguiente ejemplo de utiliza para utilizar una base de datos y así poder consultar, insertar, eliminar y actualizar datos.

  USE Sistema

  NOTA: En el caso de eliminación de una base de datos, se debe cambiar el contexto de otra forma, marcará error.

- CREATE TABLE

  Elmasri y Navathe (2002) comentan que sirve para especificar una nueva relación dándole un nombre y especificando sus atributos y restricciones. Los atributos se especifican primero y a cada uno se le dan los siguientes valores:

  - Un *nombre*
  - Un *tipo de dato* para especificar su dominio de valores.

    Entre los tipos de datos disponibles para atributos están los numéricos, cadena de caracteres, cadena de bits, fecha y hora; los cuales se describen en la Tabla 2.

Tabla 2. Clasificación de tipos de datos.

Tipo de dato	Clasificación	Nombre
Numérico	Números enteros (de diversos tamaños)	INTEGER, INT, SMALLINT
	Números reales (de diversas precisiones)	FLOAT, REAL, DOUBLE PRECISION
	Números con formato	DECIMAL(precisión, escala), DEC (precisión, escala)
	Longitud fija	CHAR(n) o CHARACTER(n)

Cadena de caracteres		*n= número de caracteres*
	Longitud variable	VARCHAR(n), CHARVARYING(n) o CHARACTER VARYING(n) *n= número máximo de caracteres*
Cadena de bits	Longitud fija	BIT(n) *n= número de bits*
	Longitud variable	BIT VARYING(n) *n= número de bits*
Fecha		DATE (YEAR,MONTH, DAY = YYYY-MM-DD)
Hora	Time	TIME (HOUR,MINUTE,SECOND = HH:MM:SS), TIME(i) *i = precisión en fracciones de segundo* TIME WITH TIME ZONE.
	Marca de tiempo	TIMESTAMP
	Intervalo	INTERVAL

Fuente: Elmasri y Navathe (2002).

- Cualquier *restricción* del atributo y valores por omisión:

  Dado que SQL permite NULL como valor para un atributo, se puede especificar la *restricción* NOT NULL si es que no se permite el valor nulo para un determinado atributo. Esta restricción se debe especificar siempre para los atributos de clave primaria de toda relación, así como para cualquier otro atributo cuyos valores no deban ser nulos. También es posible definir un *valor por omisión* para un atributo añadiendo la cláusula DEFAULT <valor> a la definición de un atributo y si no se especifica la cláusula DEFAULT, el valor de omisión por defecto (!) es NULL.

  Siguiendo las especificaciones de los atributos (o columnas), podemos especificar *restricciones de tabla*, incluidas las de clave y de integridad referencial. La cláusula PRIMARY KEY especifica uno o más atributos que

constituyen la clave primaria de una relación. La cláusula UNIQUE especifica claves alternativas (secundarias). La integridad referencial se especifica mediante la cláusula FOREIGN KEY, y con esto cuando se insertan o eliminan tuplas o cuando se modifica un valor de atributo de clave externa se ve violada la restricción de integridad referencial.

Se pueden especificar las restricciones de clave, de integridad de entidad e integridad referencial, dentro de la sentencia CREATE TABLE o también se puede hacer después, usando la instrucción ALTER TABLE. La siguiente sentencia se muestra como ejemplo de la instrucción CREATE TABLE y sus características.

```
CREATE TABLE Articulos(
IdCodigo VARCHAR(10) NOT NULL,
IdTipo INT,
Descripcion VARCHAR(50),
Precio MONEY,
PRIMARY KEY (IdCodigo),
FOREIGN KEY (IdTipo) REFERENCES Tipos(IdTipo));
```

- INSERT

  Pérez López (2007) explica que las sentencias del lenguaje de modificación de datos se usan en la interrogación y manipulación de datos en esquemas de bases de datos existentes. Entre estas sentencias destacan las que se utilizan para realizar actualizaciones en las bases de datos y cuyo exponentes más importantes son las sentencias INSERT, UPDATE y DELETE.

  En el caso de la agregación de nuevas filas de datos a las tablas ya existentes, la sentencia básica es INSERT, la cual agrega una o más filas nuevas a una tabla. La sintaxis de esta sentencia simplificada es:

  INSERT [INTO] tabla_o_vista [(lista_de_columnas)] valores_de_datos

  La sentencia hace que *valores_de_datos* se inserte como una o más filas de la *tabla_o_vista* que se nombra. El argumento *lista_de_columnas* es una lista separada por comas de los nombres de tablas que se pueden utilizar para especificar las

columnas para las que se suministran datos. Si no se especifica *lista_de_columnas*, todas las columnas de la tabla o vista recibirán datos.

Cuando una *lista_de_columnas* no enumera todas las columnas de la tabla o vista, se inserta un valor NULL (o el valor predeterminado si se ha definido alguno para la columna) en aquellas columnas que no se hayan enumerado en la lista. Todas las columnas no especificadas en la lista de columnas deben permitir valores NULL o tener un valor predeterminado asignado.

Los *valores_de_datos* suministrados deben corresponderse con la lista de columnas. El número de valores de datos debe ser el mismo que el número que el número de columnas y el tipo de datos, precisión y escala de cada valor de datos debe coincidir con los de la columna correspondiente. Hay dos maneras de especificar los valores de datos:

- Utilizar una cláusula VALUES para especificar los valores de datos para una fila:

  INSERT [INTO] tabla_o_vista VALUES (lista de valores_de_datos)

  Si no se especifica una lista de columnas, los valores deberán especificarse en la misma secuencia que las columnas de la tabla o vista. La lista de valores y la lista de columnas deben contener el mismo número de elementos y el tipo de dato de cada valor debe ser compatible con el tipo de datos de la columna correspondiente, o en caso contrario se producirá un error. La siguiente sentencia es ejemplo de un INSERT con VALUES omitiendo los datos de la tabla:

  ```
  INSERT INTO Pedidos VALUES (1,'cl1','2016-09-01',4000,'Pendiente');
  ```

- Utilizar una subconsulta SELECT para especificar los calores de datos para una o más filas:

  INSERT INTO tabla_o_vista SELECT columnas FROM tablas

- SELECT

Elmasri y Navathe (2002) la definen como una sentencia básica para recuperar información de una base de datos. La forma básica de la sentencia SELECT, en ocasiones denominada correspondencia o bloque *select-from-where*, consta de tres clausulas SELECT, FROM y WHERE y tiene la siguiente forma:

SELECT <lista de atributos>

FROM <lista de tablas>

WHERE <condición>

Donde:

< lista de atributos> es una lista de nombres y atributos cuyos valores vas a ser recuperados por la consulta.

< lista de tablas> es una lista de nombres de las relaciones necesarias para procesar la consulta.

< condición> es una expresión condicional (booleana) que identifica las tuplas que van a ser recuperadas por la consulta.

En la siguiente consulta se ejemplifica el uso de dichas clausulas:

```
SELECT IdCodigo, Precio, Cantidad
FROM DetallePedido
WHERE IdPedido = @IdPedido
```

La cláusula SELECT de SQL especifica los *atributos de proyección*, y la cláusula WHERE especifica la *condición de especificación*.

En SQL se puede usar el mismo nombre para dos (o más) atributos siempre que los atributos pertenezcan a diferentes relaciones. Si es éste el caso, y una consulta hace referencia a dos o más atributos con el mismo nombre, se debe *calificar* el nombre del atributo con el nombre de la relación, a fin de evitar ambigüedades. Esto se hace anteponiendo el nombre de la relación al nombre del atributo y separando los dos mediante un punto. Para ilustrar esto, se muestra una consulta haciendo uso de dichos elementos:

```
SELECT DetallePedido.IdPedido, Pedidos.IdPedido, DetallePedido.Precio,
DetallePedido.Cantidad
FROM DetallePedido INNER JOIN Pedidos ON DetallePedido.IdPedido =
Pedidos.IdPedido
WHERE DetallePedido.IdPedido = 1
```

Para evitar que exista ambigüedad en el caso de consultas que hagan referencia dos veces a la misma relación, se puede declarar nombres de relación alternativos, llamados alias o variables de tuplas, el cual puede ir tras la palabra clave AS, renombrando los atributos de la relación dentro de la consulta, tal como se muestra en la siguiente consulta:

```
SELECT Pedidos.IdPedido AS idPedido,
        Pedidos.IdCuenta AS idCuenta,
        Pedidos.Fecha AS fecha,
        Pedidos.Subtotal AS subtotal,
        Pedidos.Estado AS estado
FROM Pedidos
JOIN Clientes ON Pedidos.IdCuenta = Clientes.IdCuenta
```

En SQL también se presenta la *omisión de la cláusula* WHERE la cual indica una selección incondicional de tuplas, por lo tanto, todas las tuplas de la relación especificada en la cláusula FROM son aceptadas y seleccionadas para el resultado de la consulta. Si se especifica más de una relación en la cláusula FROM y no hay clausula WHERE, se selecciona el producto cartesiano (todas las posibles combinaciones de tuplas) de esas relaciones. Para recuperar los valores de todos los atributos de las tuplas seleccionadas, no se tiene que enumerar los nombres de los atributos explícitamente en SQL, basta con especificar un asterisco (*), que significa todos los atributos. Un ejemplo de esto, es la consulta que se muestra a continuación:

```
SELECT * FROM Pedidos
```

SQL permite al usuario ordenar las tuplas del resultado de una consulta según los valores de uno o más atributos, empleando la cláusula ORDER BY. El orden por omisión es el ascendente. Se puede incluir la palabra clave DESC si se quiere que los valores queden ordenados en forma descendente. La palabra clave ASC sirve para especificar explícitamente el orden ascendente.

Pérez López (2007) ejemplifica en la siguiente consulta el uso de la cláusula ORDER BY:

```
SELECT IdPedido from pedidos ORDER BY IdPedido ASC
```

Las consultas multitabla o JOIN, también denominadas combinaciones o composiciones, permiten recuperar datos de dos tablas o más según las relaciones lógicas entre ellas. Las combinaciones indican cómo debería utilizar SQL Server los datos de una tabla para seleccionar las filas de otra tabla.

Una condición de combinación (o composición) define la forma en la que las tablas se relacionan en una consulta al:

- Especificar la columna de cada tabla que debe usarse para la combinación. Una condición de combinación típica especifica una clave externa de una tabla y su clave asociada a otra tabla.
- Especificar un operador lógico (=, <>, etc.) para usarlo en los valores de comparación de las columnas.

En SQL, la cláusula INNER JOIN combina registros de dos tablas siempre que existan valores coincidentes en un campo común. Se habla, de un JOIN normal que admite la sintaxis alternativa que se indica a continuación en la cláusula FROM (Pérez López, 2007).

Sintaxis:

FROM tabla1 INNER JOIN tabla2 ON tabla1.campo1 operadorComparación tabla2.campo2.

Véase como ejemplo la siguiente consulta que hace uso de la cláusula INNER JOIN:

```
SELECT Pedidos.IdPedido, Pedidos.IdCuenta, Pedidos.Estado, Pedidos.Fecha,P
edidos.Subtotal,Clientes.Nombre FROM Pedidos INNER JOIN Clientes
On Pedidos.IdCuenta = Clientes.IdCuenta WHERE IdPedido = 1
```

La cláusula TOP permite extraer solo los primeros registros de un resultado. Esta cláusula está disponible para las instrucciones SELECT, INSERT, UPDATE, DELETE y MERGE.

Gabillaud (2015) comenta que el número de registros devueltos se especifica como argumento de la cláusula TOP. Este número se puede expresar como un valor o un

porcentaje. Tanto es un caso como en el otro es importante no subestimar las nociones de redondeo e igualdad de criterios de ordenación.

La cláusula TOP devuelve siempre un número entero de registros. Por lo tanto, el valor calculado del número de registros que se tienen que devolver siempre se redondea al entero inmediatamente superior.

Otro caso particular que hay que tener en cuenta es de las igualdades. El uso de la cláusula TOP se asocia muy habitualmente a la cláusula ORDER BY para ordenar los datos según el orden deseado.

En el siguiente ejemplo se usa la cláusula TOP para saber la cantidad total de registros en la columna *IdPedido* ordenados descendentemente de la tabla *Pedidos*, lo cual regresará un sólo número entero:

```
SELECT TOP 1 IdPedido FROM Pedidos ORDER BY IdPedido DESC
```

- DROP PROC

Gabillaud (2015) declara que la sentencia DROP se utiliza para borrar totalmente una tabla, índice, procedimiento almacenado o base de datos, llevándose con esto, sus registros. Por ejemplo, si se quiere eliminar completamente un procedimiento almacenado llamado *LISTADOPEDIDOS* se ejecuta la siguiente instrucción:

```
DROP PROC LISTADOPEDIDOS
```

- OBJECT_ID

Jones, Stephens, Plew, Garrett y Kriegel (2005) mencionan que la sentencia *object_id* es una función que regresa un objeto valido de la base de datos especificado desde un nombre de objeto válido. Estos objetos pueden ser tablas, procedimientos almacenados, vistas, triggers, un valor predeterminado o una regla. El nombre puede ser completamente cualificado como <server>.<database>.<dbo>.<objectName> y podrían ir entre comillas simples o dobles. Otra de las funcionalidades es determinar si un objeto ya existe, por lo que devuelve su ID.

En el siguiente ejemplo comprueba la existencia de un procedimiento almacenado al verificar si la tabla tiene un ID de objeto, si es así, la elimina y si no existe, no ejecuta la eliminación.

```
IF OBJECT_ID('LISTADOPEDIDOS') IS NOT NULL
        DROP PROC LISTADOPEDIDOS
GO
```

- BEGIN-END

Cebollero, Coles, Natarajan, Bruchez y Shaw (2015) explican que SQL utiliza *begin-end* como palabras clave para agrupar múltiples declaraciones o instrucciones juntas en un mismo bloque. Estas sentencias no alteran el orden de ejecución de las declaraciones que contiene, ni definen una transacción atómica o limitan el alcance así como no realizan ninguna función que no sea la definición de un simple agrupamiento de sentencias en SQL. Esto puede acelerar el desarrollo y depurar de una manera más fácil, especialmente si se escriben scripts más complejos.

En el caso del uso de las sentencias *begin-end* dentro de un *stored procedure*, no es realmente necesario utilizarlas, pero da una facilidad de delimitación del cuerpo, ya que es solo a referencia del estilo de codificación, lo cual no afecta el rendimiento o funcionamiento del procedimiento almacenado. Un ejemplo de esto puede ser el siguiente código:

```
CREATE PROC LISTADOPEDIDOS
AS
BEGIN
        SELECT Pedidos.IdPedido AS idPedido,
                Pedidos.IdCuenta AS idCuenta,
                Pedidos.Fecha AS fecha,
                Pedidos.Subtotal AS subtotal,
                Pedidos.Estado AS estado
                FROM Pedidos
                JOIN Clientes ON Pedidos.IdCuenta = Clientes.IdCuenta
END
```

GO

- CREATE PROC

  Gabillaud (2013) declara los procedimientos almacenados o también llamados *stored procedures* como objetos correspondientes a un conjunto de instrucciones LMD, las cuales se ejecutan con la llamada a la instrucción EXECUTE, interactuando con la base de datos mediante argumentos, devolución de valores y ejecución remota, teniendo sus permisos de acceso. Éstos se almacenan en la caché de memoria, compilados después de la primera ejecución aumentando de esta forma el rendimiento en las próximas ejecuciones.

  Dentro de SQL Server, los procedimientos almacenados se definen como una sucesión de instrucciones Transact SQL, las cuales son almacenados en su respectiva base de datos e identificada por su nombre y al momento de su ejecución realizará las acciones contenidas en los mismos y llevará el resultado al entorno que ha solicitado la ejecución del procedimiento.

  Como algunos de los usos que se le dan a los procedimientos almacenados se mencionan los siguientes:

  - Encadenamiento de instrucciones.
  - Incremento de rendimiento.
  - Seguridad en la ejecución.
  - Manipulación de los datos del sistema.
  - Aplicación de reglas de la compañía.
  - Tratamientos en cascada. (Gabillaud, 2013).

  La sintaxis para la creación de los procedimientos almacenados es la siguiente:

  CREATE PROC[EDURE] nombre[; numero][(argumento1[,…])] [{FOR REPLICATION | WITH RECOMPILE}]{WITH ENCRYPTION} AS instrucciones.

  Donde:

  nombre: es el nombre del objeto, único en la base de datos el cual va precedido por un signo # si es temporal y ## si es global.

  numero: es el número de orden de los procedimientos que presenten el mismo nombre.

Argumento1,…: argumento con el formato @nombreTipo[VARYING][= valor][OUTPUT] el cual permite especificar un argumento devuelto por el procedimiento.

FOR REPLICATION: indica que el procedimiento se utilizará durante la replicación.

WITH RECOMPILE: indica que el procedimiento se compilará en cada ejecución.

WITH ENCRYPTION: permite encriptar el código en las tablas del sistema.

Como ejemplo del uso de procedimientos almacenados, se muestra la siguiente instrucción:

```
CREATE PROC LISTADOPEDIDOS
AS
BEGIN
SELECT Pedidos.IdPedido AS idPedido,
        Pedidos.IdCuenta AS idCuenta,
        Pedidos.Fecha AS fecha,
        Pedidos.Subtotal AS subtotal,
        Pedidos.Estado AS estado
        FROM Pedidos
        JOIN Clientes ON Pedidos.IdCuenta = Clientes.IdCuenta
END
GO
```

- DELETE

Pérez López (2007) explica que en el caso de la eliminación de filas de datos en tablas ya existentes, la sentencia básica es DELETE, la cual quita una o varias filas de una tabla o vista. Una forma simplificada de la sintaxis de DELETE es:

DELETE FROM tabla_o_vista WHERE condición

El argumento *tabla_o_vista* nombra la tabla o vista de la que se van a eliminar las filas. Se eliminan todas las filas de *tabla_o_vista* que reúnan los requisitos de la *condición* de búsqueda de la cláusula WHERE. Si no se especifica una clausula WHERE, se eliminarán todas las filas de *tabla_o_vista.*

Un ejemplo de la sentencia DELETE es la siguiente instrucción:

```
DELETE Clientes WHERE IdCuenta = 1
```

- UPDATE

  Pérez López (2007) comenta que en el caso de la actualización de filas de datos ya existentes, la sentencia básica es UPDATE. Dicha instrucción puede cambiar los valores forma individual, grupo o todas las filas de una tabla o vista y al hacer referencia a éstas sólo se puede cambiar los datos de una a la vez.

  La sintaxis más sencilla es:

  UPDATE [usuario.] tabla [alias]

  SET {columna = expresión [, columna = expresión]…| (columna [, columna]…) = (subconsulta)}

  [WHERE condición];

  Las clausulas principales son:

  - SET: Contiene una lista separada por comas de columnas a actualizar y su nuevo valor, con el formato *columna = expresión*. El valor ingresado por la expresión incluye elementos como constantes, valores calculados por una expresión compleja o seleccionados por una columna de otra tabla o vista. La subconsulta debe seleccionar la misma cantidad de columnas (con tipos de datos compatibles) que haya entre los paréntesis al lado izquierdo de la cláusula SET. Las columnas que se igualen a expresiones deben anteponerse a las columnas entre paréntesis que se igualen a las subconsultas, dentro de una misma sentencia UPDATE.
  - WHERE: Especifica la condición de búsqueda para definir filas de tablas y vistas de origen que proporcionarán valores para las expresiones de la cláusula SET.

  Un ejemplo del uso de esta sentencia es la siguiente:

```
UPDATE Pedidos SET Estado = @estado WHERE IdPedido = @idPedido
```

  Cabe destacar que no sólo existe el de cambio de datos (utilizando la sentencia UPDATE) si no también la *actualización de todas las filas* con UPDATE y la utilización de *UPDATE con subconsulta* (Pérez López, 2007).

La *actualización de todas las filas* se realiza cuando se omite la sentencia WHERE en la instrucción de UPDATE, de forma que no se condicione la renovación de los datos.

Cuando se utiliza la sentencia *UPDATE con subconsulta*, ésta permite seleccionar las filas para actualizar en función de los datos contenidos en otras tablas.

En este tipo de actualización tiene la siguiente estructura:

UPDATE [usuario.] tabla [alias]
SET {columna = expresión [, columna = expresión]…| (columna [, columna]…) = (subconsulta)}
IN (SELECT – FROM - WHERE);

3.3.2 FORMAS NORMALES

Normalización

El proceso de normalización, según la propuesta original Codd (1972a), se somete un esquema de relación a una serie de pruebas para <<certificar>> si pertenece o no a una cierta forma normal. El proceso, que sigue un estilo descendente evaluando cada relación respecto a los criterios de normas formales y descomponiendo las relaciones a medida que sea necesario, puede considerarse por lo tanto un *diseño relacional por análisis*. En un principio, Codd propuso tres formas normales, a las cuales llamó primera, segunda y tercera forma normal. Posteriormente Boyce y Codd propusieron una definición más estricta de 3FN, a la que se conoce como forma normal de Boyce-Codd (FNBC). Todas estas formas normales se basan en las dependencias funcionales entre los atributos de una relación. Más adelante, se propusieron una cuarta (4FN) y quinta forma normal (5FN), basadas en los conceptos de dependencias multivaluadas y dependencias de reunión, respectivamente (Elmasri y Navathe, 2000).

La normalización de los datos puede considerarse como un proceso de análisis de los esquemas de relación dados basado en sus DF y claves primarias para alcanzar las propiedades deseables de:

1. Minimizar la redundancia.
2. Minimizar las anomalías de inserción, eliminación y actualización.

Los esquemas de relación insatisfactorios que no cumplan determinadas condiciones, las pruebas de formas normales, se descomponen en esquemas de relación más pequeños que satisfagan dichas pruebas y que de este modo posean las propiedades deseables. Por consiguiente, el procedimiento de normalización proporciona a los diseñadores los siguientes aspectos:

- Un marco formal para analizar los esquemas de relación basándose en sus claves y dependencias funcionales entre sus atributos.
- Una serie de pruebas de formas normales que pueden efectuarse sobre esquemas de relación individuales de modo que la base de datos relacional pueda normalizarse hasta el grado deseado.

La forma normal de una relación hace referencia a la condición de forma normal más alta y de este modo, indica el grado al que ha sido normalizada. Las formas normales cuando se consideran de manera aislada de otros factores, no garantizan un buen diseño de datos (Elmasri y Navathe, 2000).

Primera Forma Normal

La primera forma normal (1FN) se considera parte de la definición formal de relación en el modelo relacional básico, históricamente, se definió para prohibir los atributos multivaluados, los atributos compuestos y sus combinaciones. Establece que el dominio de un atributo debe incluir sólo valores atómicos (simples e indivisibles) y que el valor de cualquier atributo en una tupla debe ser un valor individual proveniente del dominio de ese atributo. Así pues, 1FN prohíbe las <<relaciones dentro de relaciones>> o las <<relaciones como atributos de tuplas>>. Los únicos valores de atributos que permite 1FN son valores atómicos (o indivisibles) (Elmasri y Navathe, 2000).

Segunda Forma Normal

La segunda forma normal (2FN) se basa en el concepto de dependencia funcional total. Ésta incluye la verificación de dependencias funcionales cuyos atributos del miembro

izquierdo son parte de la clave primaria. Si la clave primaria contiene un único atributo, no es en absoluto preciso aplicar la prueba. Un esquema de relación R está en 2FN si todo atributo no primo A en R depende funcionalmente de manera total de la clave primaria de R. Si un esquema de relación no está en 2NF, se le puede <<normalizar en 2FN>> dando lugar a varias relaciones 2FN en las que los atributos no primos estén asociados solo a la parte de la clave primaria de la que dependen funcionalmente de manera total (Elmasri y Navathe, 2000).

Tercera Forma Normal

La tercera forma normal (3FN) se basa en el concepto de dependencia transitiva. De acuerdo con la definición original de Codd, un esquema de relación R está en 3FN si está en 2FN y ningún atributo no primo de R depende transitivamente de la clave primaria (Elmasri y Navathe, 2000).

A continuación se presenta en la Tabla 3, el resumen de las formas normales basadas en claves primarias y su normalización correspondiente.

Tabla 3. Resumen de formas normales basadas en claves primarias y su normalización correspondiente.

Forma Normal	**Comprobación**	**Solución (normalización)**
Primera (1FN)	Una relación no debería tener ningún atributo no atómico ni relaciones anidadas.	Formar relaciones nuevas para cada atributo no atómico o relación anidada.
Segunda (2FN)	Para las relaciones en las que la clave primaria contiene múltiples atributos, ningún atributo no clave debería depender funcionalmente de una parte de la clave primaria.	Descomponer y crear una nueva relación para cada clave parcial con su atributo o atributos dependientes. Asegurarse de que se mantiene una relación con la clave primaria original y todos los atributos que

		dependan funcionalmente de manera total de ella.
Tercera (3FN)	Una relación no debería tener un atributo no clave determinado funcionalmente por otro atributo no clave (o por un conjunto de atributos no claves). Esto es, no debería existir una dependencia transitiva por parte de un atributo no clave de una clave primaria.	Descomponer y crear una relación que incluya el atributo o atributos no clave que determinen funcionalmente a otro u otros atributos no clave.

Fuente: Elmasri y Navathe (2000).

3.4 VISUAL STUDIO 2015

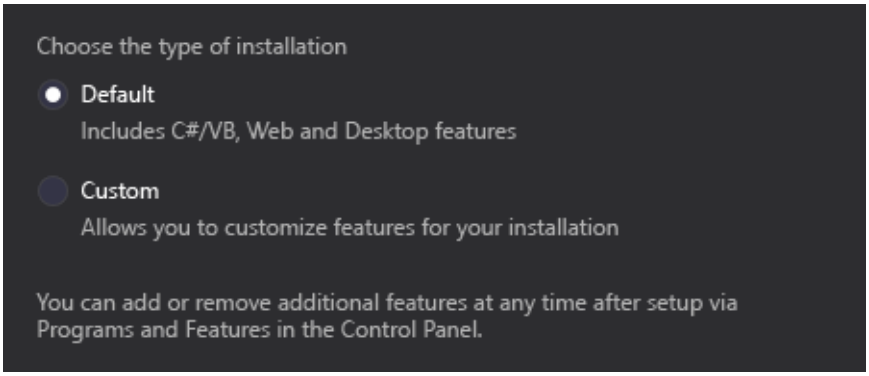
3.4.1 NUEVAS CARACTERÍSTICAS

Visual Studio Community 2015 integra un conjunto de herramientas para desarrolladores, servicios en la nube y extensiones que habilitan a uno o más desarrolladores para crear grandes aplicaciones y juegos desde la web, para la tienda de Windows, escritorio, Android y IOS (Novedades de Visual Studio 2015, s.f).

Entre sus nuevas características se encuentran las siguientes (Características de Sql Server Management Studio, s.f):

- Nueva experiencia de configuración:
  La experiencia de instalación de Visual Studio Community 2015 es mejor ya que sólo se instalan las partes que se necesitan. Esto hace la instalación más rápida para muchos escenarios comunes incluidos .NET o desarrollo Web. Si se necesitan otros elementos de desarrollo como el desarrollo móvil, C++ o F#, se elige la instalación personalizada con sus SDK necesarios. La interfaz de elección para la instalación se muestra en la Figura 9.

Figura 9. Elección en el tipo de instalación.

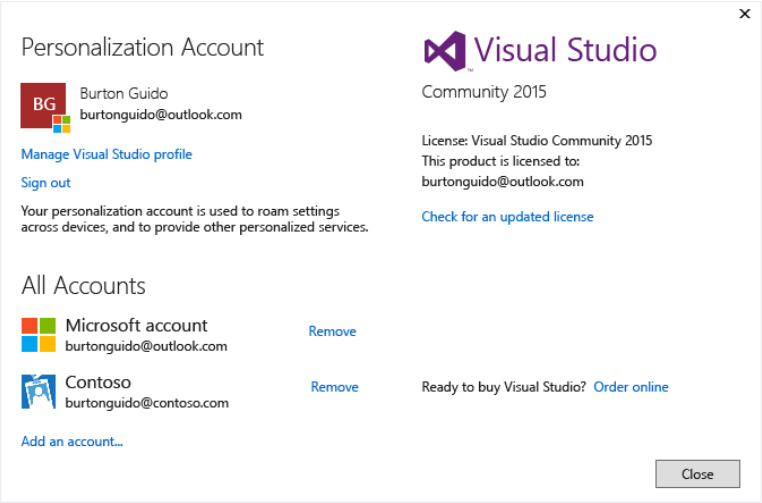


Fuente: Novedades de Visual Studio 2015 (s.f).

- Inicio de sesión a través de múltiples cuentas:

  Con Visual Studio Community, la nueva experiencia de inicio de sesión simplifica en gran medida el acceso de recursos en línea, incluso si se tienen diferentes cuentas en Visual Studio Community 2015 como se muestra en la Figura 10. Después de que se inicia con Visual Studio, se puede ingresar automáticamente a todas las instancias a Visual Studio 2015 en la computadora. La cuenta con la que se inicia se comparte entre entidades, de modo que siempre pueda acceder al grupo de servicios de Visual Studio desde el explorador, recursos y sitios web desde la suscripción a Microsoft Azure.

Figura 10. Inicio de sesión en Visual Studio Community 2015.



Fuente: Novedades de Visual Studio 2015 (s.f).

- Selección de plataforma

  Visual Studio Community 2015 soporta el desarrollo en dispositivos móviles multiplataforma, de manera que se puede desarrollar aplicaciones y juegos en IOS, Android y Windows compartiendo el código fuente dentro de IDE de Visual Studio. Y por supuesto, soporta las clásicas aplicaciones de escritorio mejor que nunca, con mejoras en lenguajes, librerías y herramientas. Algunas de estas son:

  - Aplicaciones multiplataforma en C# con Xamarin para Visual Studio.
  - Aplicaciones multiplataforma en HTML/JavaScript con Apache Cordova.
  - Juegos móviles multiplataforma en C# con Unity.
  - Aplicaciones y librerías multiplataforma para el nativo C++.
  - Aplicaciones Windows universales por cualquier dispositivo Windows 10.

  Véase la Figura 11.

Figura 11. Plataformas Windows.

Fuente: Novedades de Visual Studio 2015 (s.f).

  - Web (Con ASP.NET).
  - Clásico escritorio y tienda Windows (.NET Framework y C++).

- Barra de menú con prevista de dispositivos:

  En el menú de vista previa de dispositivos disponibles se podrá observar los distintos tamaños de pantalla en la que se mostrará la interfaz de usuario basada en XAML.

- Diagnóstico de gráficos en Visual Studio:

  Desde Visual Studio 2013, diagnóstico de gráficos de Visual Studio ha agregado nuevas características, incluyendo análisis de Frame, soporte de Windows Phone y herramientas para línea de comandos. También ha sido agregado el soporte para la depuración de aplicaciones DirectX12.

- Conexión a servicios:

  Visual Studio facilita la conexión de las aplicaciones con los servicios. El nuevo asistente para agregar un servicio de conexión a la configuración del proyecto, agrega el soporte de autenticación necesario y la descarga de paquetes NuGet para la codificación de apliaciones.

  El asistente para agregar Servicios de conexión también integra un manejador de cuentas para hacer fácil el trabajar con multiples cuentas de usuario y suscripciones. En Visual Studio 2015 soporta los servicios Azure Mobile Services, Azure Storage, Oficce 365 (correo, contactos, calendario, archivos, usuarios y grupos) y Saleforce. Los nuevos servicios pueden ser agregados de forma continua y descubrirlo en el asistente dando clic a la opción "Buscar nuevo enlace de servicios". Diseñar la propia interfaz de usuario.

- Soporte de depuración entre plataformas:

  Se puede utilizar Visual para crear y depurar aplicaciones móviles nativas para correr en Windows, IOS y dispositivos Android. De igual forma se puede utilizar el emulador de Visual Studio para Android, o conectar un dispositivo y depurar directamente en Visual Studio.

  - JavaScript/Cordova: Las herramientas de Visual Studio para Apache Cordova se utilizan para construir aplicaciones nativas para Windows, IOS y Android con JavaScript. La depuración de aplicaciones en la librería MSDN es detallada en el soporte de depuración de Visual Studio para Cordova.
  - C#/Xamarin: Xamarin es utilizado para construir aplicaciones nativas para Windows, IOS y Android en Visual Studio con C#. La guía desarrollo de Xamarín describe la experiencia de la depuración en IOS y demás dispositivos.

- C++/Android:

Utilizando las plantillas de desarrollo de Visual C++ multiplataforma junto con herramientas de terceros como el NDK de Android para crear aplicaciones nativas en Windows y Android.

- Depuración y diagnóstico:

  Lo siguiente son herramientas nuevas o mejoradas que realizan diferentes tipos de análisis y diagnóstico en el código:

  - PerfTips: Muestra en tiempo de ejecución los métodos durante la depuración, lo que permite detectar rápidamente cuellos de botella sin tener que invocar la instrumentación del código fuente (ejecutable en la forma binaria).
  - Lista de errores: La lista de errores ahora admite el filtrado en cualquier columna. Ahora muestra una vista de errores, advertencias y análisis de código en la solución C# o Visual Basic mientras escribe, incluso cuando genera miles de advertencias. La nueva lista de errores es compatible con lo ya existente de otras versiones.
  - Herramienta de uso de GPU (Unidad de procesamiento gráfico).
  - La herramienta de uso de GPU ayuda a coleccionar y analizar el uso de datos GPU en DirectX de aplicaciones y juegos, para así averiguar si los cuellos de botella se originan en la CPU o en la GPU.

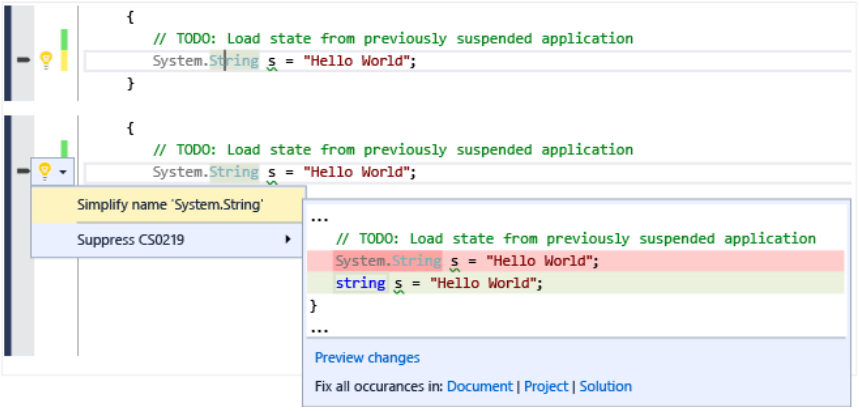
- Análisis de código en tiempo real:

  El nuevo compilador Roslyn para C# y Visual Basic no sólo provee tiempos de compilación más rápida, si no que permite escenarios completamente nuevos como el análisis de código en directo, que proporciona comentarios y sugerencias personalizadas en el editor de código mientras escribe.

  En Visual Studio 2015, se muestran bombillas en el margen izquierdo (cuando se usa el teclado) o en el puntero de mouse. La bombilla indica en tiempo real que el compilador ha detectado un problema en el código así como una sugerencia de cómo resolver ese problema. Cuando se observa la bombilla, se puede hacer clic sobre ella y accionar las sugerencias.

Se puede observar un ejemplo en la Figura 12.

Figura 12. Presentación de bombillas de problemas en código.



Fuente: Novedades de Visual Studio 2015 (s.f).

- Disfrutar de las mejoras adicionales del IDE:
  - Ajustes sincronizados

    Visual Studio 2013 introduce la sincronización de algunas de las configuraciones como el editor de texto, enlaces de teclas, temas, fuentes colores y alias de entorno.
  - Actualización automática de extensiones:

    Al instalar alguna de las extensiones de Visual Studio ahora se podrán actualizar automáticamente a otra versión disponible en la galería de Visual Studio (Novedades de Visual Studio 2015, s.f).
  - Títulos en el menú:

    Las opciones principales del menú son configurables, aunque ya vienen algunas por defecto se pueden configurar en la opción *Herramientas* > *Opciones* > *Propiedades Generales.*
  - Imágenes de alta resolución y soporte táctil:

    El IDE de Visual Studio ahora tiene alta resolución de imágenes en pantallas más densas (en áreas como menús, menús de contexto, barras de comandos en ventanas de herramientas y en algunos proyectos el explorador de soluciones). Y en una pantalla táctil en la ventana del editor de código de

Visual Studio, ahora se puede tocar, mantener presionado, acercar, desplazarse, seleccionar texto o invocar menús contextuales.

- Diseños contextuales:

  Ya se pueden crear diseños de ventanas personalizadas, por ejemplo se pueden definir un diseño para usarlo en una máquina de escritorio y otro diferente para usar en una laptop o dispositivo con pantalla pequeña. O quizás se prefiera un diseño para una interfaz de usuario para el proyecto y otra para un proyecto de base de datos.

- Centro de notificación:

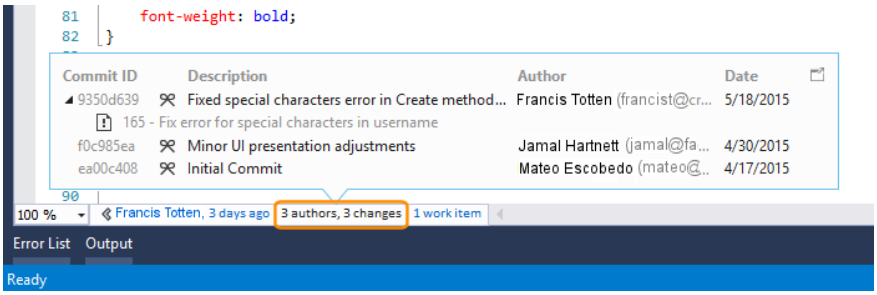
  La interfaz del centro de notificaciones ha sido coordinada para hacer más fácil y rápido la exploración. Se han agregado tipos adicionales de notificaciones, problemas de rendimiento, problemas de representación y bloqueos, con la novedad de decidir si se quiere que Visual Studio muestre esas notificaciones.

- CodeLens:

  Descubre qué pasa con el código (sólo para ediciones Enterprise y Professional).

  Ahora ya se puede mantener enfocado en el trabajo mientras se busca información acerca del código sin salir del editor. Se pueden revisar cambios y otros historiales de elementos, errores y revisiones de código almacenado en el Equipo de Servicios de Visual Studio (VSTS). Se pueden observar el historial de un archivo de código en el editor de Visual Studio, como se muestra en la Figura 13.

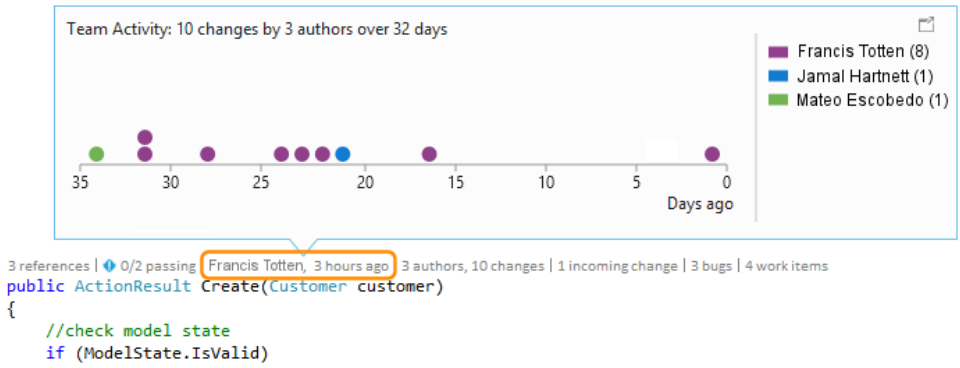
Figura 13. Historial de archivo de código.



Fuente: Novedades de Visual Studio 2015 (s.f).

De igual manera se puede visualizar gráficos sobre quién ha modificado el código. Esto puede ayudar a encontrar patrones en los cambios del equipo y evaluar el impacto. Véase la Figura 14.

Figura 14. Gráfico de modificaciones registradas en archivo de código en Visual Studio.



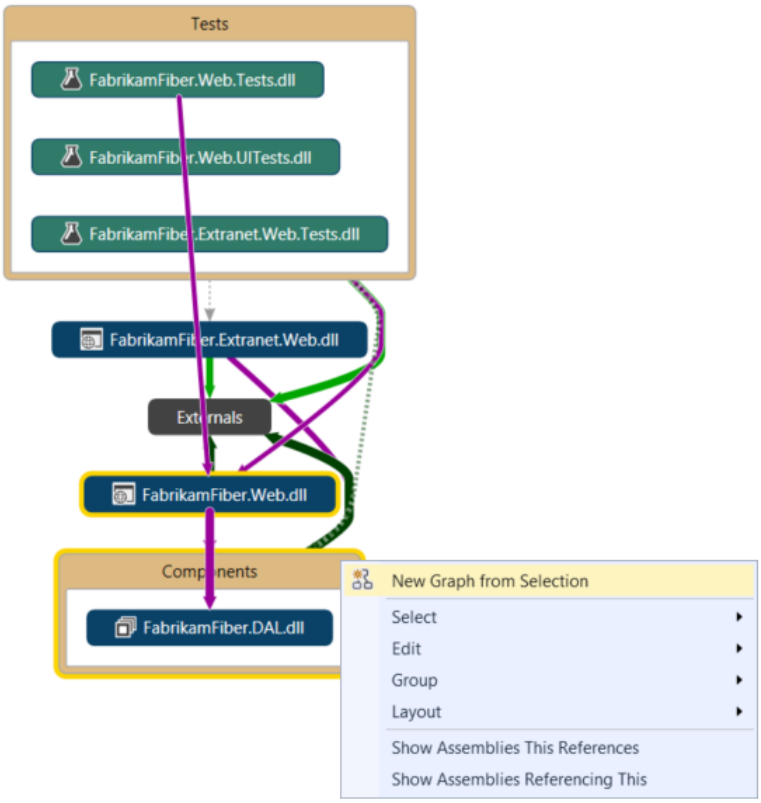
Fuente: Novedades de Visual Studio 2015 (s.f).

- Herramientas de modelado y diseño (solo para ediciones Enterprise)
  - Mapas de código y gráficos de dependencia:

    En Visual Studio edición Enterpise, cuando se quiere entender dependencias específicas en el código, se puede visualizar creando mapas de código, mediante éstos se puede navegar éstas relaciones que aparecen junto al código. El mapa de código puede ayudar a seguir la pista del código mientras se trabaja en la depuración del código, así se leerá menos código mientras se aprende más sobre el diseño del mismo.

    En ésta versión, se pueden crear menús de atajo para elementos de código y enlaces mucho más fáciles de usar por grupos de comandos en secciones relacionadas con la selección, edición, así como gestión y cambio del diseño de contenidos de grupos. Un ejemplo de estos mapas podría ser el que se muestra en la Figura 15.

Figura 15. Mapa de código y gráficos de dependencia.



Fuente: Novedades de Visual Studio 2015 (s.f).

- Mejora de diagramas.
- Estilo y filtrado a proyectos de prueba.
- Simplificado de enlaces de dependencia externa:

  Los vínculos de dependencia ya no representan la herencia de *System.Object*, *System.ValueType*, *System.Delegate* y *System.Enum*, lo que facilita ver las dependencias externas en el mapa de código.
- Ejercicios de enlaces de dependencia teniendo en cuenta los filtros:

  Se obtendrá un diagrama claro y útil para comprender los vínculos de dependencia, ya que se encuentra más ordenado tomando en cuenta las opciones de filtrado de enlaces seleccionados.
- Elementos del código se agregan al mapa de código con su contexto:

  Porque los diagramas ahora aparecen con su contexto, obteniendo mayor utilidad cuando se arrastran o sueltan elementos de código

desde el explorador de soluciones, vistas de clases, navegador de objetos o cuando se seleccionan elementos del explorador de soluciones y se eligen mostrar en el mapa de código.

- Obtener mapas de código reactivos más rápidamente:
  Al arrastrar y soltar operaciones, produce resultados inmediatos y los enlaces entre los nodos son creados mucho más rápido, sin afectar subsecuentemente operaciones iniciadas por el usuario tal como la expansión de un nodo o la solicitud a más nodos.
- Omitir reconstruir soluciones:
  Provee mejor funcionamiento cuando se crean y agregan diagramas.
- Filtra elementos de código de nodos y grupos.
- Filtra las relaciones para realizar diagramas más fáciles de leer.
- Crea diagramas desde la vista de clases y el navegador de objetos.
- Diagramas de capas:
  Actualiza diagramas usando la vista de clases y el navegador de objetos. Para conocer los requerimientos de diseño de software, se puede usar diagramas de capas para describir las dependencias deseadas para el software. Manteniendo el código consistente con el diseño, detectando un código que no cumpla con las restricciones, validando de esta forma el código futuro con la base línea.
- Diagramas UML.
- Explorador de arquitectura.

- Herramientas de extensibilidad de Visual Studio:
  Nunca había sido tan fácil instalar herramientas de extensibilidad en Visual Studio, y ahora son incluidas como un componente opcional durante la instalación.

3.4.2 DEFINICIÓN DE CONEXIONES A BASES DE DATOS

Se pueden registrar muchas conexiones distintas a bases de datos inclusive a distintos proveedores de base de datos para esto nuestra cadena de conexión se puede definir dentro del archivo app.config (Torres Remon, 2012).

Antes de implementar la cadena de conexión hay que tener en cuenta que el archivo app.config ya viene asignado al proyecto y Visual Studio 2015 lo hace accesible al usuario, aspecto que no pasaba en las versiones anteriores a Visual Studio 2012 (Torres Remon, 2012).

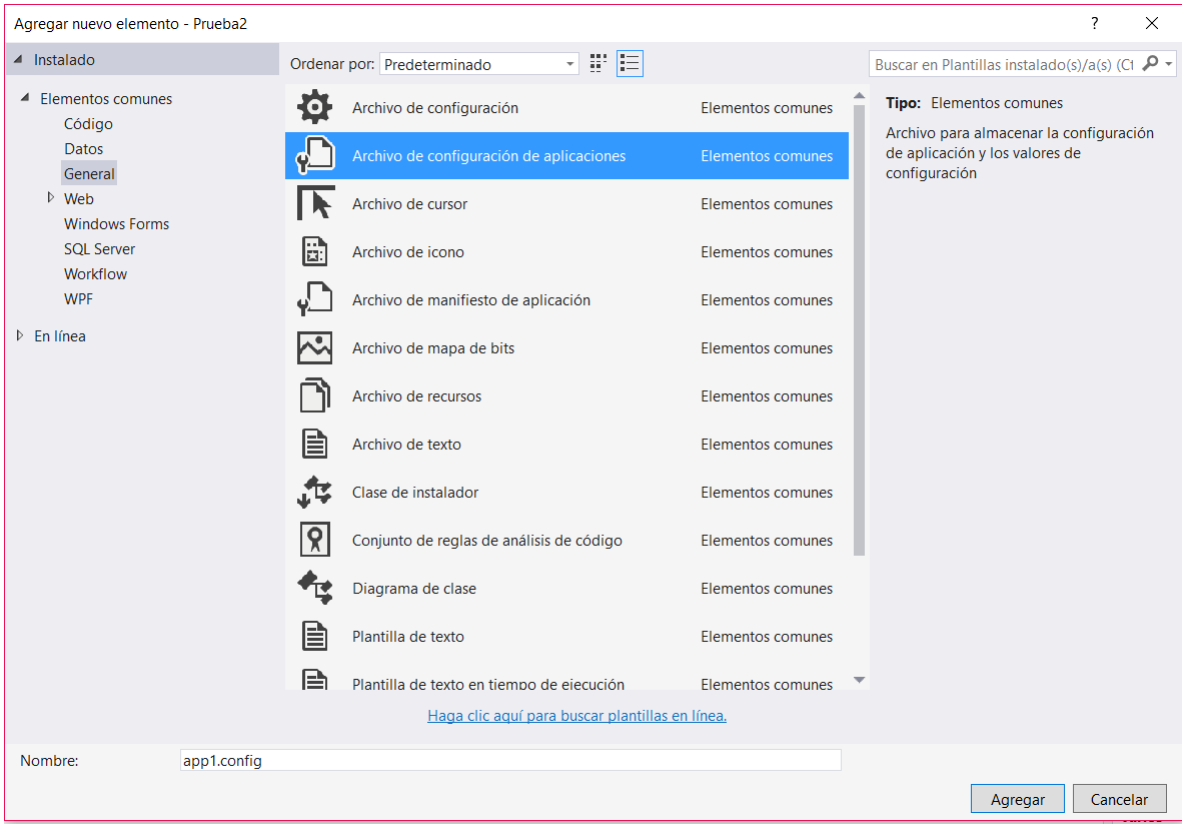
El cual presenta el siguiente código base:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2" />
    </startup>
</configuration>
```

En caso de que no sea visible se puede agregar un app.config de la siguiente forma:

- Clic derecho sobre el proyecto
- Seleccionar Agregar > Nuevo elemento > Plantilla General > Archivo de Configuración de aplicaciones como se muestra en la Figura 16.

Figura 16. Agregar nuevo archivo de conexión App.config.



El nombre predeterminado es app1.config cambiarlo por app.config si .NET pregunta si se desea sobrescribir, seleccionar si ya que este es el archivo que permitirá configurar las cadenas de conexión e inicialmente está vacío de configuración así es que no se perderá nada (Torres Remon, 2012).

3.4.3 COMPONENTES

System.Configuration

Es un espacio de nombres que contiene los tipos que proporcionan el modelo de programación para controlar los datos de la configuración. Para el desarrollo de aplicaciones que hagan uso de accesos a bases de datos en SQL Server, se necesita de la referencia de System.Configuration al .NET Framework, para con esto hacer uso del archivo *app.config* y sus datos de conexión. En las Figuras 17, 18 y 19, se demostrará la secuencia para añadir dicha referencia a un proyecto (Espacio de nombres System.Configuration, s.f).

Figura 17. Agregar referencia de System.Configuration.

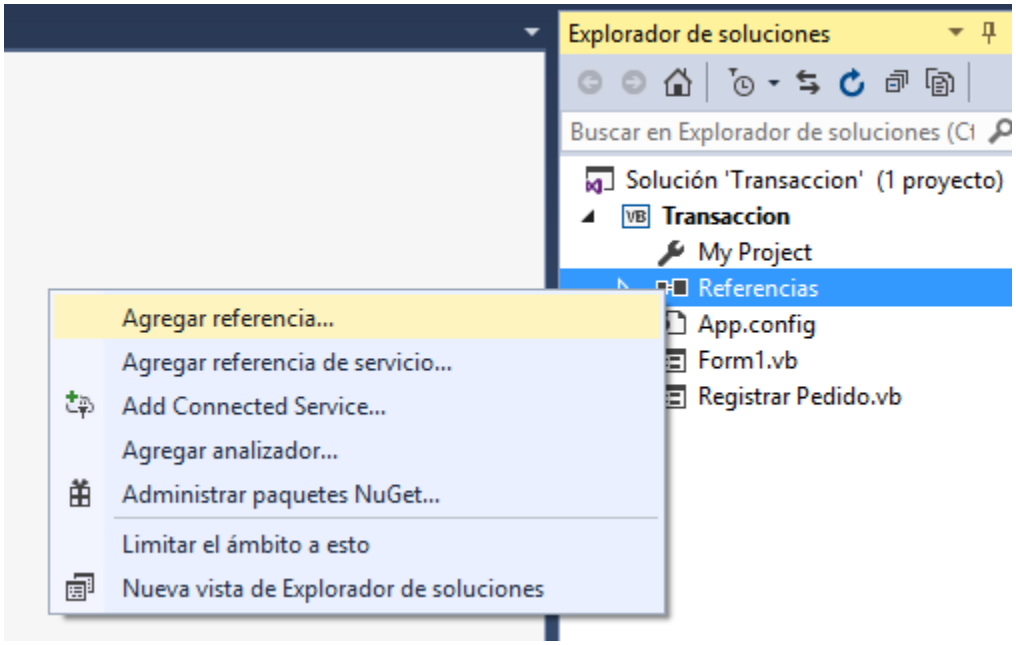


Figura 18. Selección de referencia de System.Configuration.

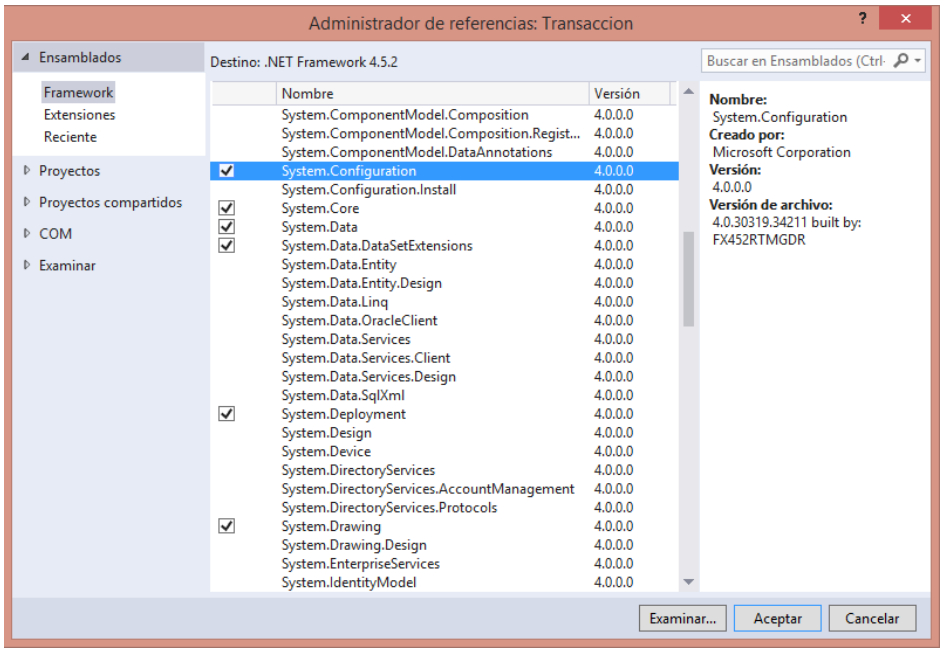
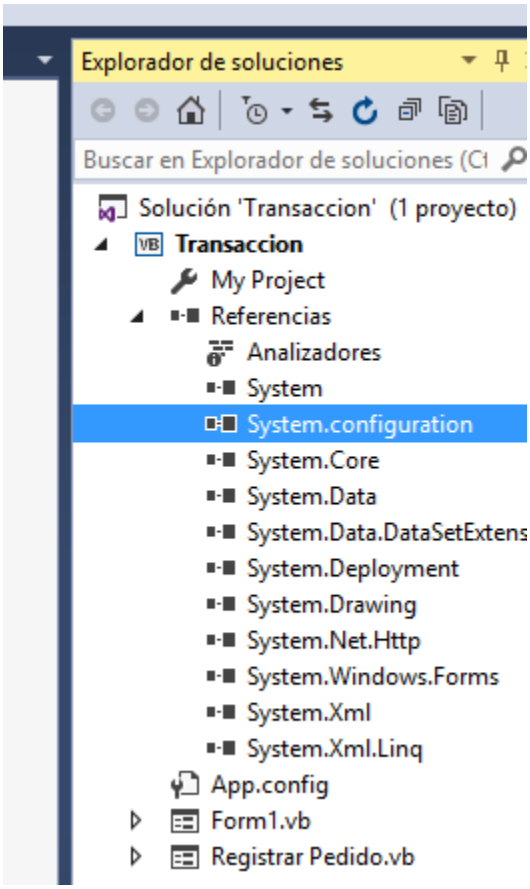


Figura 19. Verificación de referencia de System.Configuration.



System.Data.SqlClient

Es un espacio de nombres que provee de datos .NET Framework para Sql Server. Dentro de su colección de clases es utilizada para tener acceso a la base de datos en el servidor SQL según su espacio administrado. El objeto DataSet residente en memoria puede ser llenado al utilizar SqlDataAdapter, para de esta forma realizar consultas y actualizaciones a la base de datos (Torres Remon, 2012).

Las clases que componen este espacio de nombres se muestran en la Tabla 4.

Tabla 4. Descripción de clases de System.Data.SqlClient.

Nombre	Descripción
SqlClientFactory	Representa un conjunto de métodos para la creación de instancias de la implementación de las clases manejadas como origen de datos de System.Data.SqlClient como proveedor.
SqlCommand	Se representa como un procedimiento almacenado que se ejecuta en una base de datos. Dicha clase no se puede heredar.
SqlCommandBuilder	Esta clase genera automáticamente comandos de tabla única utilizados para conciliar los cambios realizados en el objeto DataSet con la base de datos asociada en el servidor SQL. No se puede heredar.
SqlConnection	Representa una conexión abierta de la base de datos en Sql Server. Esta clase no se puede heredar.
SqlConnectionString Builder	Administra y crea cadenas de conexión de una forma más fácil utilizadas por la clase SqlConnection.
SqlDataAdapter	Representa una conexión a la base de datos así como un conjunto de comandos de datos que se utilizan para llenar un DataSet y actualizar la base de datos en el servidor Sql. Esta clase no se puede heredar.
SqlDataReader	Posibilita una forma de leer una secuencia de filas hacia adelante en una base de datos de Sql Server. Esta clase no se puede heredar.
SqlError	Se encarga de volver a compilar información relevante para una advertencia o error devuelto por Sql Server.
SqlException	Presenta una excepción cuando Sql Server devuelve un error o advertencia. No se puede heredar esta clase.
SqlParameter	Representa un parámetro de un objeto SqlCommandn, y de forma opcional, a su asignación de columnas de un DataSet. Esta clase no puede heredarse.

SqlProviderServices	Implementa de DbProviderServices para el proveedor SqlClient en Sql Server.
SqlTransaction	Representa una transacción de Transact-SQL realizada en una base de datos en Sql Server. Dicha clase no puede heredarse.

Fuente: Torres Remon (2012).

SqlConnection

En el momento de desarrollo de una aplicación cliente-servidor siempre se necesitara tener abierta una conexión hacia una base de datos, por lo que la clase SqlConnection se encarga de realizar esta tarea de forma efectiva sin mediar el tipo de proveedor. Cuando se crea un objeto de ésta clase, este representa una sesión única de un origen de datos de SqlServer y cuando el caso es de un sistema de bases de datos de cliente y servidor, equivale a una conexión de red al servidor (Torres Remon, 2012).

Los principales métodos constructores se muestran en la Tabla 5.

Tabla 5. Descripción de métodos constructores de la clase SQLConnection.

Nombre	Descripción
SqlConnection	Inicializa una nueva instancia de la clase SqlConnection
SqlConnection(String)	Inicializa una nueva instancia de la clase SqlConnection mediante una cadena de conexión dada.
SqlConnection(String, SqlCredential)	Inicializa una nueva instancia de la clase SqlConnection mediante una cadena de conexión dada que no hace uso de Integrated Security = true asi como un objeto de SqlCredential que contiene el Id de usuario y contraseña.

Fuente: Torres Remon (2012).

Las propiedades se muestran en la Tabla 6.

Tabla 6. Descripción de propiedades de la clase SQLConnection.

Nombre	Descripción
ConnectionString	Obtiene o establece la cadena que se utiliza para abrir una base de datos de Sql Server. Ésta invalida a DbConnectin.ConnectionString
Container	Obtiene IContainer que contiene Component y se hereda de este mismo.
Credential	Obtiene o establece el objeto de SqlCredential para esta conexión.
Database	Obtiene el nombre de la base de datos actual o de la que se va a ocupar una vez que se ha abierto la conexión. Invalida a DbConnection.Database.
DataSource	Obtiene el nombre de la instancia de Sql Server a la cual se puede conectar. (Invalida a DbConnection.DataSource).
DbProviderFactory	Obtiene el objeto DbProviderFactory para la instancia de DbConnection.
ServerVersion	Obtiene la cadena que contiene la versión de la instancia de Sql Server con la que el cliente está conectado. Invalida a DbConnection.ServerVersion.

Fuente: Torres Remon (2012).

Lo métodos se muestran en la Tabla 7.

Tabla 7. Descripción de métodos de la clase SQLConnection.

Nombre	Descripción
BeginTransaction	Inicia una transacción de base de datos
BeginTransaction(String)	Inicia una transacción de base de datos con un nombre de transacción especificado.
Close	Cierra la conexión con la base de datos. Es el método preferido para cerrar cualquier conexión abierta.
CreateCommand	Crea y devuelve un objeto SqlCommand asociado a la conexión SqlConnection.
CreateDbCommand	Crea y devuelve un objeto DbCommand asociado a la conexión actual.
Dispose	Libera los recursos utilizados por Component el cual es heredado de éste mismo.
Dispose(Boolean)	Libera los recursos no administrados que utiliza Component y libera los recursos administrados de forma opcional. (Hereda de Component).
Open	Abre una conexión de base de datos con los valores de propiedad que especifica ConnectionString. Invalida a DbConnection.Open.
ToString	Devuelve un objeto String el cual contiene el nombre del objeto Component. Este método no debe reemplazarse. Hereda de Component.

Fuente: Torres Remon (2012).

Para implementar una cadena a una base de datos del motor de base de datos SQL Server es necesario tomar en cuenta lo siguiente.

1.- Importar las librerías necesarias para los objetos de conexión.

```
Imports System.Data.SqlClient
Imports System.Configuration
```

2.- Declarar el objeto de la clase SqlConnection en conjunto con los elementos de la conexión

```
Dim cn As New SqlConnection("Server=(local); Database=bd; Integrated Security=SSPI")
```

3.- Declaración del principio del bloque que hará uso de la conexión a la base de datos asignándole a cn la propiedad ConnecionString

```
    Using x As New
SqlConnection(ConfigurationManager.ConnectionStrings("cn").ConnectionString)
```

SqlDataAdapter

SqlDataAdapter se utiliza como un puente entre DataSet y el servidor SQL para recuperar y guardar datos, todo mediante la asignación de Fill, el cual se encarga de cambiar los datos en el DataSet para que coincidan con los del origen de datos y Update haciéndolo de manera contraria, es decir, que cambie los datos en el origen de datos para que coincidan con los del DataSet. Todo esto mediante sentencias de Transact-SQL (Torres Remon, 2012).

Cabe destacar que la actualización se realiza fila a fila, para lo cual para cada una de ellas ya sea de inserción, actualización o eliminación, el método Update determina el tipo de cambio y con ello la ejecución de la plantilla de comando (Insert, Update o Delete) para difundir la modificación al origen de datos (Torres Remon, 2012).

Cuando SqlDataAdapter rellene un objeto DataSet, creará las tablas y columnas necesarias para los datos devueltos, si es que aún no existen (Torres Remon, 2012).

Al crearse una instancia de SqlDataAdapter, las propiedades de lectura y escritura se establecen en sus valores iniciales. Existen las propiedades InsertCommand, DeleteCommand y UpdateCommand son plantillas genéricas que se llenan automáticamente

con valores individuales de las filas modificadas mediante un mecanismo de parámetros (Torres Remon, 2012).

A continuación se muestran los métodos constructores, propiedades, así como métodos comunes en la Tabla 8, 9 y 10, respectivamente.

Tabla 8. Descripción de métodos constructores de la clase SQLDataAdapter.

Nombre	Descripción
SqlDataAdapter	Inicializa una nueva instancia de la clase SqlDataAdapter.
SqlDataAdapter(SqlCommand)	Inicializa una nueva instancia de la clase SqlDataAdapter con la especificación adicional de la propiedad SelectCommand.
SqlDataAdapter(String, SqlConnection)	Inicializa una nueva instancia de la clase SqlDataAdapter con la especificación adicional de la propiedad SelectCommand y un objeto SqlConnection.
SqlDataAdapter(String, String)	Inicializa una nueva instancia de la clase SqlDataAdapter con la especificación adicional de la propiedad SelectCommand y una cadena de conexión.

Fuente: Torres Remon (2012).

Tabla 9. Descripción de propiedades de la clase SqlDataAdapter.

Nombre	Descripción
DeleteCommand	Obtiene y establece un procedimiento almacenado o una instrucción de Transact-SQL para eliminar registros de un conjunto de datos.
DesigMode	Obtiene un valor que indica si Component está actualmente en modo de diseño. Hereda de Component.
InsertCommand	Obtiene o establece un procedimiento almacenado o una instrucción de Transact-SQL para insertar nuevos registros en el origen de los datos.
SelectCommand	Obtiene o establece un procedimiento almacenado o una instrucción de Transact-SQL para seleccionar registros en el origen de los datos.
UpdateCommand	Obtiene o establece un procedimiento almacenado o una instrucción de Transact-SQL para actualizar los registros en el origen de los datos.

Fuente: Torres Remon (2012).

Tabla 10. Descripción de métodos de la clase SqlDataAdapter.

Nombre	Descripción
Dispose	Libera todos los recursos utilizados por Component. Hereda de Component.
Fill(DataSet)	Agrega o actualiza filas en DataSet y se hereda de DbDataAdapter
Fill(DataTable)	Agrega o actualiza filas en un intervalo especificado en el DataSet que coincidan con las de origen de datos utilizando el DataTable nombre. (Heredado de DbDataAdapter).
Fill(DataSet, String)	Agrega o actualiza filas en el DataSet que coincidan con las de origen de datos utilizando el DataSet y DataTable nombres. (Heredado de DbDataAdapter).
Fill(DataTable, IDataReader)	Agrega filas al objeto DataTable o las actualiza para hacerlas coincidir con las que están en el origen de datos, utilizando el nombre de DataTable así como la interfaz IDataReader especificada.
Fill(DataTable, IDbCommand, CommandBehavior)	Añade filas a un objeto DataTable o las actualiza para hacerlas coincidir con las que están en el origen de datos, utilizando los objetos DataTable, IDbCommand y CommandBehavior especificados. Se hereda de DbDataAdapter.
Fill(DataTable(), IDataReader, Int32, Int32)	Agrega filas en un intervalo especificado de la colección de objetos DataTable o las actualiza para hacerlas coincidir con las filas del origen de datos.
Fill(DataSet, String, IDataReader,Int32,Int32)	Agrega filas en un intervalo especificado de DataSet o las actualiza para hacerlas coincidir con

	las filas del origen de datos utilizando los nombres de DataSet y DataTable. Se hereda de DataAdapter.
Fill(DataTable(), Int32, Int32, IDbCommand, CommandBehavior)	Agrega filas en un intervalo especificado de DataSet o las actualiza para hacerlas coincidir con las filas del origen de datos utilizando los nombres de DataSet y DataTable. Se hereda de DbDataAdapter.
Finalize	Libera recursos no administrados y realiza otras operaciones de limpieza antes de que se reclame el objeto Component durante la recolección de los elementos que no se utilizaron. Este hereda de Component.
ToString	Devuelve un String que contiene el nombre de la Component, si existe. Este método no se debe invalidar. (Heredado de Component).
Update(DataRow())	Actualiza los valores de la base de datos mediante la ejecución de las instrucciones INSERT, UPDATE o DELETE respectivas para cada inserta, actualiza o elimina la fila de la matriz especificada en el DataSet. (Heredado de DbDataAdapter).
Update(DataSet)	Actualiza los valores de la base de datos mediante la ejecución de las instrucciones INSERT, UPDATE o DELETE respectivas para cada inserta, actualiza o elimina la fila de la manera especificada DataSet. (Heredado de DbDataAdapter).
Update(DataTable)	Actualiza los valores de la base de datos mediante la ejecución de las instrucciones INSERT, UPDATE o DELETE respectivas para cada

	inserta, actualiza o elimina la fila de la manera especificada DataTable. (Heredado de DbDataAdapter).
Update(DataRow(), DataTableMapping)	Realiza un llamado a las instrucciones INSERT, UPDATE o DELETE respectivas para cada fila insertada, actualizada o eliminada en la matriz de objetos DataRow especificada. Este hereda de DbDataAdapter.
Update(DataSet, String)	Actualiza los valores de la base de datos mediante la ejecución de las instrucciones INSERT, UPDATE o DELETE respectivas para cada insertarlos, actualizarlos o fila eliminada de la DataSet con el parámetro DataTable nombre. (Heredado de DbDataAdapter).

Fuente: Torres Remon (2012).

La sintaxis de un SqlDataAdapter es la siguiente:

Dim nombreDataAdapter as New SqlDataAdapter ("Intruccion", CadenaConexion)

En donde el *nombre del adaptador* el cual lleve como inicial de la palabra *da*, la *instrucción* indica la consulta o procedimiento almacenado a ejecutarse en el servidor y la cadena de conexión es aquella en la que se especifique el nombre de la cadena preparada para la conexión (Torres Remon, 2012).

SqlCommand

Es una clase que representa un procedimiento almacenado o una instrucción de Transact-SQL que se ejecuta en una base de datos de SQL Server, la cual no puede heredarse (Microsoft SqlCommand, s.f).

Al crearse una instancia de esta clase, las propiedades de lectura y escritura se establecen en sus valores iniciales (Torres Remon, 2012).

Sus principales métodos constructores se muestran en la Tabla 11.

Tabla 11. Descripción de métodos constructores de la clase SqlCommand.

Nombre	Descripción
SqlCommand	Inicializa una nueva instancia de la clase SqlCommand.
SqlCommand(String)	Inicializa una nueva instancia de la clase SqlCommand con el texto de la consulta.
SqlCommand(String, SqlConnection)	Inicializa una nueva instancia de la clase SqlCommand con el texto de la consulta y una conexión SqlConnection.
SqlCommand(String, SqlConnection, SqlTransaction)	Inicializa una nueva instancia de la clase SqlCommand con el texto de la consulta, un objeto SqlConnection y el objeto SqlTransaction.

Fuente: Torres Remon (2012).

Sus propiedades se muestran en la Tabla 12.

Tabla 12. Descripción de propiedades de la clase SqlCommand.

Nombre	Descripción
CommandText	Obtiene o establece la instrucción de Transact-SQL, el nombre de tabla o el procedimiento almacenado que se ejecutan en el origen de datos. (Invalida DbCommand.CommandText).
CommandTimeout	Obtiene o establece el tiempo de espera antes de terminar el intento de ejecutar un comando y generar un error. (Invalida DbCommand.CommandTimeout).
CommandType	Obtiene o establece un valor que indica cómo se interpreta la propiedad CommandText. (Invalida DbCommand.CommandType).

Connection	Obtiene o establece la interfaz SqlConnection que usa esta instancia de SqlCommand.
Parameters	Obtiene la estructura SqlParameterCollection.
Transaction	Obtiene o establece la transacción SqlTransaction en la que se ejecuta SqlCommand.
UpdateRowSource	Obtiene o establece la manera en que se aplican los resultados del comando a DataRow cuando lo utiliza el método Update de DbDataAdapter. (Invalida DbCommand.UpdatedRowSource).

Fuente: Torres Remon (2012).

Sus métodos se describen en la Tabla 13.

Tabla 13. Descripción de métodos de la clase SqlCommand.

Nombre	Descripción
Cancel	Intenta cancelar la ejecución de un SqlCommand. (Invalida DbCommand.Cancel()).
CreateParameter	Crea una nueva instancia de un objeto SqlParameter.
Dispose	Libera todos los recursos que usa Component. (Heredado de Component).
Equals(Object)	Determina si el objeto especificado es igual al objeto actual. (Heredado de Object).
ExecuteDbDataReader	Ejecuta el texto de comando en la conexión. Se hereda de DbCommand.

ExecuteNonQuery	Ejecuta una instrucción de Transact-SQL en la conexión y devuelve el número de filas afectadas. (Invalida DbCommand.ExecuteNonQuery()).
ExecuteReader	Envía la propiedad CommandText a Connection y crea un objeto SqlDataReader.
ExecuteScalar	Ejecuta la consulta y devuelve la primera columna de la primera fila del conjunto de resultados devuelto por la consulta. Las demás columnas o filas no se tienen en cuenta. (Invalida DbCommand.ExecuteScalar()).
ExecuteXmlReader	Envía CommandText a Connection y crea un objeto XmlReader.
Finalize	Libera los recursos no administrados y realiza otras operaciones de limpieza antes que se reclame el objeto Component durante la recolección de elemento que no se utilizaron. Este se hereda de Component.
GetType	Obtiene el Type de la instancia actual. (Heredado de Object).
ToString	Devuelve un String que contiene el nombre de la Component, si existe. Este método no se debe invalidar. (Heredado de Component)

Fuente: Torres Remon (2012).

DataSet

Es una caché de memoria interna que almacena los datos recuperados de un origen de datos el cual representa un componente fundamental de la arquitectura ADO.NET. Por otra parte es una representación residente de memoria de datos que proporciona un modelo de programación relacional coherente independientemente del origen de datos, ya que se pueden utilizar con muchos y distintos orígenes de datos, datos XML o para la administración de datos locales de la aplicación (Torres Remon, 2012).

DataSet está compuesto por una colección de objetos DataTable que se pueden relacionar entre ellos mediante objetos DataRelation (Torres Remon, 2012).

El objeto DataSet es esencial para la compatibilidad con escenarios de datos distribuidos que se encuentran desconectados de ADO.NET. Este objeto representa un conjunto completo de datos que incluye tablas relacionadas, restricciones así como relaciones entre las tablas (Torres Remon, 2012).

A continuación de muestran sus métodos constructores, propiedades y principales métodos en la Tabla 14, 15 y 16, respectivamente.

Tabla 14. Descripción de métodos constructores de la clase DataSet.

Nombre	Descripción
DataSet	Inicializa una nueva instancia de la clase DataSet.
DataSet(String)	Inicializa una nueva instancia de un DataSet clase con el nombre especificado.
DataSet(SerializationInfo, StreamingContext)	Infraestructura producto que no está diseñada para usarse directamente desde el código. Inicializa una nueva instancia de un DataSet clase que tiene la información de serialización especificado y el contexto.
DataSet(SerializationInfo, StreamingContext, Boolean)	Infraestructura producto que no está diseñada para usarse directamente desde el código. Inicializa una nueva instancia de la clase DataSet.

Fuente: Torres Remon (2012).

Tabla 15. Descripción de propiedades de la clase DataSet.

Nombre	Descripción
CaseSensitive	Obtiene o establece un valor que indica si comparaciones dentro de cadenas DataTable objetos distinguen mayúsculas de minúsculas.
Container	Obtiene el contenedor del componente. (Heredado de MarshalByValueComponent).
DataSetName	Obtiene o establece el nombre del actual DataSet.
HasErrors	Obtiene un valor que indica si hay errores en cualquiera de los DataTable objetos dentro de este DataSet.
Namespace	Obtiene o establece el espacio de nombres de la DataSet.
Relations	Obtiene la colección de relaciones que vincula las tablas y permitir la navegación de primario tablas a las tablas secundarias.
Tables	Obtiene la colección de tablas contenidas en el DataSet.

Fuente: Torres Remon (2012).

Tabla 16. Descripción métodos de la clase DataSet.

Nombre	Descripción
Clear	Borra la DataSet de los datos mediante la eliminación de todas las filas de todas las tablas.
Copy	Copia la estructura y los datos para este DataSet.
CreateDataReader	Devuelve un DataTableReader con un conjunto de resultados por DataTable, en la misma secuencia que las tablas aparecen en la Tables colección.
CreateDataReader(DataTable())	Devuelve un DataTableReader con un conjunto de resultados por DataTable.
Dispose	Libera todos los recursos que usa MarshalByValueComponent.(Heredado de MarshalByValueComponent).
GetService	Obtiene el implementador de la IServiceProvider. (Heredado de MarshalByValueComponent).

GetType	Obtiene el Type de la instancia actual. (Heredado de Object).
GetXml	Devuelve la representación XML de los datos almacenados en el DataSet.
Reset	Borra todas las tablas y quita todas las relaciones, restricciones externas y tablas de la DataSet. Las subclases deben reemplazar Reset para restaurar un DataSet a su estado original.
ToString	Devuelve un String que contiene el nombre de la Component, si existe. Este método no se debe invalidar. (Heredado de MarshalByValueComponent).

Fuente: Torres Remon (2012).

Transaction

En Visual Studio hay espacio de nombres con sus respectivas clases que permiten escribir aplicaciones transaccionales en conjunto con administradores de recursos propios llamado *System.Transactions,* en la que se permita que tanto los recursos volátiles como el alta de un único recurso duradero puedan confirmar o deshacer fácilmente. Una de esas clases es *Transaction* la cual representa una transacción (Torres Remon, 2012).

SqlTransaction

La clase SqlTransaction es una clase no heredable la cual representa una transacción en Transact-SQL realizándose en una base de datos dada de alta en SQL Server (Torres Remon, 2012).

Al realizarse la aplicación se crea un objeto SqlTransaction al ejecutarse la instrucción BeginTransaction mediante el objeto de la conexión (SqlConnection). Una vez creado el objeto de la transacción las operaciones asociadas llamadas por la aplicación son realizadas por el mismo (Torres Remon, 2012).

Sus principales propiedades son las que se muestran en la Tabla 17 y sus métodos son aquellos mencionados en la Tabla 18.

Tabla 17. Descripción de propiedades principales de la clase SqlTransaction.

Propiedades	Descripción
Connection	Obtiene el objeto SQLConnection asociado con la transacción y si ya ésta ya no es válida devuelve Nothing.
DbConnection	Especifica el objeto DBConnection que está asociado con la transacción. Esta se hereda de DbTransaction.
IsolationLevel	Especifica el IsolationLevel o nivel de aislamiento para la transacción. Ésta invalida a DbTransaction.IsolationLevel.

Fuente: Torres Remon (2012).

Tabla 18. Descripción de métodos principales de la clase SqlTransaction.

Método	Descripción
Commit	Confirma la transacción de base de datos. Esta invalida a DbTransaction.Commit
Dispose	Libera los recursos no administrados que utiliza DbTransaction de cual se hereda.
Finalize	Permite que un objeto que se hereda de Object el cual intenta liberar recursos y realizar otras operaciones de limpieza antes de ser reclamado por la recolección de elementos que no se utilizan.
GetType	Obtiene el objeto Type de la instancia actual. Este se hereda de Object.
Rollback	Se encarga de deshacer una transacción en estado pendiente. Invalida a DbTransaction.Rollback.
Rollback(String)	Deshace la transacción con un nombre o punto de almacenamiento especificado, que se encuentre en estado pendiente.

ToString	Devuelve una cadena representante del objeto actual. Este hereda de Object.

Fuente: Torres Remon (2012).

BeginTransaction

Es un método de la clase *Transaction*, el cual inicia una transacción de la base de datos el cual es sobrecargado por *BeginTransaction (String)* el cual lleva especificado el nombre de la transacción en *String*, *BeginTransaction (isolationLevel)* que inicia la transacción con un nivel especificado de aislamiento, *BeginTransaction* que solo inicia la transacción y BeginTransaction (IsolationLevel, String) que inicia tanto con el nivel de aislamiento especificado como con el nombre de la transacción (Torres Remon, 2012).

Commit

Es un método de la clase SQLTransaction que se encarga de confirmar la transacción de base de datos, y una vez realizada no es posible deshacerla ya que las modificaciones se han convertido en permanentes dentro de la base de datos (Torres Remon, 2012).

Rollback()

Es un método de la clase SQLTransaction el cual deshace una transacción desde un estado pendiente y esta sobrecargado por la instrucción Rollback y Rollback (String) (Torres Remon, 2012).

TABLE ADAPTER

Configuración de TableAdapter

Para la configuración de un TableAdapter, se definen consultas según las necesidades de la aplicación y respecto a estas se configura su acceso a la base de datos para determinada entidad o tabla (Información general sobre TableAdapter, s.f).

A comparación de los adaptadores de datos estándar, éste TableAdapter puede contener varias consultas que se rellenen de datos en las tablas asociadas, a fin de que cada consulta retorne su mismo esquema. Por ejemplo, si se tiene una tabla llamada Pedidos y se

quiere consultar los pedidos cuya fecha coincida en cierto lapso de registro se puede crear una consulta que rellene con dichos datos (Información general sobre TableAdapter, s.f).

Las consultas de TableAdapter son instrucciones SQL o procedimientos almacenados que se pueden ejecutar mediante una aplicación las cuales pueden rellenar una tabla de datos (consultas Fill y FillBy) o devolver nuevas tablas de datos rellenadas con los datos devueltos por la consulta (consultas GetData y GetDataBy) (Información general sobre TableAdapter, s.f).

Para la asignación del nombre al TableAdapter al momento en que se crea, está basado en el nombre de la tabla con la que se está trabajando, es decir, si se crea un TableAdapter basado en la tabla *Clientes* de la base de datos, los TableAdapter se denominan como consecuencia ClientesTableAdapter. En caso que se desee cambiar el nombre de la clase del TableAdapter se puede hacer utilizando la propiedad *Name* en el *Diseñador de DataSet* (Información general sobre TableAdapter, s.f).

En la Tabla 19 se describen las propiedades y métodos del TableAdapter.

Tabla 19. Descripción de métodos y propiedades de TableAdapter.

Nombre	Descripción
TableAdapter.Fill	Se encarga de rellenar de datos que regresa como resultado de un comando SELECT de una tabla que se encuentra asociada al TableAdapter.
TableAdapter.Update	Envía los cambios al origen de datos (BD) y devuelve un entero que representa el número de filas a las que afecta la actualización.
TableAdapter.GetData	Devuelve un DateTable relleno con datos.
TableAdapter.Insert	Crea una nueva fila al DataTable.
TableAdapter.ClearBeforeFill	Determina si se vacía una tabla de datos antes de llamar a uno de los métodos Fill.

Fuente: Información general sobre TableAdapter (s.f).

DataTable

En la biblioteca ADO.NET se integra un objeto central que es DataTable que incluye entre sus objetos a DataSet y DataView (Torres Remon, 2012).

Cabe destacar que al tener acceso a los objetos del DataTable, estos tienen distinción entre mayúsculas y minúsculas condicionalmente (Torres Remon, 2012).

DateTable representa una tabla de memoria de una base de datos el cual pertenece al espacio de nombres System.Data (Microsoft DataTable, s.f).

En la Tabla 20 se muestran sus métodos constructores. Algunas de las propiedades del DataTable se muestran en la Tabla 21.

Tabla 20. Descripción de métodos constructores de la clase DataTable.

Nombre	Descripción
DataTable()	Realiza la inicialización de una nueva instancia de la clase DataTable (sin argumentos).
DataTable(SerializationInfo, StreamingContext)	Inicializa una nueva instancia de la clase DataTable con parámetros SerializationInfo y StreamingContext.
DataTable(String)	Inicializa una nueva instancia de la clase DataTable en conjunto con la especificación del nombre de la tabla.
DataTable(String, String)	Inicializa una nueva instancia de la clase DataTable en conjunto con la especificación del nombre de la tabla, así como el espacio de nombres de clase.

Fuente: Microsoft DataTable (s.f).

Tabla 21. Descripción de las propiedades de la clase DataTable.

Nombre	Descripción
Case Sensitive	Determina si las comparaciones de cadenas realizadas dentro de la tabla distinguen mayúsculas y minúsculas.
Columns	Obtiene el conjunto de columnas que pertenecen a cierta tabla.
Constraints	Obtiene el conjunto de restricciones mantenidas por cierta tabla.

DataSet	Obtiene el DataSet al que pertenece a cierta tabla.
IsInitialized	Obtiene un valor que indica si el DataTable se inicializa.
Locale	Obtiene o establece la información de la configuración regional que se utiliza para comparar las cadenas de la tabla.
Namespace	Obtiene o establece el espacio de nombres para la representación XML de los datos almacenados en el DataTable.
PrimaryKey	Obtiene o establece una matriz de columnas que funcionan como claves principales para la tabla de datos.
Rows	Obtiene el conjunto de filas que pertenecen a determinada tabla.
TableName	Obtiene o establece el nombre del DataTable.

Fuente: Microsoft DataTable (s.f).

Sus principales métodos son los que se muestran a continuación en la Tabla 22.

Tabla 22. Desripción de métodos de la clase DataTable.

Nombre	Descripción
BeginInit()	Inicializa un objeto DataTable empleado en un formulario u otro componente, la cual se produce en tiempo real.
Clear()	Elimina el DataTable de los datos.
Clone()	Clona la estructura del DataTable, incluidas sus esquemas y restricciones.
Copy()	Copia la estructura y datos del DataTable.
CreateDataReader()	Regresa un DataTableReader correspondientes a los datos en el DataTable.
CreateInstance()	Infraestructura. Crea una nueva instancia de DataTable.
Dispose()	Libera los recursos no administrados que usa MarshalByValueComponent y libera los recursos administrados de forma opcional. (Heredado de MarshalByValueComponent).
EndInit()	Termina o interrumpe la inicialización de un objeto DataTable que se utiliza en un formulario u otro componente.
Equals(Object)	Determina si el objeto especificado es igual al objeto actual.

Finalize()	Permite que un objeto intente liberar recursos y realizar otras operaciones de limpieza antes de ser reclamado por el recolector de basura. (Heredado de MarshalByValueComponent).
GetChanges()	Obtiene una copia de DataTable que contiene los cambios realizados desde que se cargó AcceptChanges por última vez.
GetErrors()	Obtiene una matriz de objetos DataRow que contienen errores.
GetHasCode()	Sirve como la función predeterminada de hash (Heredado de Object).
GetRowType()	Obtiene el tipo de fila.
GetType()	Obtiene el Type de la instancia actual. Es heredado de Object.
NewRow()	Crea un nuevo DataRow con el mismo esquema de la tabla.
Reset()	Restablece el DataTable a su estado original, quita todos los datos, índices, relaciones y columnas de la tabla. Si un conjunto de datos incluye una tabla de datos, la tabla todavía será parte del conjunto de datos después de restablece la tabla.
Select()	Obtiene una matriz de los objetos DataRow.
Select(String)	Obtiene una matriz de los objetos DataRow que coincides con los criterios de filtro.
ToString()	Obtiene el TableName y DisplayExpression, si hay alguna, como cadena concatenada (Invalida MarshalByValueComponent.ToString()).

Fuente: Microsoft DataTable (s.f).

DataAdapter

Es una clase representante de un conjunto de comandos SQL así como una conexión a la base de datos que se usan para rellenar DataSet y actualizar el origen de datos. A continuación se describen constructores, propiedades y métodos en las Tablas 23, 24 y 25, respectivamente (Microsoft DataAdapter, s.f).

Tabla 23. Descripción de constructores de la clase DataAdapter.

Nombre	Descripción
DataAdapter()	Inicializa una nueva instancia de la clase DataAdapter.
DataAdapter(DataAdapter)	Inicializa una nueva instancia de la clase DataAdapter a partir de un objeto existente siempre que sea del mismo tipo.

Fuente: Microsoft DataAdapter (s.f).

Tabla 24. Descripción de propiedades de la clase DataAdapter.

Nombre	Descripción
AcceptChangesDuringFill	Obtiene o establece un valor que indica si AcceptChanges se llama desde un DataRow después de ser agregado a DataTable en alguna de las operaciones.
AcceptChangesDuringUpdate	Obtiene o establece si AcceptChanges se llama durante una Update.
CanRaiseEvents	Obtiene un valor que demuestra si el componente puede provocar un evento. (Heredado de Component).
Container	Obtiene IContainer que contiene Component. (Heredado de Component).
ContinueUpdateOnError	Obtiene o establece un valor que especifíca si generar una excepción cuando un error se produce durante una actualización de fila.
DesignMode	Obtiene un valor que indica si Component está actualmente en modo de diseño. (Heredado de Component).
Events	Obtiene la lista de controladores de eventos que se agregan a Component. (Heredado de Component).

FillLoapOption	Obtiene o establece el LoadOption el cual determina cómo rellena el adaptador el DataTable desde el DbDataReader.
MissingMappingAction	Determina la acción que se realizará cuando los datos de entrada no tienen una tabla o columna correspondiente.
Site	Obtiene o establece la ISite de Component. (Heredado de Component).
TableMappings	Obtiene una colección que proporciona la asignación principal entre una tabla de origen y un DataTable.

Fuente: Microsoft DataAdapter (s.f).

Tabla 25. Descripción de los principales métodos de la clase DataAdapter.

Nombre	Descripción
CreateObjRef(Type)	Crea un objeto que contiene toda la información necesaria para generar un proxy para comunicarse con un objeto remoto. (Heredado de MarshalByRefObject).
Dispose()	Libera los recursos que usa Component. (Heredado de Component).
Dispose(Boolean)	Libera los recursos no administrados que usa DataAdapter y lo que si son administrados, de forma opcional. (Invalida Component.Dispose (Boolean)).
Equals(Object)	Determina si el objeto especificado es igual al objeto actual. (Heredado de Object).
Fill(DataSet)	Agrega o actualiza filas en el DataSet que coincidan con las del origen de datos.
Fill(DataSet, String, IDataReader, Int32, Int32)	Agrega o actualiza filas en un intervalo especificado en el DataSet que coincidan con las de origen de datos utilizando el DataSet y DataTable.

Fill(DataTable, IDataReader)	Agrega o actualiza filas en el DataTable que coincidan con las de origen de datos utilizando el DataTable nombre y especificado IDataReader.
Finalize()	Libera recursos no administrados y realiza otras operaciones de limpieza antes de que se reclame el elemento Component durante la recolección de elementos no utilizados. (Heredado de Component).
GetFillParameters()	Obtiene los parámetros establecidos por el usuario al ejecutar una instrucción SELECT de SQL.
GetHashCode()	Sirve como la función hash predeterminada. (Heredado de Object).
GetType()	Obtiene el Type de la instancia actual. (Heredado de Object).
OnFillError(FillErrorEventArgs)	Se invoca cuando se produce un error durante una Fill.
ResetFillLoadOption()	Restablece FillLoadOption a su estado predeterminado y causas DataAdapter.Fill respetando AcceptChangesDuringFill.
ToString()	Devuelve un String que contiene el nombre de la Component, si existe. Este método no se debe invalidar. (Heredado de Component).
Update(DataSet)	Llama a las instrucciones INSERT, UPDATE o DELETE respectivas para cada inserta, actualiza o elimina la fila de la manera especificada DataSet desde un DataTable denominado "Table".

Fuente: Microsoft DataAdapter (s.f).

ConnectionString

El objeto Connection heredado de DbConnection viene definido por cada proveedor de datos en .NET Framework, así como su propiedad ConnectionString. La sintaxis varía de acuerdo a cada uno de estos proveedores la cual se presenta en la Tabla 26 (Connection String Syntax, s.f).

Tabla 26. Descripción de proveedores de acceso a datos.

Nombre	Descripción
System.Data.SqlClient	Proveedor de acceso a datos para Microsoft SQL Server.
System.Data.OleDb	Proveedor de acceso a datos para fuentes de datos expuestas utilizando OleDb.
System.Data.Odbc	Proveedor de acceso a datos para fuentes de datos expuestas utilizando ODBC.
System.Data.Oracle.Client	Proveedor de acceso a datos para la versión 8.1.7 de Oracle.

Fuente: Connection String Syntax (s.f).

DataView

Es una clase que permite el enlace de los datos con formularios (Windows Forms o Web Forms), de igual manera puede presentar un subconjunto de datos de un DataTable, lo cual permite tener dos controles enlazados a un objeto DataTable y muestren diversas versiones de los datos. Un ejemplo de su funcionamiento es que un control DataView muestra las filas de la tabla y otro muestre solo las filas que se eliminen de un DataTable (Torres Remon, 2012).

Dentro de la clase DataTable tiene la propiedad DefaultView la cual se encarga de devolver un objeto DataView predeterminado para una tabla. Por ejemplo, para crear en la tabla una vista personalizada, es necesario establecer la propiedad *RowFilter* en el objeto DataView que devuelve precisamente la propiedad DefaultView (Torres Remon, 2012).

Existen más propiedades funcionales como *RowFilter* y *Sort* las cuales sirven funcionan para crear un vista filtrada y ordenada de datos, posteriormente la propiedad *Item* devolverá un único DataRowView. De igual manera se pueden eliminar y agregar de un conjunto de filas mediante los métodos Delete y AddNew, respectivamente. Al utilizarse estos métodos, se establece la propiedad RowStateFilter para especificar que DataView muestre sólo las filas eliminadas o nuevas (Torres Remon, 2012).

A continuación se describen los métodos constructores, propiedades y métodos principales de la clase DataView en las Tablas 27, 28 y 29, respectivamente.

Tabla 27. Descripción de métodos constructores de la clase DataView.

Nombre	Descripción
DataView()	Inicializa una nueva instancia de DataView.
DataView (DataTable)	Inicializa una nueva instancia de DataView con la especificación del objeto DataTable.
DataView(DataTable, String, String,DataViewRowState)	Inicializa una nueva instancia de DataView con la especificación del objeto DataTable, RowFilter, Sort y DataViewRowState.

Fuente: Torres Remon (2012).

Tabla 28. Descripción de las propiedades de la clase DataView.

Nombre	Descripción
Count	Obtiene el número de registros de la DataView después de la aplicación de RowFilter y RowStateFilter.
IsOpen	Obtiene un valor que indica si el origen de datos está abierto actualmente y proyecta vistas de datos en la DataTable.
Item	Obtiene una fila de datos de una tabla especificada.
RowFilter	Obtiene o establece la expresión usada para filtrar las filas que se deben mostrar en DataView.
Sort	Obtiene o establece el criterio de ordenación de columnas y el criterio de ordenación para el DataView.
Table	Obtiene o establece el DataTable de origen.

Fuente: Torres Remon (2012).

Tabla 29. Descripción de métodos de la clase DataView.

Nombre	Descripción
AddNew	Agrega una nueva fila a DataView.
Close	Cierra el objeto DataView.
Delete	Elimina una fila en el índice especificado.
Dispose()	Libera todos los recursos que usa MarshalByValueComponent. (Heredado de MarshalByValueComponent).

Equals(DataView)	Determina si las instancias de DataView especificadas se consideran iguales.
Finalize	Permite que un objeto intente liberar recursos y realizar otras operaciones de limpieza antes de ser reclamado por el recolector de basura. (Heredado de MarshalByValueComponent).
Find(Object)	Encuentra una fila en la DataView por el valor de clave de ordenación especificado.
Find(Object[])	Encuentra una fila en la DataView por los valores de clave de ordenación especificados.
FinRows(Object)	Devuelve una matriz de DataRowView objetos cuyas columnas coinciden con el valor de criterio de ordenación especificado.
Open	Abre un DataView.
ToTable()	Crea y devuelve una nueva DataTable basándose en las filas de una DataView existente.

Fuente: Torres Remon (2012).

DataRowView

La clase DataRowView representa una vista personalizada de un elemento DataRow. Sus propiedades y métodos se describen en las Tablas 30 y 31, respectivamente (Microsoft DataRowView, s.f).

Tabla 30. Descripción de propiedades de la clase DataRowView.

Nombre	Descripción
DataView	Obtiene el DataView al que pertenece esta fila.
IsEdit	Indica si la fila está en modo de edición.
IsNew	Indica si un DataRowView nuevo.
Item()	Obtiene o establece un valor en una columna especificada
Item(String)	Obtiene o establece un valor en una columna especificada.
Row	Obtiene el DataRow que se está visualizando.
RowVersion	Obtiene la descripción de la versión actual de la DataRow.

Fuente: Microsoft DataRowView (s.f).

Tabla 31. Descripción de métodos de la clase DataRowView.

Nombre	Descripción
BeginEdit()	Inicia un procedimiento de edición.
CancelEdit()	Cancela un procedimiento de edición.
CreateChildView(DataRelation)	Devuelve un DataView para el elemento secundario DataTable con el elemento secundario especificado DataRelation.
CreateChildView(String)	Devuelve un DataView para el elemento secundario DataTable con el elemento secundario especificado DataRelation nombre.
CreateChildView(String, boolean)	Devuelve un DataView para el elemento secundario DataTable con los valores especificados DataRelation nombre y principal.
CreateChildView(DataRelation,Boolean)	Devuelve un DataView para el elemento secundario DataTable con los valores especificados DataRelation primaria.
Delete()	Elimina una fila.
EndEdit()	Confirma los cambios en la base de DataRow y finaliza la sesión de edición que se inició con BeginEdit. Use CancelEdit para descartar los cambios realizados en el DataRow.
Equals(Object)	Obtiene un valor que indica si el actual DataRowView es idéntico al objeto especificado (Invalida Object.Equals(Object)).
Finalize()	Permite que un objeto intente liberar recursos y realizar otras operaciones de limpieza antes de ser reclamado por el recolector de basura (Heredado de Object).
GetHashCode()	Devuelve el código hash de la DataRow objeto (Invalida Object.GetHashCode()).

GetType()	Obtiene el Type de la instancia actual. (Heredado de Object).
MemberwiseClone()	Crea una copia superficial del Object actual. (Heredado de Object).
ToString()	Devuelve una cadena que representa al objeto actual. (Heredado de Object).

Fuente: Microsoft DataRowView (s.f).

Como ejemplo del uso de esta clase, lo siguiente describe el evento CellClick de un DataGridView para capturar el índice de un cliente seleccionado por el usuario.

```
Dim i% = dgClientes.CurrentRow.Index
```

Se declara dvCli de tipo DataView para poder tener acceso a los registros de los clientes y así poder filtrar sus pedidos.

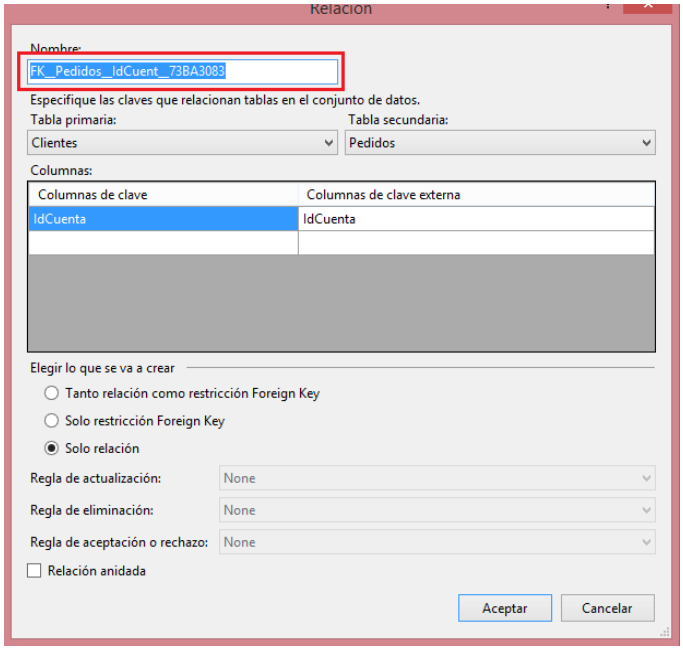
```
Dim dvCli As DataView = ds.Clientes.DefaultView
```

Se declara drv de tipo DataRowView para comparar la fila por la fila de la relación que tiene como Pedido y así obtener solo los registros coincidentes y enviarlos al dgPedidos.

```
Dim drvCli As DataRowView = dvCli(i)
dgPedidos.DataSource = drvCli.CreateChildView("FK__Pedidos__IdCuent__73BA3083")
lblTotal.Text = dgPedidos.RowCount - 1
```

Para obtener el parámetro considerado por el CreateChildView es el nombre asignado en la relación entre la tabla Cliente y Pedidos, para poder obtener este nombre se presionó doble clic sobre la línea de relación mostrado en el asistente de DataSet, copie el nombre y pegue en el parámetro de dicho elemento (en el código). Véase la Figura 20.

Figura 20. Extracción de parámetro para CreateChildView.



Quedando la instrucción de la siguiente manera:

```
dgPedidos.DataSource = drvCli.CreateChildView("FK__Pedidos__IdCuent__73BA3083")
```

LINQ TO SQL

Creación de proyecto, clases y conexión para LINQ to SQL.

En Visual Studio se pueden crear clases de LINQ to SQL que representen las tablas creadas en SQL Server las cuales con utilizadas por el Diseñador O/R el cual consta de dos distintas partes en el área de diseño, a la izquierda de muestran las entidades, asociaciones y jerarquías de herencia y en la derecha para los métodos de la clase DataContext, procedimientos almacenados y funciones (Walkthrough: Creating LINQ to SQL Classes (O/R Designer, s.f).

Este Diseñador Relacional de Objetos provee de una superficie visual de diseño para crear clases de entidad LINQ to SQL y relaciones basadas en objetos de una base de datos. También brinda funcionalidad para asignar funciones y procedimientos almacenados a los métodos DataContext para de esta forma proveer de datos y llenar las clases de entidad (LINQ to SQL Tools in Visual Studio, s.f).

Para la utilización de aplicaciones basadas en LINQ es necesario que la configuración se realice en el entorno de la misma, por lo que hay que realizar una serie de pasos previos a la elaboración de un sistema o aplicación, como la creación de clase de datos (en O/R Designer) y conexión.

Configuración de app.config

Cuando se configura una aplicación con los accesos a datos por medio de LINQ to SQL es necesario también realizar modificaciones al archivo de acceso a bases de datos app.config el cual es muy útil en situaciones en las que es posible que se edite la cadena de conexión después de implementar la aplicación. Dicha configuración se realiza a como se muestra en el siguiente fragmento de código:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
        <add name="LinQTo.My.MySettings.SistemaConnectionString"
connectionString="Data Source=LIZETH\SQLSERVER;Initial
Catalog=Sistema;Integrated Security=True"
            providerName="System.Data.SqlClient" />
    </connectionStrings>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2" />
    </startup>
</configuration>
```

Implementación de consultas en LINQ

En la historia de desarrollo de aplicaciones se han ido desarrollando diversos lenguajes para cada origen de dato, como SQL (para bases de datos relacionales), XQuery (para XML) entre otros, de manera que un desarrollador de aplicaciones debe aprender para cada lenguaje su forma de acceso a datos de acuerdo a cada origen o formato admitido. Pero LINQ simplifica esta situación ya que provee un modelo para trabajar con los datos de varios formatos y orígenes de datos ya que en sus consultas trabaja con objetos, utilizando de esta forma, los mismos modelos de codificación básicos para las consultas y la transformación de los datos de documentos XML, bases de datos, conjuntos de datos y entidades de ADO.NET, así como las colecciones .NET Framework y cualquier otro origen o formato que exista como proveedor LINQ (Torres Remon, 2012).

Para la realización de consultas se determinan 3 etapas las cuales se componen de los siguientes elementos:

1. Obtención de orígenes de datos ya sea uno sólo o varios.
2. Generación de la consulta.
3. Ejecución de la consulta.

Operaciones básicas de una consulta LINQ

Torres Remon (2012) indica que para implementar una consulta condicionada en LINQ to SQL es necesario utilizar ciertas instrucciones o clausulas específicas las cuales son:

- Clausula From:
  Se utiliza para especificar el origen de datos del cual se desea extraer la información, por lo que esta instrucción es la primera que se determina. Por ejemplo:
  Dim cons = From c In DataContext
- Clausula Where:
  Se encarga de filtrar elementos de un origen de origen los cuales formarán parte del resultado. Por ejemplo:
  Dim clientes = From c In grupo Where c = "Carlos"
- Clausula Order By:

Se utiliza para ordenar los elementos de la secuencia resultante de forma ascendente (Ascending) o descendiente (Descending) según convenga.

- Clausula Select:

  Especifica la forma y el contenido (el cómo y qué) de los elementos resultantes. Cuando ésta genera algo distinto a una copia de un elemento origen, se determina *proyección*. El caso de la selección múltiple de campos de origen de datos, maneja dos opciones:

  1. Especificar los campos que se desea que aparezca en el resultado. En éste caso el compilador definirá un anónimo que tiene esos campos como propiedades.
  2. Definir un tipo con nombre que contenga los campos concretos a incluir en el resultado y cree e inicialice instancias del tipo. Esta opción es útil cuando se desee utilizar los resultados de forma individual fuera de la colección o cuando se usen como parámetros en llamadas a método.

- Clausula Join:

  Se usan para combinar más de un origen de datos en la cláusula From de varias formas. Por ejemplo:

```
Dim valorCombo = cboTipo.SelectedValue.ToString

Dim articulos = From t In daAge.Tipos Join a In daAge.Articulos On t.IdTipo
Equals a.IdTipo

        Where t.Concepto = valorCombo Select a.IdCodigo, a.IdTipo,
a.Descripcion, a.Precio

dgArticulos.DataSource = artículos
```

  Nota: La cláusula Join es equivalente a INNER JOIN en SQL.

- Clausula Group By:

  Se encarga de agrupar los elementos de un resultado proporcionados por una consulta.

DataContext

La clase DataContext es el principal punto de entrada para el marco de trabajo de LINQ to SQL ya que es el origen de todas las entidades asignadas en una conexión de la base

de datos. Mantiene seguimiento en los cambios realizados en las entidades recuperadas así como una *memoria caché de identidad* para garantizar que las entidades recuperadas más de una vez se representan utilizando una misma instancia de objeto (Torres Remon, 2012).

Esta clase hace uso del espacio de nombres System.Data.Linq. (DataContext (Clase), s.f).

Dentro de una aplicación típica de LINQ to SQL crean instancias de la clase DataContext en el ámbito de métodos o como miembro de clases de duración corta, representantes de un conjunto lógico de operaciones de bases de datos relacionadas (DataContext (Clase), s.f).

A continuación se describen los principales métodos constructores, propiedades y principales métodos en la Tabla 32, 33 y 34, respectivamente.

Tabla 32. Descripción de métodos constructores de la clase DataContext.

Nombre	Descripción
DataContext(IDbConnection)	Inicializa una nueva instancia de la clase DataContext haciendo referencia a la conexión usada por el .NET Framework.
DataContext(String)	Inicializa una nueva instancia de la clase DataContext haciendo referencia a un origen de archivo.
DataContext(IDbConnection, MappingSource)	Inicializa una nueva instancia de la clase DataContext haciendo referencia a una conexión y un origen de asignación.
DataContext(String, MappingSource)	Inicializa una nueva instancia de la clase DataContext haciendo referencia a un origen de archivo y un origen de asignación.

Fuente: DataContext (Clase) (s.f).

Tabla 33. Descripción de propiedades de la clase DataContext.

Nombre	Descripción
ChangeConflicts	Obtiene una colección de objetos que produjeron conflictos de simultaneidad cuando se llamó a SubmitChanges.
CommandTimeout	Obtiene o establece un valor que incrementa el tiempo de espera para las consultas que se habría agotado durante el período de tiempo de espera predeterminado.
Connection	Obtiene la conexión usada por el marco de trabajo.
DeferredLoadingEnabled	Obtiene o establece un valor que indica si se presentará retraso en la carga de las relaciones uno a varios o uno a uno.
LoadOptions	Obtiene o establece el DataLoadOptions asociado al DataContext.
Log	Obtiene o establece el destino para escribir la consulta o comando SQL.
Mapping	Obtiene el MetaModel en el que se basa la asignación.
ObjectTrackingEnabled	Obtiene o establece un valor que indica si el seguimiento de los objetos está habilitado.
Transaction	Obtiene o establece una transacción local para que .NET Framework pueda tener acceso a la base de datos.

Fuente: DataContext (Clase) (s.f).

Tabla 34. Descripción de principales métodos de la clase DataContext.

Nombre	Descripción
CreateDatabase	Crea una base de datos en el servidor.
CreateMethodCallQuery (Of TResult)	Ejecuta la función de base de datos con valores de tabla asociada a la especificación del método CLR.
DatabaseExist	Determina si se puede abrir la base de datos asociada.
DeleteDatabase	Elimina la base de datos asociada.

Dispose	Libera los recursos usados por la instancia actual de la clase DataContext.
Dispose(Boolean)	Libera los recursos no administrados utilizados por la clase DataContext y, opcionalmente, libera los administrados.
Equals(Object)	Determina si el objeto especificado es igual al objeto actual. (Heredado de Object).
ExecuteCommand	Ejecuta comandos SQL directamente en la base de datos.
ExecuteDynamicDelete	Se ejecuta dentro de los métodos para invalidación de eliminación, dando a LINQ to SQL la tarea de generar y ejecutar SQL dinámico para operaciones de eliminación.
ExecuteDynamicInsert	Se ejecuta dentro de los métodos para invalidación de inserción, dando a LINQ to SQL la tarea de generar y ejecutar SQL dinámico para operaciones de inserción.
ExecuteDynamicUpdate	Se ejecuta dentro de los métodos para invalidación de actualización, dando a LINQ to SQL la tarea de generar y ejecutar SQL dinámico para operaciones de actualización.
ExecuteMethodCall	Ejecuta el procedimiento almacenado de la base de datos o una función escalar asociada al método CLR especificado.
ExecuteQuery(Type, String, Object())	Ejecuta consultas SQL directamente en la base de datos.
ExecuteQuery(Of TResult)(String, Object())	Ejecuta consultas SQL directamente en la base de datos devolviendo objetos.
Finalize	Permite que un objeto intente liberar recursos y realizar otras operaciones de limpieza antes de ser reclamado por el recolector elementos basura. (Heredado de Object).
GetChangeSet	Obtiene los objetos modificados realizando seguimiento con DataContext.
GetCommand	Obtiene la información sobre los comandos SQL generados por LINQ to SQL.
GetTable(Type)	Devuelve una colección de objetos de un tipo determinado definido por el parámetro type.

GetTable(Of TEntity)	Devuelve una colección de objetos de un tipo determinado definidos por el parámetro TEntity como tipo.
GetType	Obtiene el Type de la instancia actual. (Se hereda de Object).
Refresh(RefreshMode, IEnumerable)	Actualiza una colección de objetos de entidad según el modo especificado.
SubmitChanges	Calcula el conjunto de objetos modificados que van a ser insertados, actualizados o eliminados, ejecutando los comandos adecuados para implementar los cambios en la base de datos.
ToString	Devuelve una cadena que representa al objeto actual. (Se hereda de Object).

Fuente: DataContext (Clase) (s.f).

Método InsertOnSubmit

Agrega una entidad en estado pendiente (insert) al Table (TEntity) (Método Table (Of TEntity).InsertOnSubmit (TEntity), s.f).

Por ejemplo:

```
For i = 0 To dgvCarrito.Rows.Count - 2
    Dim objDet As New DetallePedido
    objDet.IdPedido = txtIdPedido.Text
    objDet.IdCodigo = dgvCarrito.Item(0, i).Value
    objDet.Precio = CDec(dgvCarrito.Item(3, i).Value)
    objDet.Cantidad = CInt(dgvCarrito.Item(4, i).Value)
    dcSis.DetallePedido.InsertOnSubmit(objDet)
    dcSis.SubmitChanges()
Next
MsgBox("PEDIDO REGISTRADO")
```

CAPÍTULO 4. METODOLOGÍA

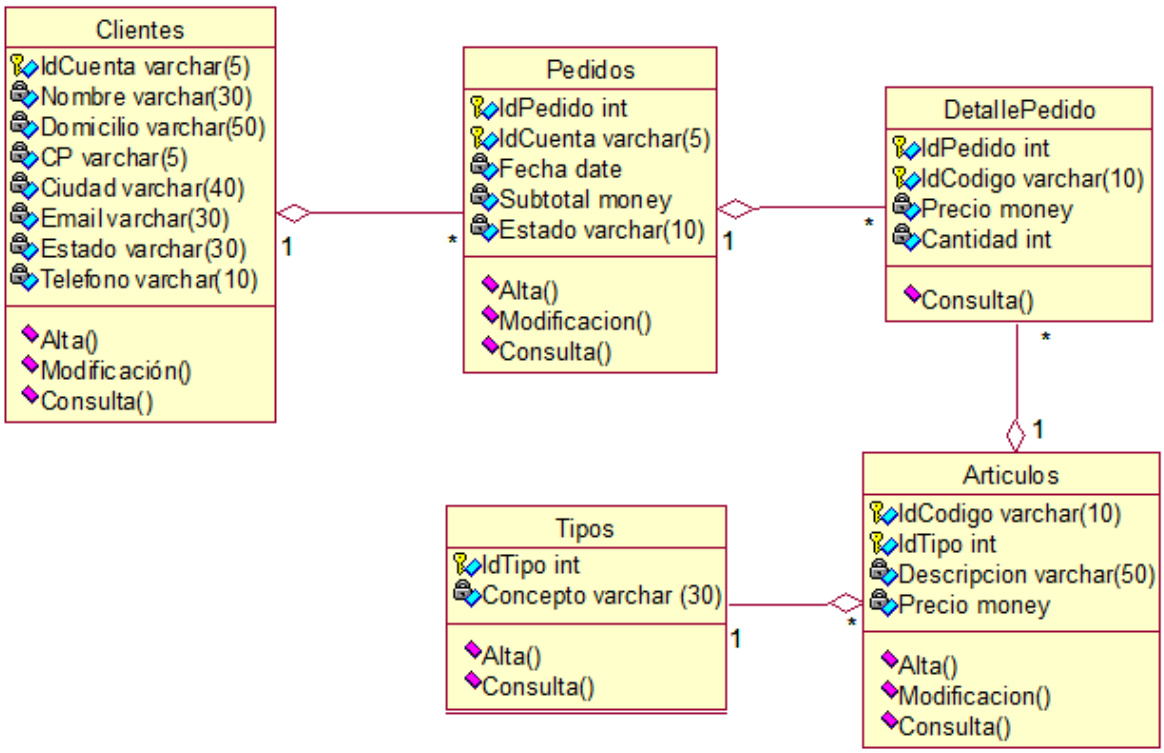
4.1 ANÁLISIS

Para realizar la investigación correspondiente a las técnicas de acceso a datos que ofrece Visual Studio Community 2015 fue necesario disponer de un modelo base, el cual es planteado en el problema siguiente: Desarrollar un sistema de información que permita el control de pedidos que realizan los clientes sobre los artículos que tiene a la venta una determinada empresa.

4.1.1 DIAGRAMA DE CLASES

El diagrama de clases se muestra a continuación. Observar la Figura 21.

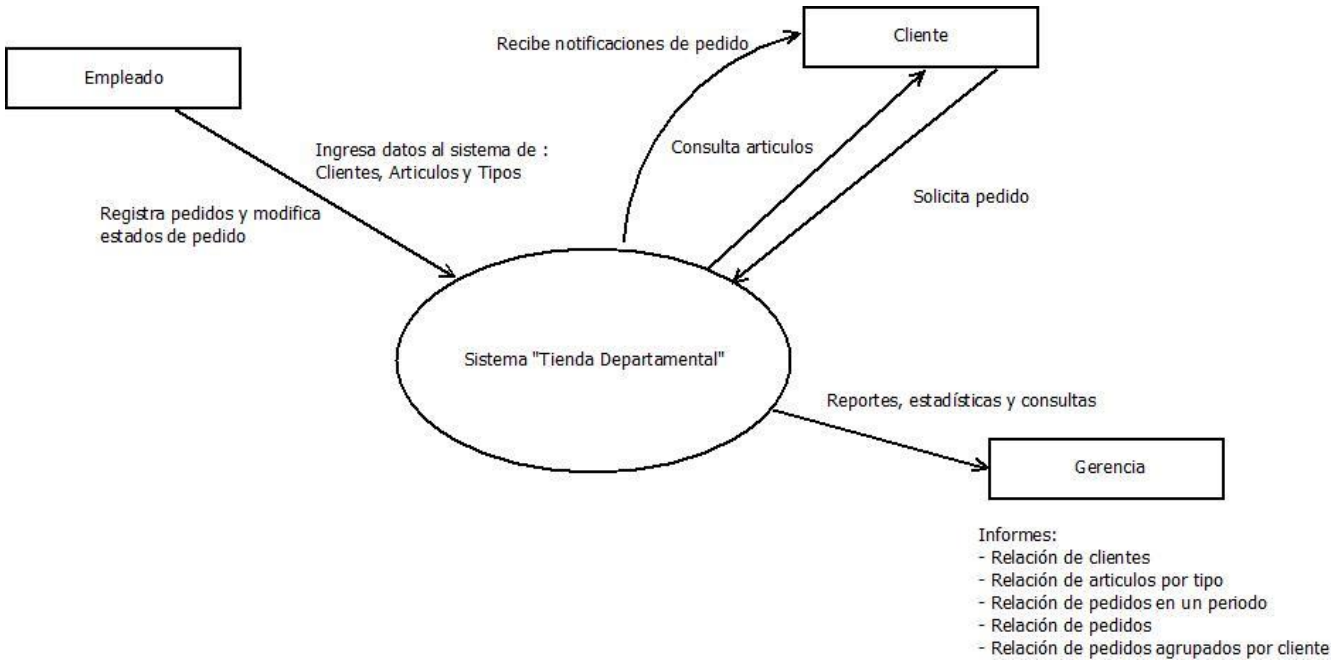
Figura 21. Diagrama de clases del Sistema para tienda departamental.



4.1.2 DIAGRAMA DE CONTEXTO

A continuación se muestra el diagrama de contexto en la Figura 22.

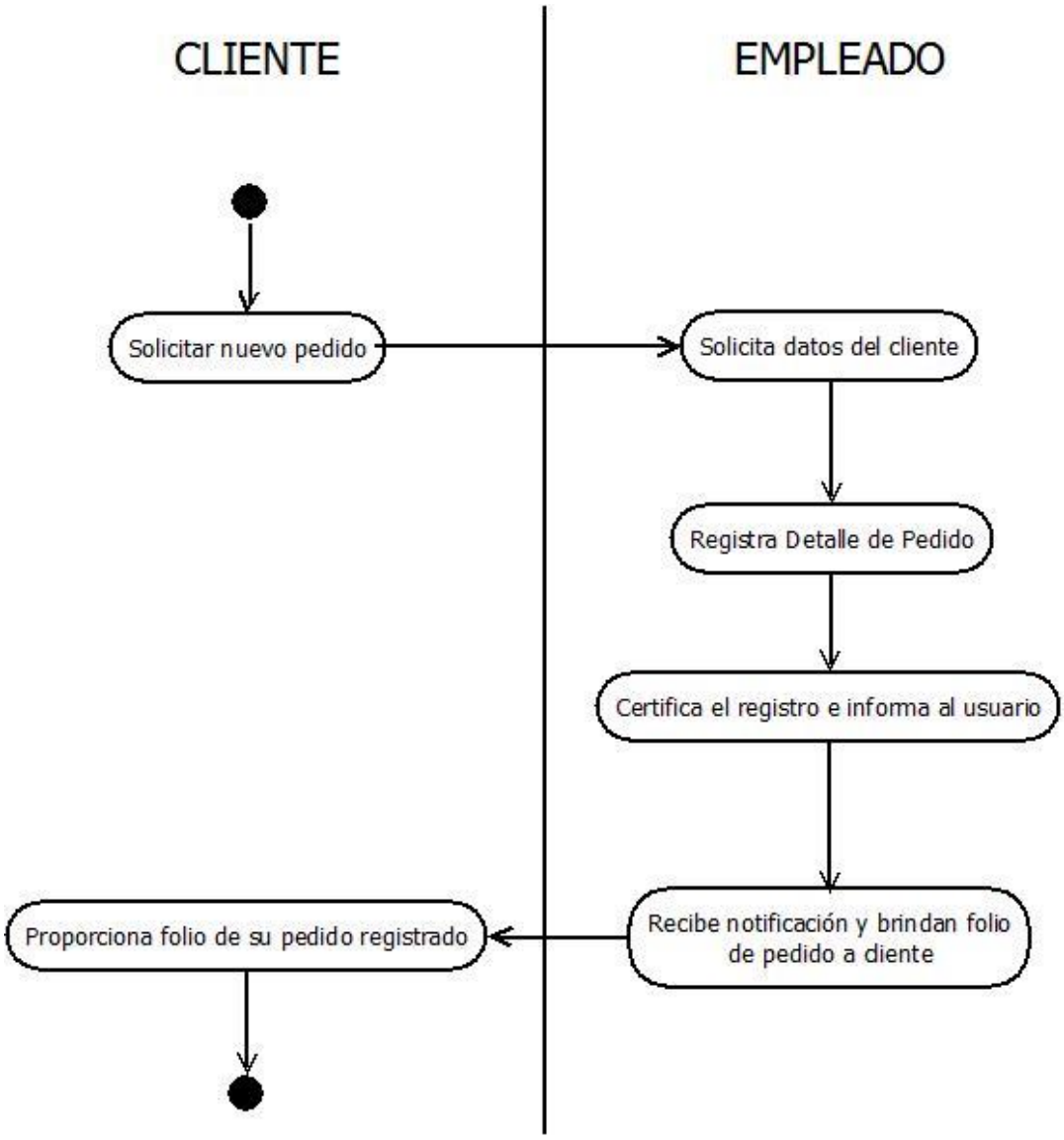
Figura 22. Diagrama de contexto.



4.1.3 DIAGRAMA DE ACTIVIDAD PARA CASO DE USO “REGISTRAR PEDIDO”

En la Figura 23 se muestra el diagrama de actividad para el caso de uso “Registrar Pedido”.

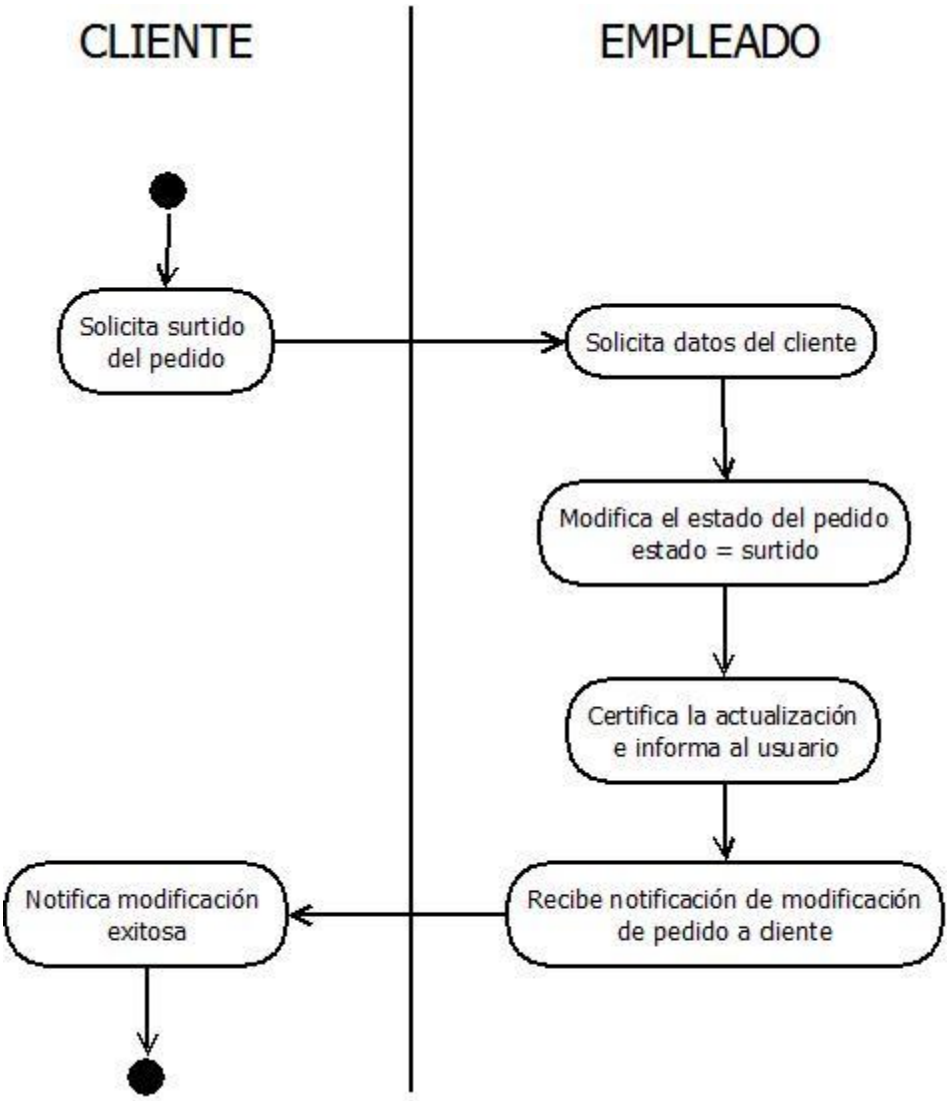
Figura 23. Diagrama de actividad para caso de uso "Registrar Pedido".



4.1.4 DIAGRAMA DE ACTIVIDAD PARA CASO DE USO “CAMBIAR ESTADO DE PEDIDO A SURTIDO”

En la Figura 24 se muestra el diagrama de actividad para el caso de uso “Cambiar estado de Pedido Surtido”.

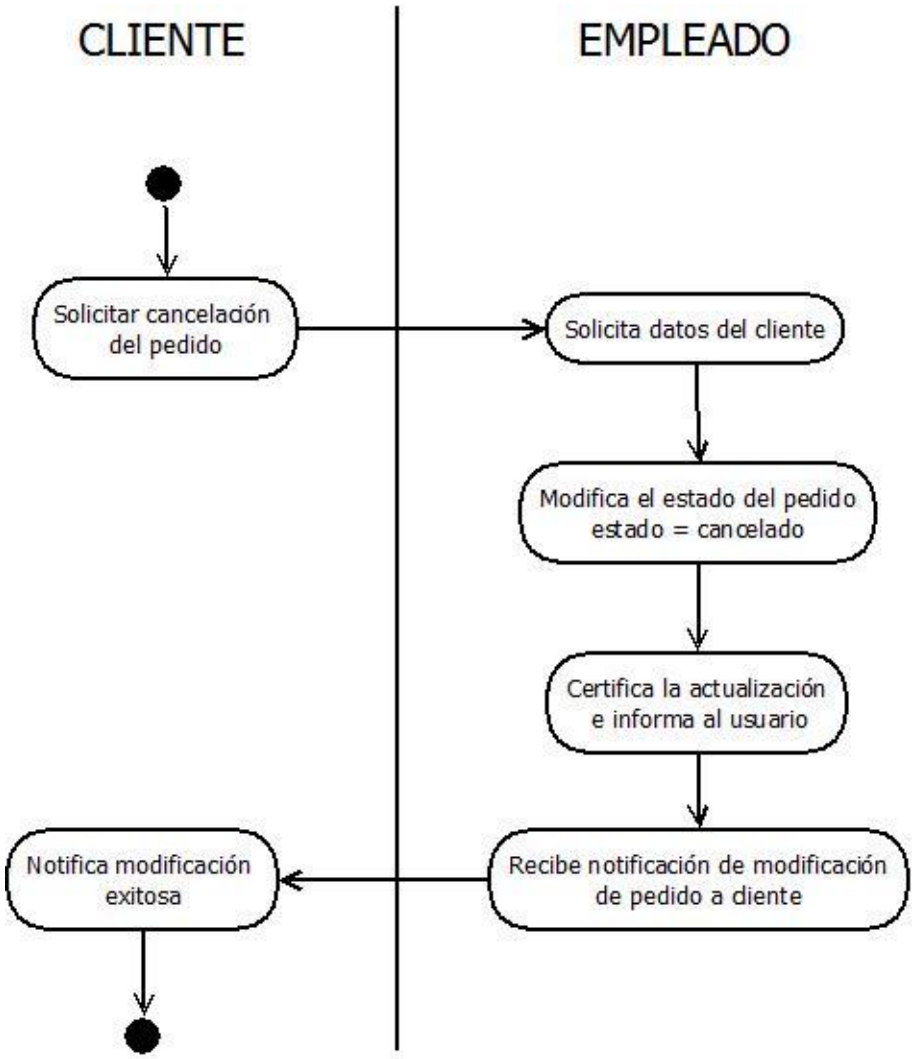
Figura 24. Diagrama de actividad para caso de uso "Cambiar estado de pedido a surtido".



4.1.5 DIAGRAMA DE ACTIVIDAD PARA CASO DE USO “CAMBIAR ESTADO DE PEDIDO A CANCELADO”

En la Figura 25 se muestra el diagrama de actividad para el caso de uso “Cambiar estado de pedido a cancelado”.

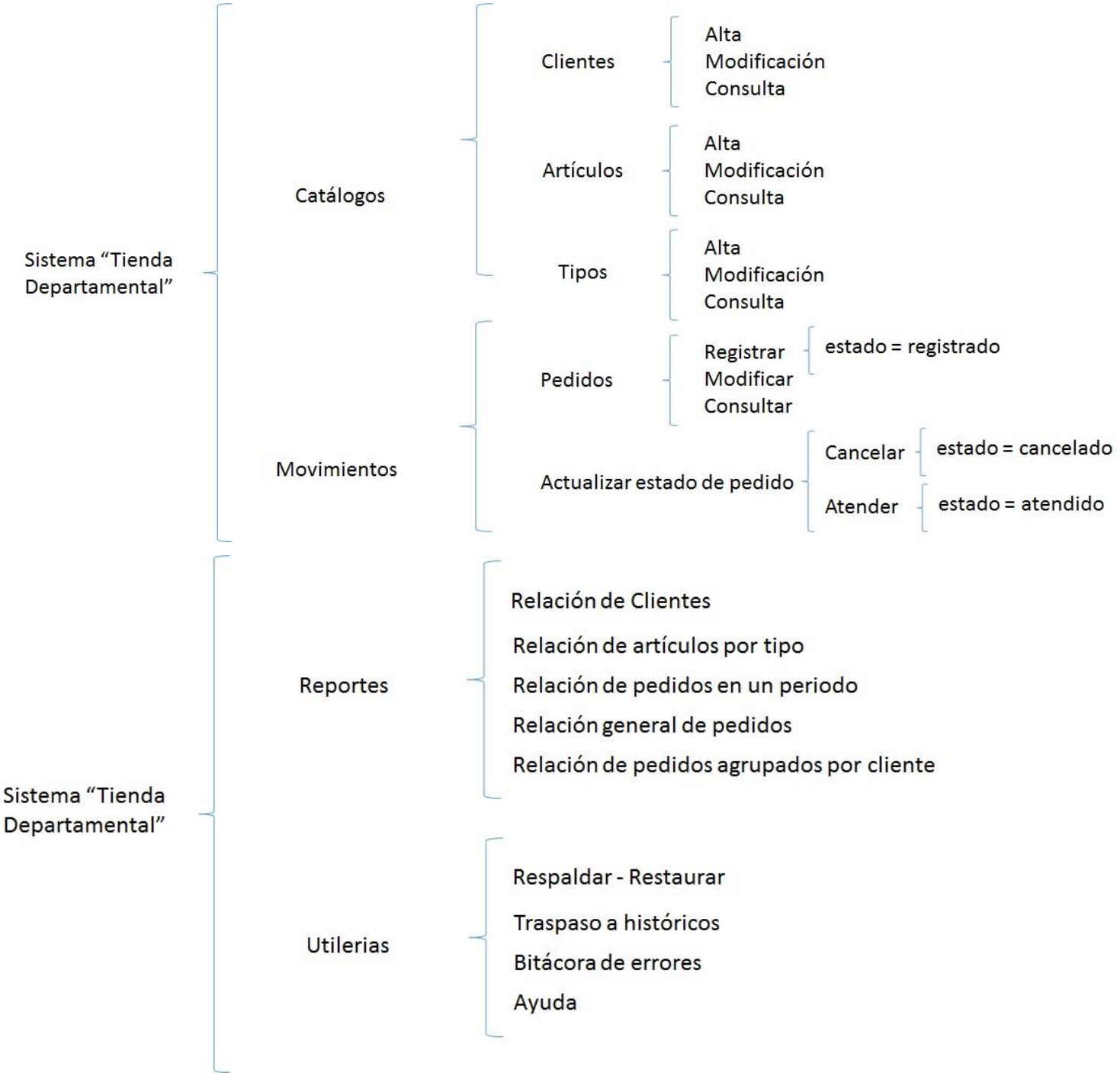
Figura 25. Diagrama de actividad para caso de uso "Cambiar estado de pedido a cancelado".



4.1.6 DIAGRAMA DE WARNIER/ORR

En la Figura 26 se muestra el diagrama de Warnier Orr.

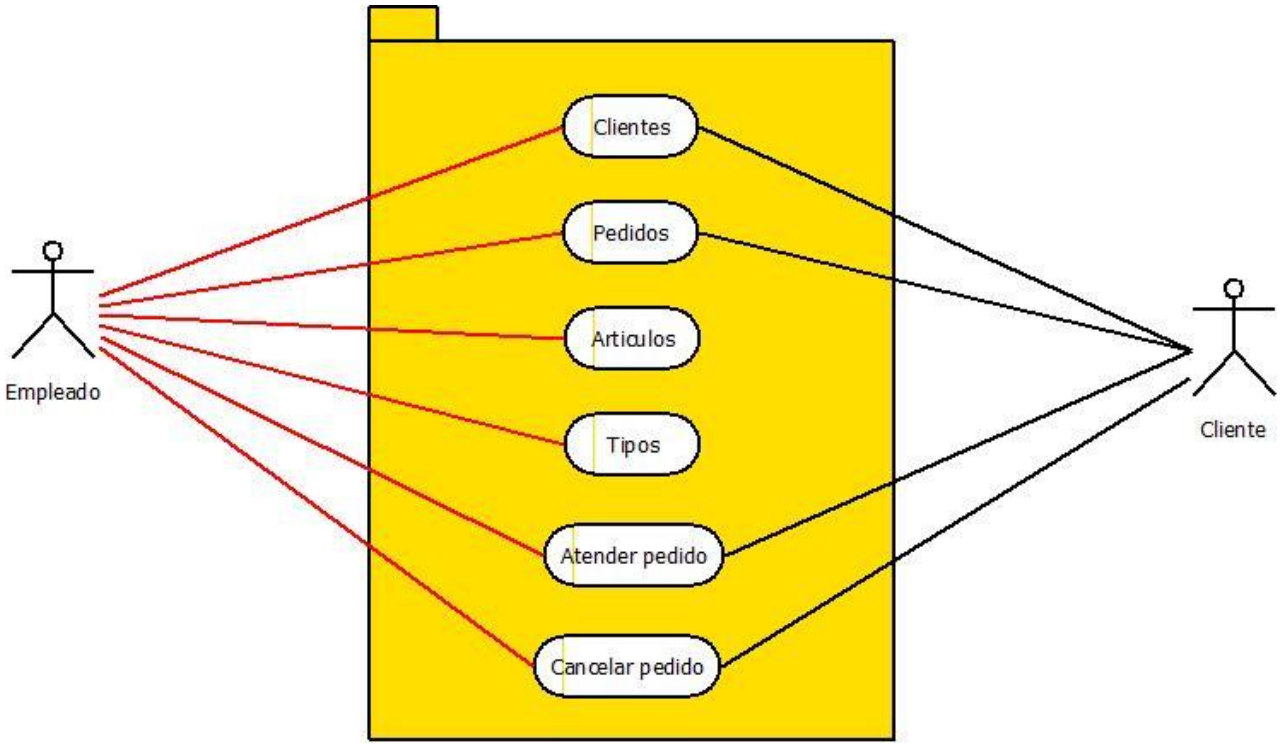
Figura 26. Diagrama de Warnier Orr.



4.1.7 DIAGRAMA DE CASOS DE USO GENERAL

En la Figura 27 se muestra el diagrama de casos de uso general.

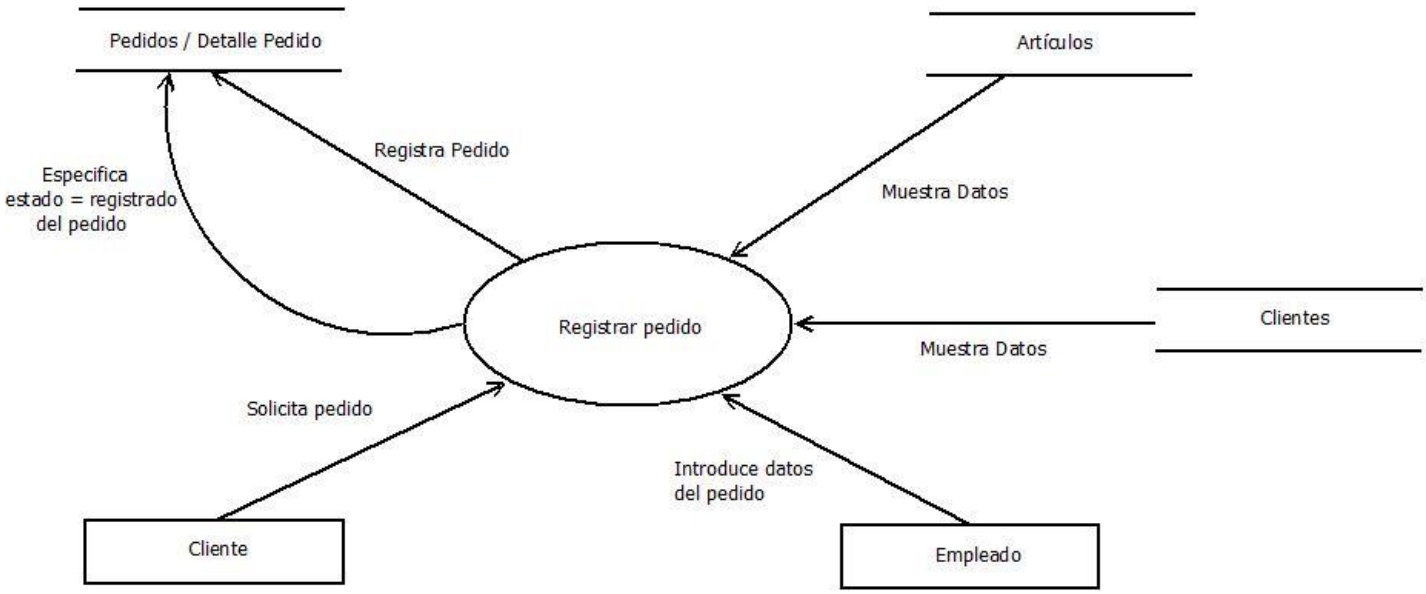
Figura 27. Diagrama de casos de uso general.



4.1.8 DIAGRAMA DE FLUJO DE DATOS PARA “REGISTRAR PEDIDO”

En la Figura 28 se muestra el diagrama flujo de datos para el caso de uso “Registrar Pedido”.

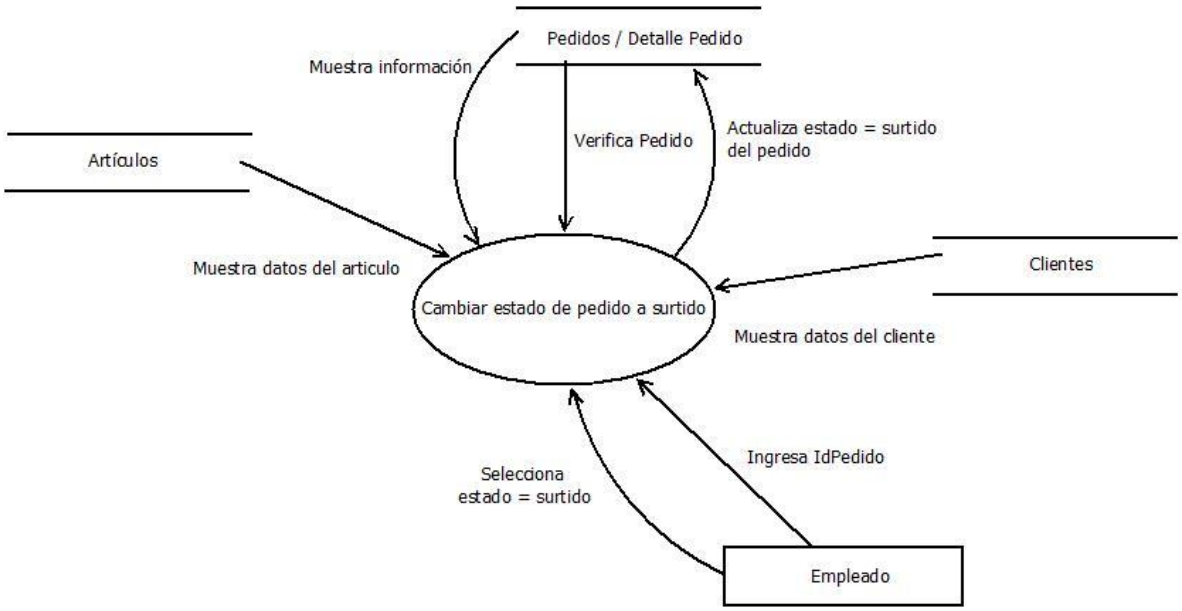
Figura 28. Diagrama de flujo de datos para el caso de uso "Registrar Pedido".



4.1.9 DIAGRAMA DE FLUJO DE DATOS PARA “CAMBIAR ESTADO DE PEDIDO A SURTIDO”

En la Figura 29 se muestra el diagrama de flujo de datos para el caso de uso “Cambiar estado de pedido a surtido”.

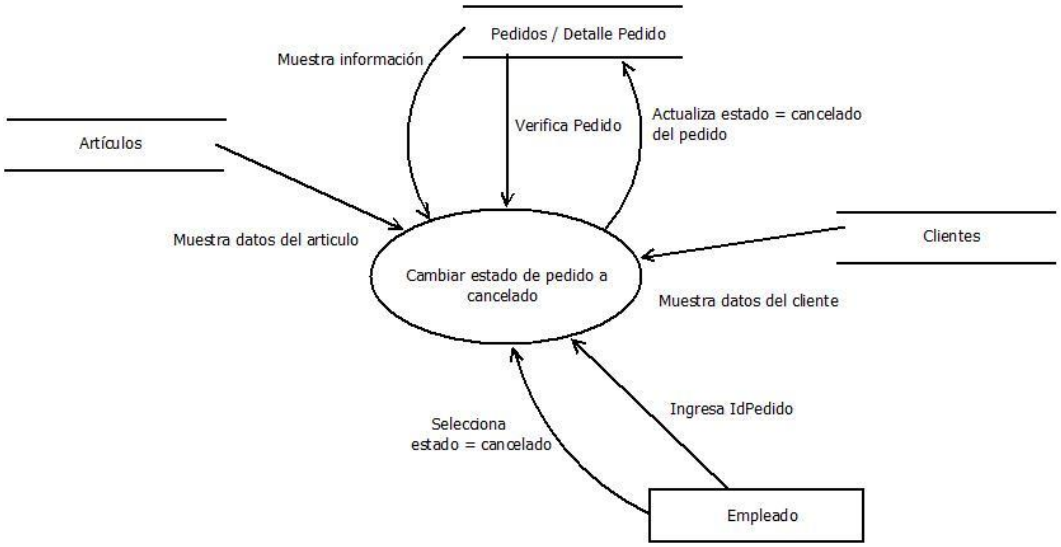
Figura 29. Diagrama de flujo de datos para el caso de uso "Cambiar estado de pedido a surtido".



4.1.10 DIAGRAMA DE FLUJO DE DATOS PARA “CAMBIAR ESTADO DE PEDIDO A CANCELADO”

En la Figura 30 se muestra el diagrama de flujo de datos para el caso de uso “Cambiar estado de pedido a cancelado”.

Figura 30. Diagrama de flujo de datos para el caso de uso "Cambiar estado de pedido a cancelado".

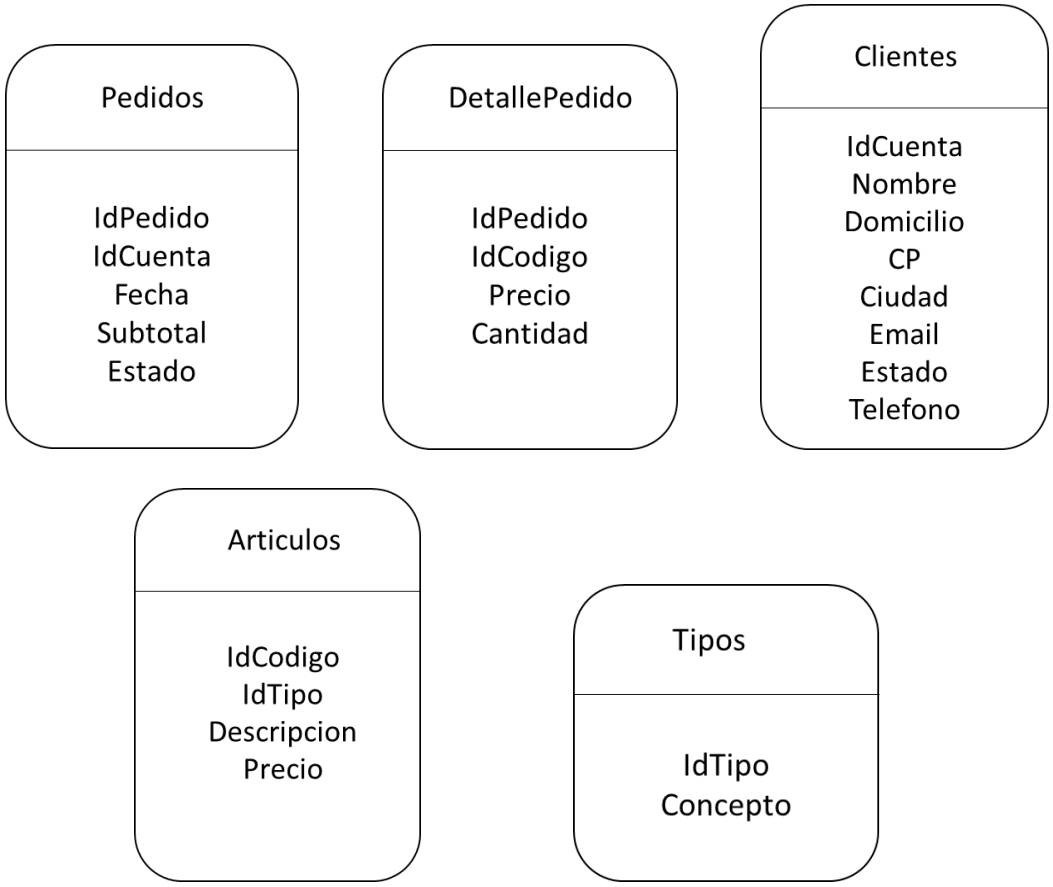


4.2 DISEÑO

4.2.1 TABLAS

A continuación se muestran las tablas que componen la base de datos en SQL Server. Observar la Figura 31.

Figura 31. Tablas o entidades que componen la base de datos.



4.2.2 DICCIONARIO DE DATOS Y NORMALIZACIÓN

Entidad: Articulos

Primera Forma Normal: La entidad **Articulos** cumple con la 1FN ya que sus atributos (IdCodigo,IdTipo, Descripcion y Precio) contienen solo valores atómicos, no existen grupos repetidos y cuenta con un campo llave (IdCodigo).

Segunda Forma Normal: la entidad **Articulos** cumple con la **2FN** ya que sus atributos no primos (Descripcion y Precio) dependen de manera completa y no parcial del campo llave (IdCodigo).

Tercera Forma Normal: la entidad **Articulos** cumple con la **3FN** ya que sus atributos no primos (Descripcion y Precio) dependen directamente y no de manera transitiva del campo llave (IdCodigo).

Objetivo: Almacenar los datos de los artículos.

Llave primaria: IdCodigo

Atributos: 4

Entidad con que se relaciona: Tipos (IdTipo)

En la Tabla 35 se muestra el diccionario de datos de la tabla Artículos.

Tabla 35. Diccionario de datos de la tabla Artículos.

No.	Nombre	Tipo	Long	Descripción	Dominio
1	IdCodigo	int		Guarda la clave de identificación del artículo.	Conjunto de dígitos válidos unidos para formar la clave del artículo.
2	IdTipo	int		Guarda la clave de identificación del tipo del artículo.	Conjunto de dígitos válidos unidos para formar la clave del tipo.
3	Descripcion	texto	50	Guarda la descripción del artículo (entiéndase por descripción, como el nombre del artículo).	Conjunto de caracteres que forman la descripción del artículo.
4	Precio	money		Guarda el precio del artículo.	Conjunto de dígitos válidos unidos para formar el precio del artículo.

Entidad: Tipos

Primera Forma Normal: La entidad **Tipos** cumple con la 1FN ya que sus atributos (IdTipo y Concepto) contienen solo valores atómicos, no existen grupos repetidos y cuenta con un campo llave (IdTipo).

Segunda Forma Normal: la entidad **Tipos** cumple con la **2FN** ya que sus atributos no primos (Concepto) dependen de manera completa y no parcial del campo llave (IdTipo).

Tercera Forma Normal: la entidad **Tipos** cumple con la **3FN** ya que sus atributos no primos (Concepto) dependen directamente y no de manera transitiva del campo llave (IdTipo).

Objetivo: Almacenar descripción los tipos de artículos.

Llave primaria: IdTipo.

Atributos: 2

En la Tabla 36 se muestra el diccionario de datos de la tabla Tipos.

Tabla 36. Diccionario de datos de la tabla Tipos.

No.	Nombre	Tipo	Long	Descripción	Dominio
1	IdTipo	int		Guarda la clave de identificación del artículo.	Conjunto de dígitos válidos unidos para formar la clave del artículo.
2	Concepto	int		Guarda la clave de identificación del tipo del artículo.	Conjunto de dígitos válidos unidos para formar la clave del tipo.

Entidad: Pedidos

Primera Forma Normal: La entidad **Pedidos** cumple con la 1FN ya que sus atributos (IdPedido, IdCuenta, Fecha, Subtotal y Estado) contienen solo valores atómicos, no existen grupos repetidos y cuenta con un campo llave (IdPedido).

Segunda Forma Normal: la entidad **Pedidos** cumple con la **2FN** ya que sus atributos no primos (Fecha, Subtotal y Estado) dependen de manera completa y no parcial del campo llave (IdPedido).

Tercera Forma Normal: la entidad **Pedidos** cumple con la **3FN** ya que sus atributos no primos (Fecha, Subtotal y Estado) dependen directamente y no de manera transitiva del campo llave (IdPedido).

Objetivo: Almacenar los pedidos procesados.

Llave primaria: IdPedido

Llave secundaria: IdCuenta

Atributos: 5

Entidad con que se relaciona: Clientes (IdCuenta).

En la Tabla 37 se muestra el diccionario de datos de la tabla Pedidos.

Tabla 37. Diccionario de datos de la tabla Pedidos.

No.	Nombre	Tipo	Long	Descripción	Dominio
1	IdPedido	int		Guarda la clave de identificación del pedido.	Conjunto de dígitos válidos unidos para formar la clave del pedido.
2	IdCuenta	texto	5	Guarda la clave de identificación de la cuenta del cliente.	Conjunto de dígitos válidos unidos para formar la clave la cuenta del cliente.
3	Fecha	date	50	Guarda la fecha en que fue realizado el pedido.	Conjunto de caracteres que guardan la fecha en que se realizó el pedido.

4	Subtotal	money		Guarda el subtotal referente al pedido.	Conjunto de dígitos válidos unidos para formar el subtotal del pedido.
5	Estado	texto	10	Guarda el estado del pedido.	Conjunto de caracteres que guardan el estado del pedido.

Entidad: DetallePedido

Primera Forma Normal: La entidad **DetallePedido** cumple con la 1FN ya que sus atributos (IdPedido, IdCodigo, Precio, Cantidad) contienen solo valores atómicos, no existen grupos repetidos y cuenta con un campo llave (IdPedido).

Segunda Forma Normal: la entidad **DetallePedido** cumple con la **2FN** ya que sus atributos no primos (Precio, Cantidad) dependen de manera completa y no parcial del campo llave (IdPedido).

Tercera Forma Normal: la entidad **DetallePedido** cumple con la **3FN** ya que sus atributos no primos (Precio, Cantidad) dependen directamente y no de manera transitiva del campo llave (IdPedido).

Objetivo: Almacena los datos referentes al detalle del pedido.

Llave primaria: IdPedido

Llave secundaria: IdCodigo

Atributos: 4

En la Tabla 38 se muestra el diccionario de datos de la tabla DetallePedido.

Tabla 38. Diccionario de datos de la tabla DetallePedido.

No.	Nombre	Tipo	Long	Descripción	Dominio
1	IdPedido	int		Guarda la clave de identificación del pedido.	Conjunto de dígitos válidos unidos para formar la clave del pedido.
2	IdCodigo	texto	10	Guarda la clave de identificación del código del artículo.	Conjunto de caracteres unidos para formar la clave del artículo.
3	Precio	Money		Guarda el precio del artículo.	Conjunto de dígitos válidos unidos para formar el precio del artículo.
4	Cantidad	int		Guarda la cantidad de artículos requeridos.	Conjunto de dígitos válidos unidos para formar la cantidad de artículos requeridos.

Entidad: Clientes

Primera Forma Normal: La entidad **Clientes** cumple con la 1FN ya que sus atributos (IdCuenta, Nombre, Domicilio, CP, Ciudad, Email, Estado y Telefono) contienen solo valores atómicos, no existen grupos repetidos y cuenta con un campo llave (IdCuenta).

Segunda Forma Normal: la entidad **Clientes** cumple con la **2FN** ya que sus atributos no primos (Nombre, Domicilio, CP, Ciudad, Email, Estado y Telefono) dependen de manera completa y no parcial del campo llave (IdCuenta).

Tercera Forma Normal: la entidad **Clientes** cumple con la **3FN** ya que sus atributos no primos (Nombre, Domicilio, CP, Ciudad, Email, Estado y Telefono) dependen directamente y no de manera transitiva del campo llave (IdCuenta).

Objetivo: Almacena los datos de los clientes

Llave primaria: IdCuenta

Atributos: 8

En la Tabla 39 se muestra el diccionario de datos de la tabla Clientes.

Tabla 39. Diccionario de datos de la tabla Clientes.

No.	Nombre	Tipo	Long	Descripción	Dominio
1	IdCuenta	texto	5	Guarda la clave de identificación del cliente.	Conjunto de caracteres unidos para formar la clave del cliente.
2	Nombre	texto	30	Guarda el nombre del cliente.	Conjunto de caracteres unidos para formar el nombre del cliente.
3	Domicilio	texto	50	Guarda el domicilio del cliente.	Conjunto de caracteres que guardan el domicilio del cliente.
4	CP	Texto	5	Guarda el código postal del cliente.	Conjunto de caracteres que guardan el código postal del cliente.
5	Ciudad	texto	40	Guarda la ciudad en la que reside el cliente.	Conjunto de caracteres que guardan la ciudad en donde reside el cliente.
6	Email	texto	30	Guarda el e-mail del cliente.	Conjunto de caracteres que guardan el e-mail del cliente.

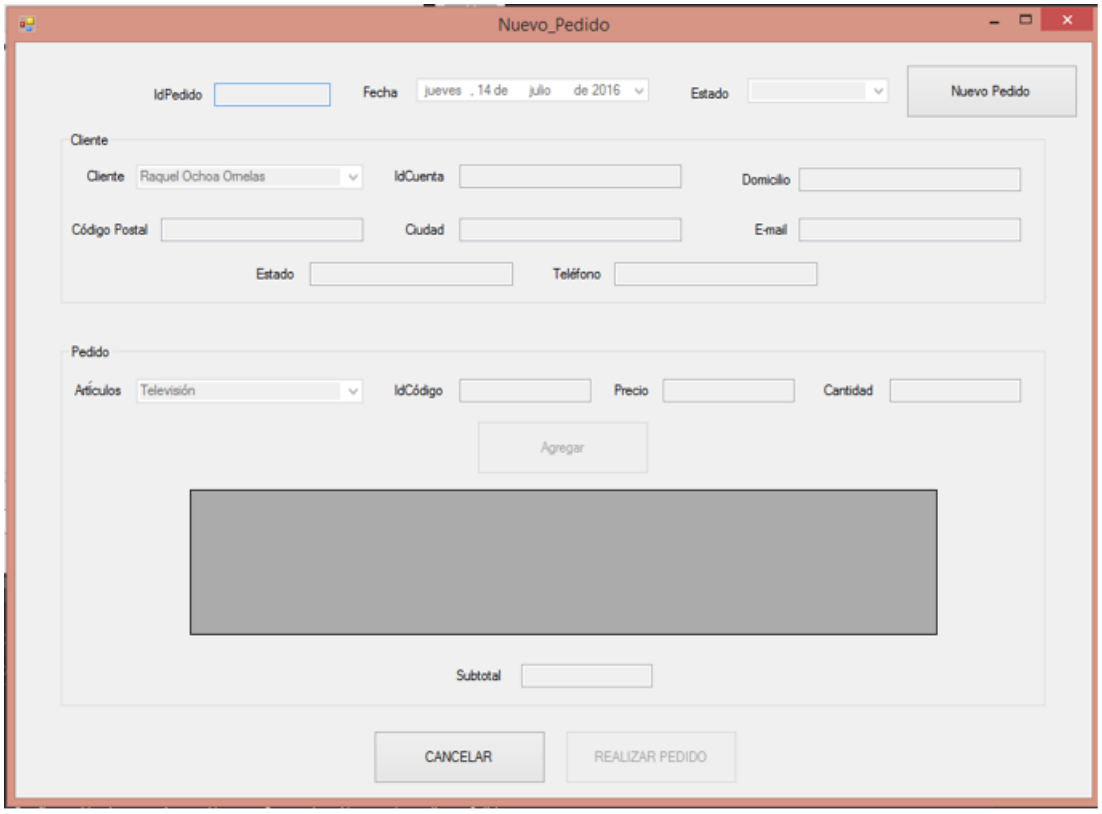
7	Estado	texto	30	Guarda el estado en donde reside el cliente.	Conjunto de caracteres que guardan el estado en donde reside el cliente.
8	Telefono	texto	10	Guarda el teléfono del cliente.	Conjunto de caracteres que guardan el teléfono del cliente.

4.2.3 DISEÑO DE PANTALLAS

Diseño de pantalla para "Registrar Pedido", utilizando Transacciones

En la Figura 32 se muestra el diseño de la pantalla que se utiliza para "Registrar Pedido" en la que describe el IdPedido, Fecha y Estado, separados del grupo de componentes del Cliente y Pedido, así como el botón de "Cancelar" y "Realizar Pedido".

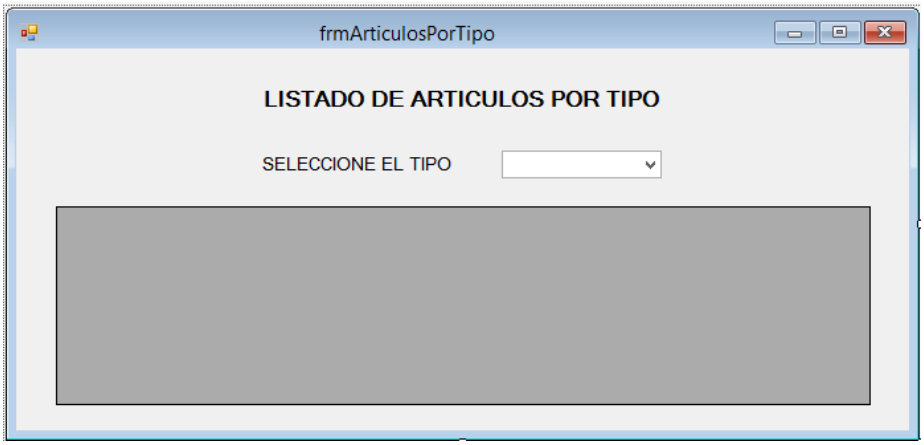
Figura 32. Diseño de pantalla para "Registrar Pedido".



Diseño de pantalla para “Consultar Artículos por Tipo”, utilizando LINQ to SQL

En la Figura 33 se muestra el diseño de la pantalla que se utiliza para “Consultar Artículos por Tipo” en la que se visualiza un ComboBox para seleccionar el tipo de artículos y un DataGridView para la visualización de los mismos.

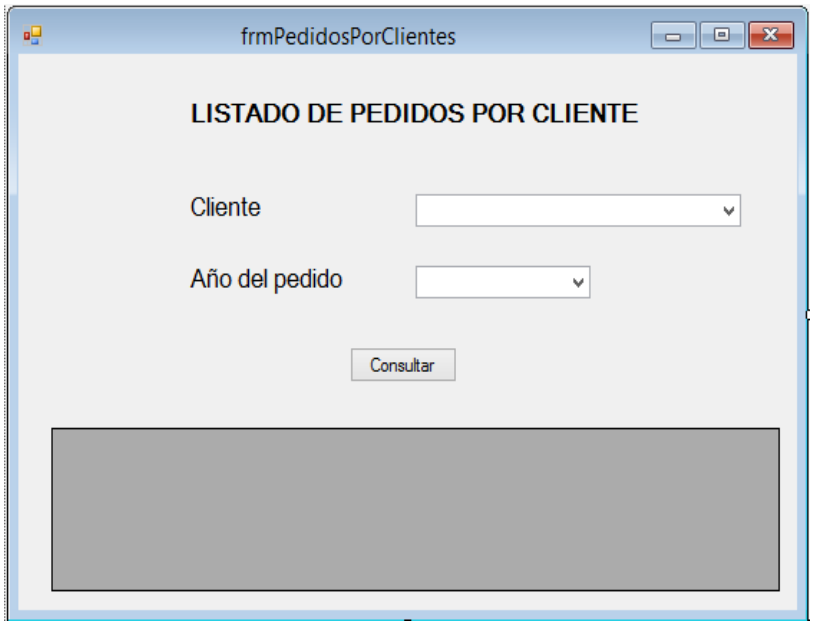
Figura 33. Diseño de pantalla para "Consultar Artículos por Tipo".



Diseño de pantalla para “Consultar Pedidos de Clientes por año”, utilizando LINQ to SQL

En la Figura 34 se muestra el diseño de la pantalla que se utiliza para “Consultar Pedidos de Clientes por año” en la que se visualiza un ComboBox para seleccionar el cliente y otro para seleccionar el año del pedido, así como un DataGridView para la visualización de los pedidos.

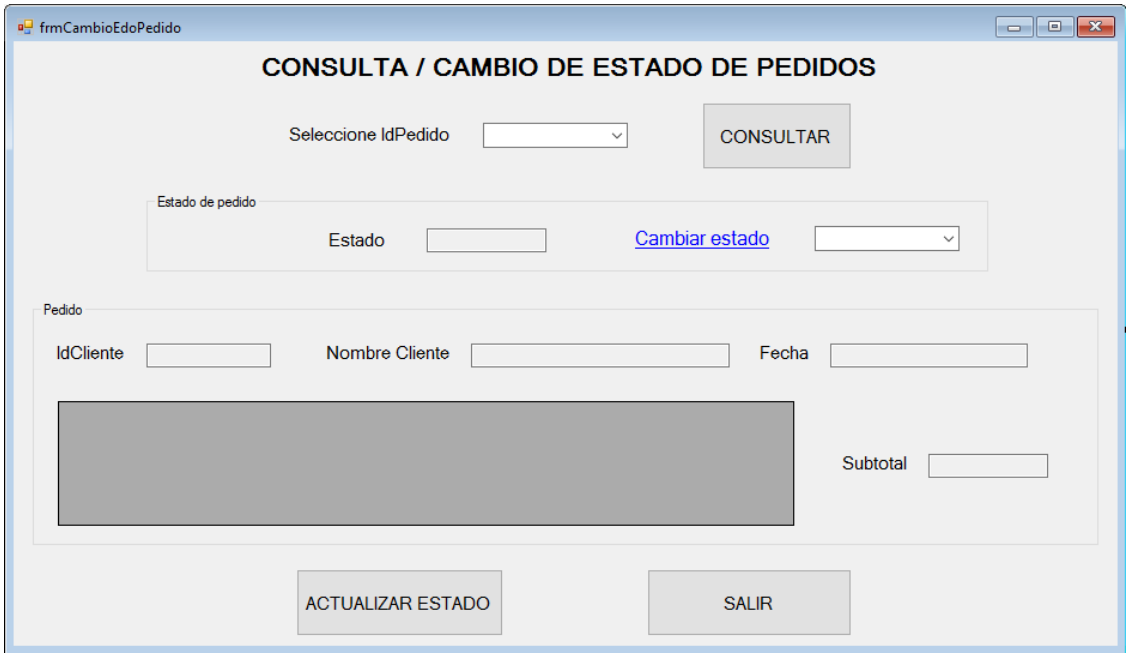
Figura 34. Diseño de pantalla para "Consultar Pedidos de Clientes por año".



Diseño de pantalla para "Consultar y Cambiar estado de Pedido", utilizando Programación en N-capas

En la Figura 35 se muestra el diseño de la pantalla que se utiliza para "Consultar y Cambiar estado de Pedido" en la que se visualiza un ComboBox para seleccionar el IdPedido y otro para seleccionar el nuevo estado del pedido, así como un DataGridView para la visualización del detalle de pedido.

Figura 35. Diseño de pantalla para "Consultar y Cambiar de estado de Pedido".



4.3 TÉCNICAS DE PROGRAMACIÓN

4.3.1 TRANSACCIONES

Una transacción consiste en un comando único o en un grupo de comandos que se ejecutan como un paquete. Las transacciones permiten combinar varias operaciones en una sola unidad de trabajo (Torres Remon, 2012).

Si en un punto de la transacción se produjera un error, todas las actualizaciones podrían revertirse y devolverse al estado que tenían antes de la transacción (Torres Remon, 2012).

Una transacción debe ajustarse a las propiedades del ACID (Atomicidad, Coherencia, Aislamiento y Durabilidad) para poder garantizar la coherencia de datos. La mayoría de los sistemas de bases de datos relacionales como Microsoft SQL Server, admiten transacciones, al proporcionar funciones de bloqueo, registro y administración de transacciones cada vez que una aplicación cliente realiza una operación de actualización, inserción o eliminación.

4.3.2 TABLE ADAPTER

Los TableAdapter proporcionan comunicación entre la aplicación y una base de datos mediante la ejecución de instrucciones SQL y procedimientos almacenados contra una base de datos además de la funcionalidad estándar de un DataAdapter, los TableAdapter proporcionan consultas adicionales que comparten un esquema común con el DataTable con tipo asociado. El TableAdapter carga los datos revueltos en su tabla de datos asociada de la aplicación o devuelve nuevas tablas de datos ya rellenadas (Torres Remon, 2012).

Torres Remon (2012) considera que se puede crear un DataAdapter a partir de 3 puntos:

- Crear con asistente para la configuración de TableAdapter:
  El asistente para la configuración de TableAdapter crea un TableAdapter único basado en la información que se proporciona al asistente.
  Hay que abrir un conjunto de datos en el Diseñador de Dataset.
  Arrastrar un TableAdapter desde la ficha DataSet del Cuadro de herramientas a la superficie de diseño.
  Al completar el asistente, se agregará una tabla de datos y un TableAdapter al conjunto de datos.
- Crear con asistente para la configuración de orígenes de datos:
  El asistente para la configuración de orígenes de datos creará un TableAdapter para cada objeto de la base de datos seleccionado durante el funcionamiento del asistente. Una vez finalizado el asistente para la configuración de orígenes de datos, se puede ver los TableAdapters creados abriendo el conjunto de datos del Diseñador de Dataset.
  Seleccionar Agregar nuevo origen de datos en la ventana de Orígenes de datos. Elegir la opción Mostrar orígenes de datos del menú Datos para abrir la ventana de Orígenes de datos.
  Al finalizar el asistente, seleccionar el tipo de origen de datos de una base de datos o Servicio Web.
- Crear a partir de objetos de base de datos del Explorador de servidores:
  Se crea un solo TableAdapter por cada objeto de bases de datos que se arrastre hasta el Diseñador de DataSet.

Abrir un conjunto de datos en el Diseñador de Dataset.

Arrastrar un objeto de base de datos desde una conexión de datos en el Explorador de servidores a la superficie del Diseñador de Dataset.

Se agregarán una tabla de datos y un TableAdapter al conjunto de datos.

4.3.3 LINQ TO SQL

LINQ es un Lenguaje-Integrado de consultas LINQ to ADO.NET que permite consultar cualquier objeto enumerable en ADO.NET mediante el uso del modelo de programación de Lenguage-Integrated Query (LINQ) (Torres Remon, 2012).

LINQ to SQL proporciona una infraestructura en tiempo de ejecución para administrar los datos relacionales como objetos. En LINQ to SQL, el modelo de datos de una base de datos relacional se asigna a un modelo de objetos expresado en el lenguaje de programación del desarrollador. Al ejecutar la aplicación, LINQ to SQL convierte las consultas integradas en el lenguaje del modelo de objetos a SQL y las envía a la base de datos para su ejecución. Cuando la base de datos devuelve los resultados, LINQ to SQL los vuelve a convertir en objetos que se pueden manipular (Torres Remon, 2012).

LINQ to SQL incluye compatibilidad con los procedimientos almacenados y las funciones definidas por el usuario en la base de datos, así como con la herencia del modelo de objetos (Torres Remon, 2012).

4.3.4 PROGRAMACIÓN EN CAPAS

La construcción de aplicaciones complejas lleva a la unión de clases y objetos en un solo proyecto llamando a esto *Programación en capas* (Torres Remon, 2012).

La programación en capas propone dividir los elementos que compone una aplicación según las funciones que realice, es decir, si se tiene que conectar una base de datos se juntarán

todos los objetos que tengan afinidad y se agruparán en una biblioteca de clase llamada Capa de Datos (Torres Remon, 2012).

Se trata de tres capas como mínimo para poder implementar una aplicación y así decir que una aplicación se encuentra en capas (Torres Remon, 2012).

Torres Remon (2012) propone a continuación cómo se describe algunas de las capas:

- Capa de presentación

  Es la capa que se encarga de modelar los elementos de tal forma que se puede interactuar con el usuario.
- Capa de datos

  Es la capa que se encarga de proveer de información conectada a una base de datos y mantenerla de manera adecuada durante todo el proceso de una aplicación.
- Capa lógica

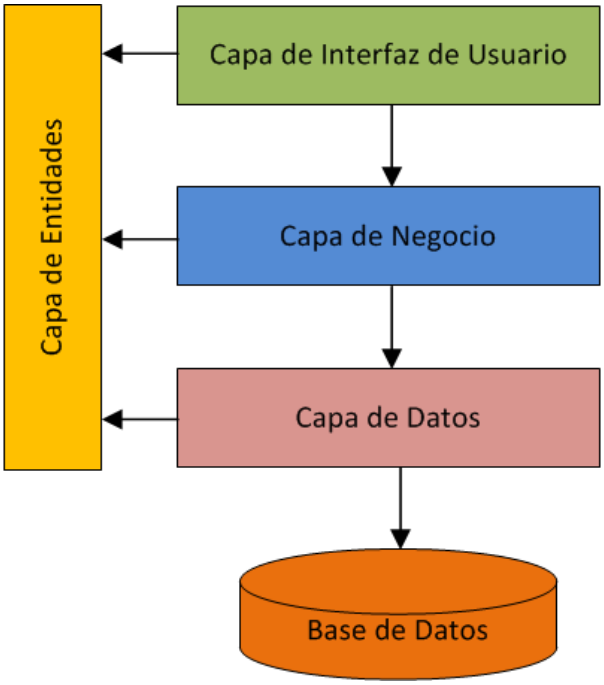
  Es la capa encargada de realizar toda la parte funcionalidad de la aplicación, es decir, servirá de puente entre la capa de presentación y las demás capas.
- Capa entidad

  En esta se expone las entidades que participan directamente en un proceso de la aplicación. Esta capa es propuesta por el Diagrama de Clases de Análisis, en la cual se toma en cuenta que una aplicación proviene de un análisis.

Las entidades son representadas como una entidad de tabla en un motor de base de datos, mientras que la interfaz (o capa de presentación) se podrá realizar en cualquier lenguaje de programación de soporte orientada a objetos como Visual Net o Java (Torres Remon, 2012).

La implementación de n-capas se verá como se muestra en la Figura 36.

Figura 36. Implementación de la programación de N-capas.



Fuente: Quintela Limón (2016).

La capa de presentación interactúa con el usuario como con la capa lógica y ésta a su vez interactúa con la Capa de Datos y la Capa Entidad (Torres Remon, 2012).

Patrón DAO

La capa de presentación solicitará funciones o procedimientos a la capa lógica, ésta capa invocará las funciones de acceso a datos de la capa datos, pasando los valores por la capa entidad, siendo aquí en donde se presenta conflicto sobre quién tendrá la responsabilidad de tener la lógica de los procesos si se necesita tener acceso a muchas entidades, por lo que el patrón DAO permite manejar la conexión al origen de datos para obtener y almacenar información dentro de las mismas, es decir, se podrán realizar consultas, insertar actualizar o eliminar usando una clase llamada DAO (Torres Remon, 2012).

Torres Remon (2012) explica los beneficios de dicho patrón:

- Separa el acceso a datos de la lógica de negocio. Esto lo hace más útil para sistemas medianos o grandes, o que manejen una lógica de negocios compleja.

- Encapsula la fuente de datos, siendo beneficio para sistemas con acceso a múltiples entradas.
- Centraliza todos los accesos a Datos en una capa independiente, en la que dependiendo de cómo se llamé la entidad se nombrará una clase que implementará funciones para dicha entidad, por ejemplo DAOCliente.
- Cuando se trabaja con DAO los datos de encuentran desconectados de la fuente de origen y es allí en donde entra la clase que por medio de objetos obtiene los valores de la entidad tanto para consultar o aplicar cualquier función de mantenimiento sobre ella.
- Cuando se ejecuta una aplicación en n-capas, se apertura la conexión a la base de datos, se ejecuta la instrucción SQL por medio de funciones, si es una operación de lectura, lleva los datos a una estructura de datos de NET y se cierra la conexión.

CAPÍTULO 5. RESULTADOS DE APLICACIONES

5.1 CONFIGURACIÓN DE VISUAL STUDIO COMMUNITY 2015

Para agregar herramientas de SQL Server a Visual Basic Community 2015 es necesario hacer lo siguiente:

Ir a “Panel de Control” – “Programas” - “Programas y características” – “Visual Basic Community 2015” y dar doble clic en el programa, de manera que le aparezca algo similar a la Figura 37 y 38.

Figura 37. Ejecución de Visual Basic Community 2015.

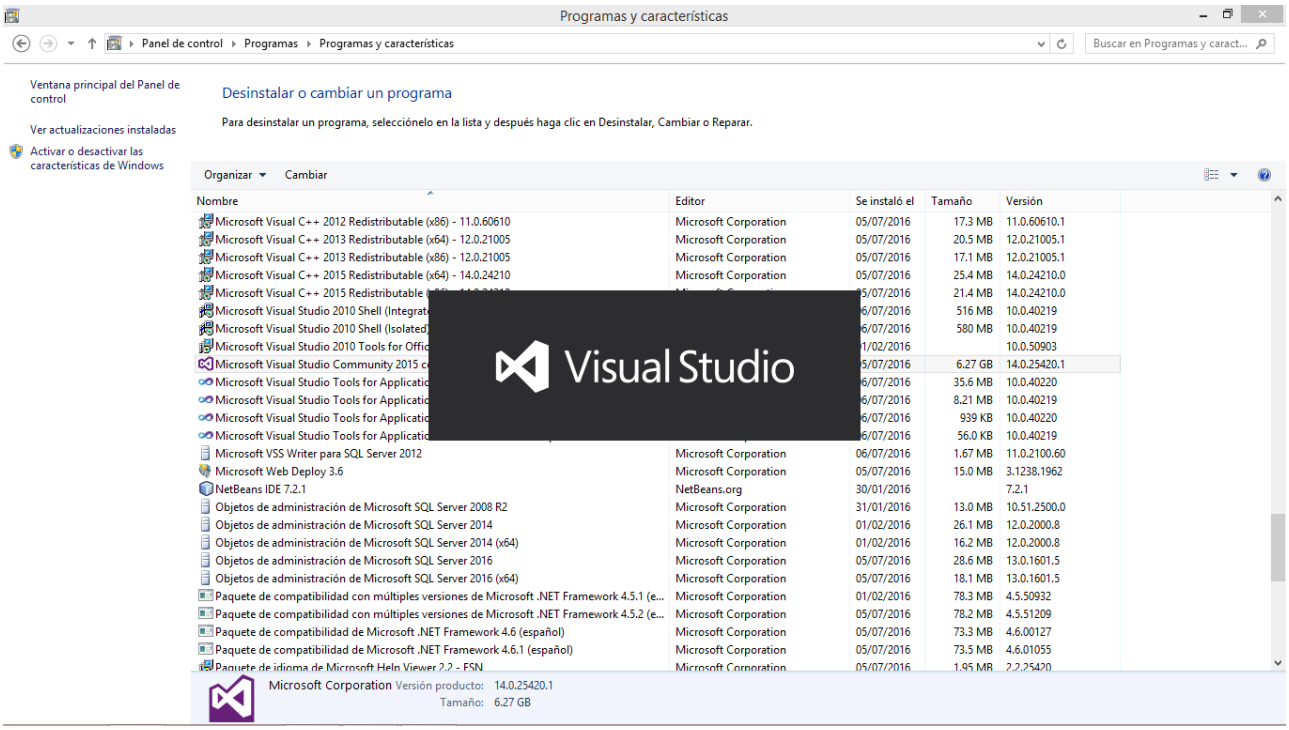
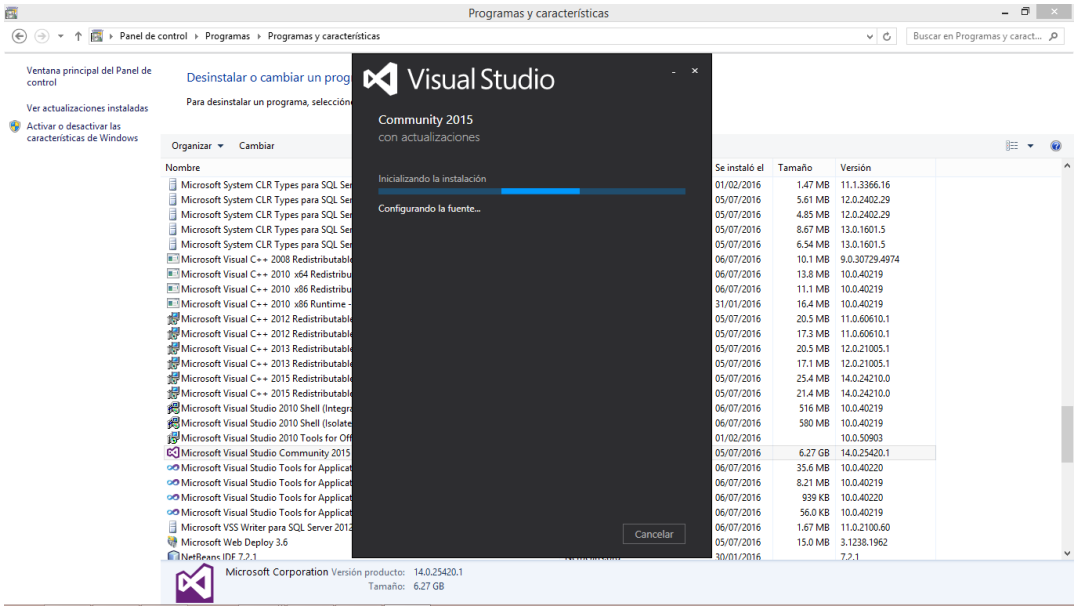


Figura 38. Continuación de la ejecución de Visual Basic Community 2015.



Dar clic en “Modificar” como lo muestra la Figura 39.

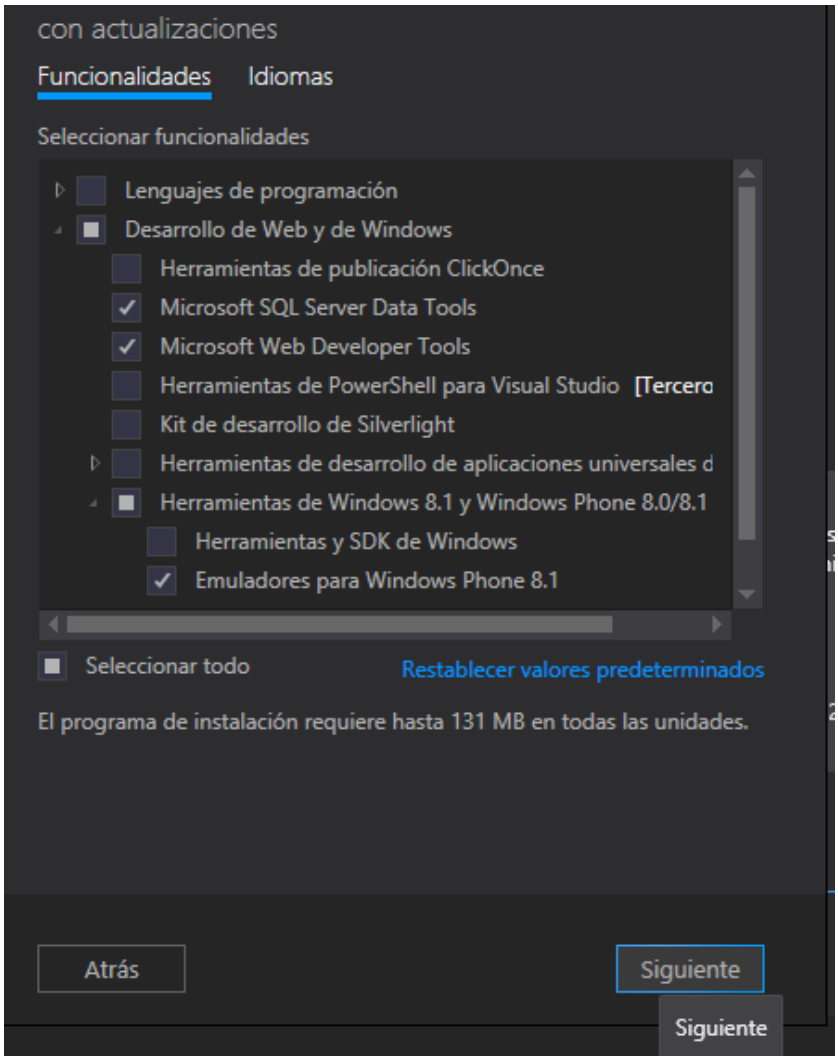
Figura 39. Demostración de opciones en el arranque de configuración de Visual Studio Community 2015.

Marcar la casilla de “Microsoft SQL Server Data Tools” como lo muestra la Figura 40.

Figura 40. Opciones de funcionalidades de Visual Studio Community 2015.

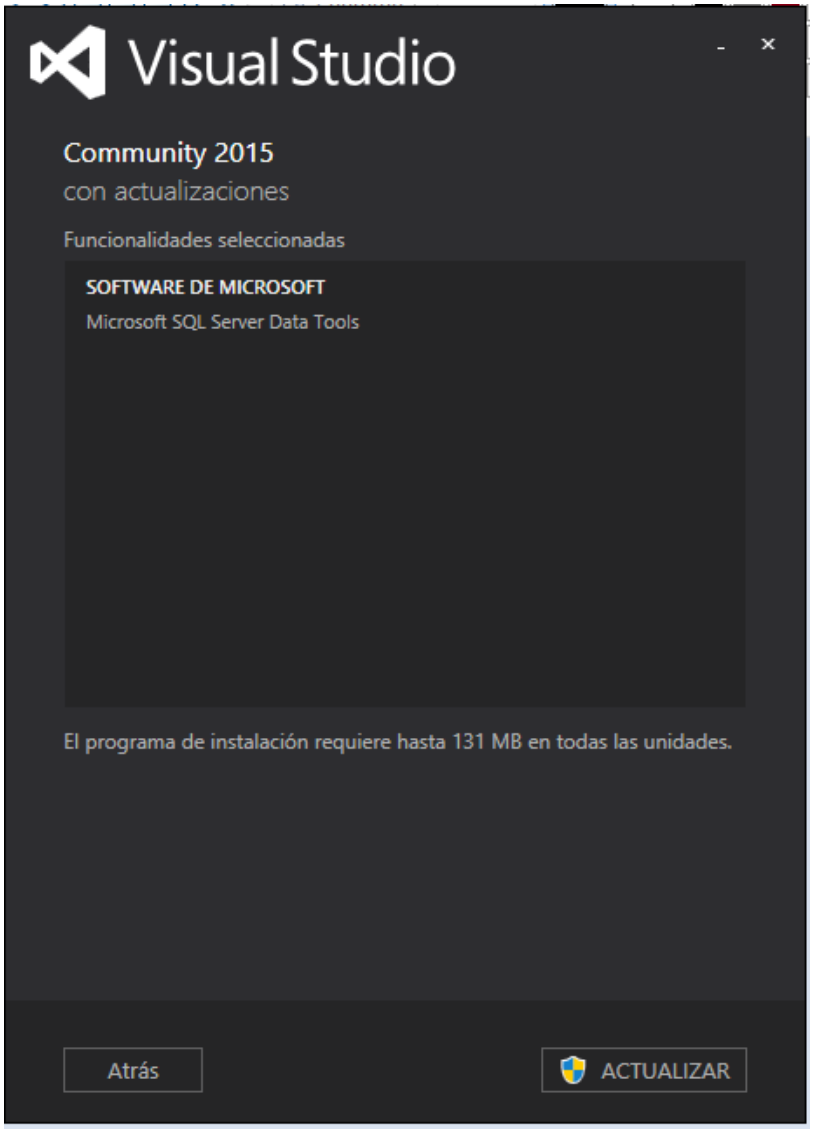
Pulsar clic en “Siguiente” como se muestra en la Figura 41.

Figura 41. Opción de continuidad tras selección de funcionalidades.



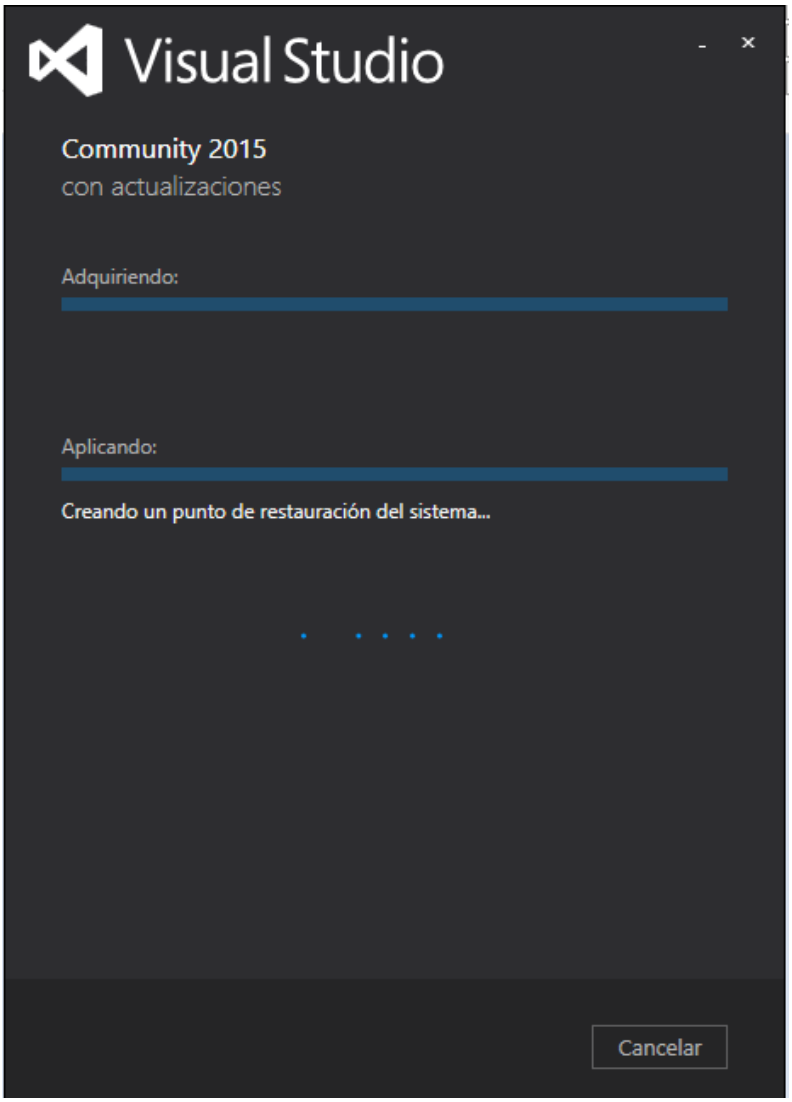
Y dar clic en “Actualizar” para aplicar la funcionalidad antes seleccionada como se muestra en la Figura 42.

Figura 42. Actualización de funcionalidades en Visual Studio Community 2015.



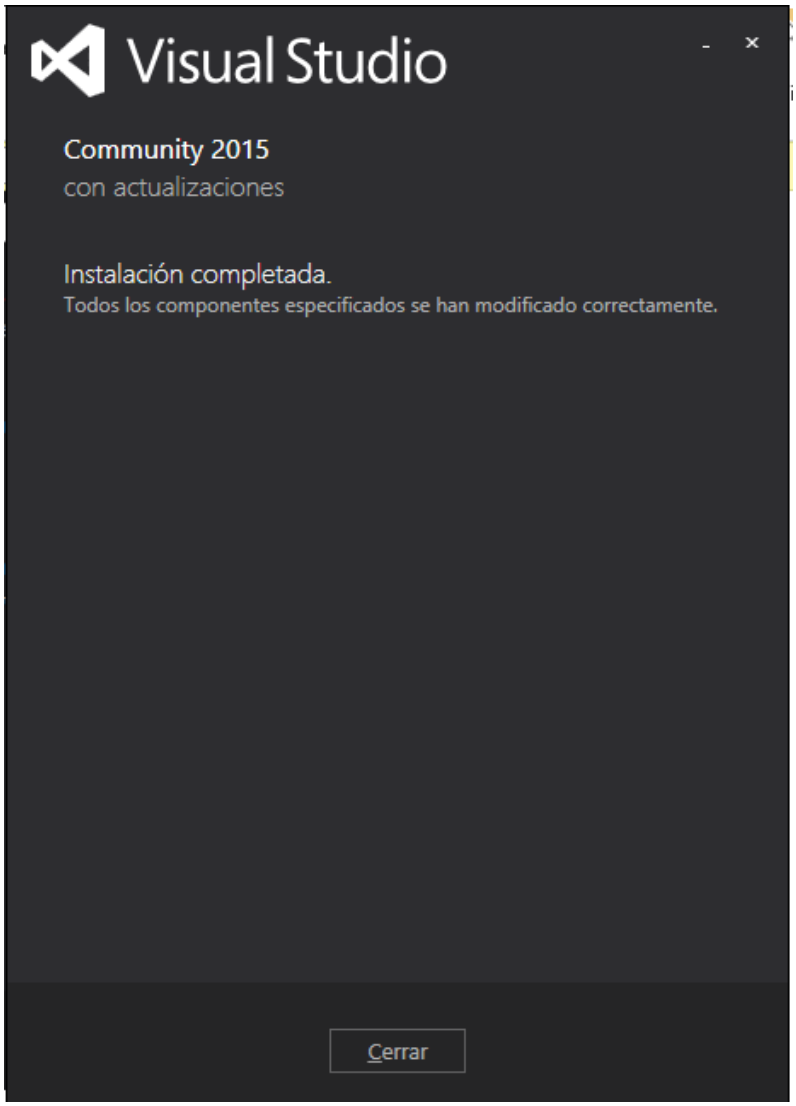
La adecuación tardará algo de tiempo en realizarse por lo que hay que esperar a que se termine el proceso. La pantalla debe mostrar algo similar a la Figura 43.

Figura 43. Proceso de actualización de funciones.



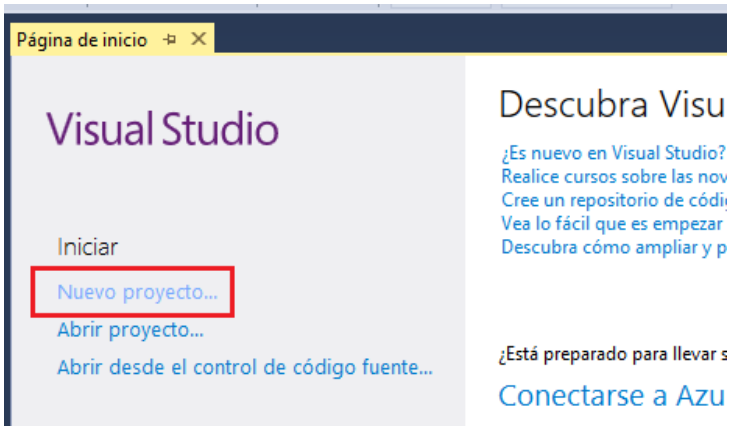
Una vez terminado el proceso de instalación cerrar la ventana. Si la instalación se realizó de manera correcta debe mostrar el mensaje que muestra la Figura 44.

Figura 44. Pantalla de terminación de proceso de actualización de funciones.



Crear un nuevo proyecto en Visual Studio en la página de inicio como se muestra en la Figura 45.

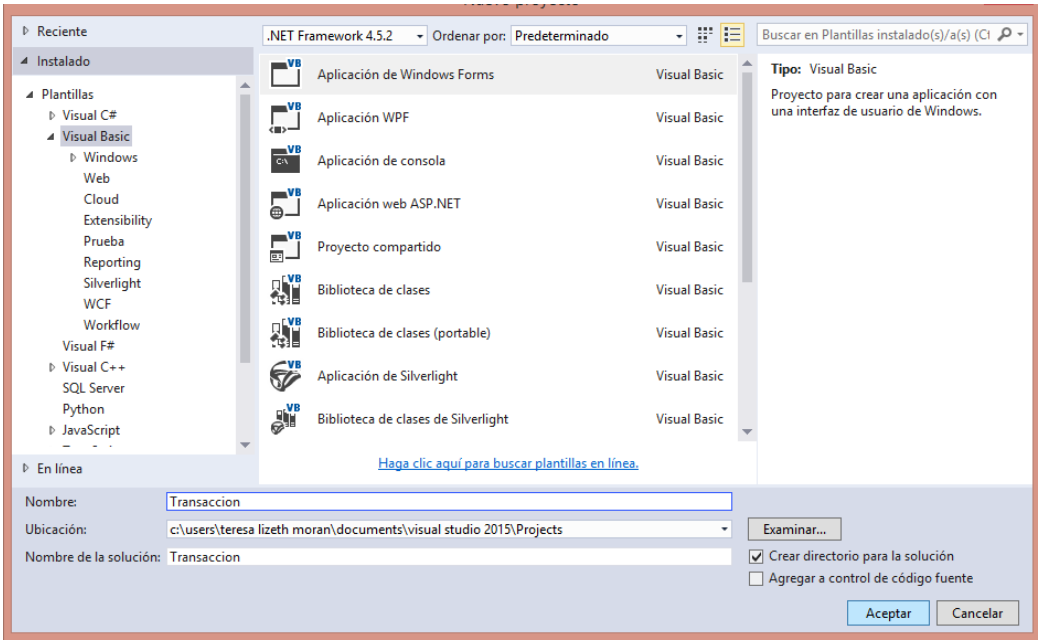
Figura 45. Creación de nuevo proyecto en Visual Studio Community 2015.



5.2 REGISTRO DE NUEVO PEDIDO CON TRANSACCIONES

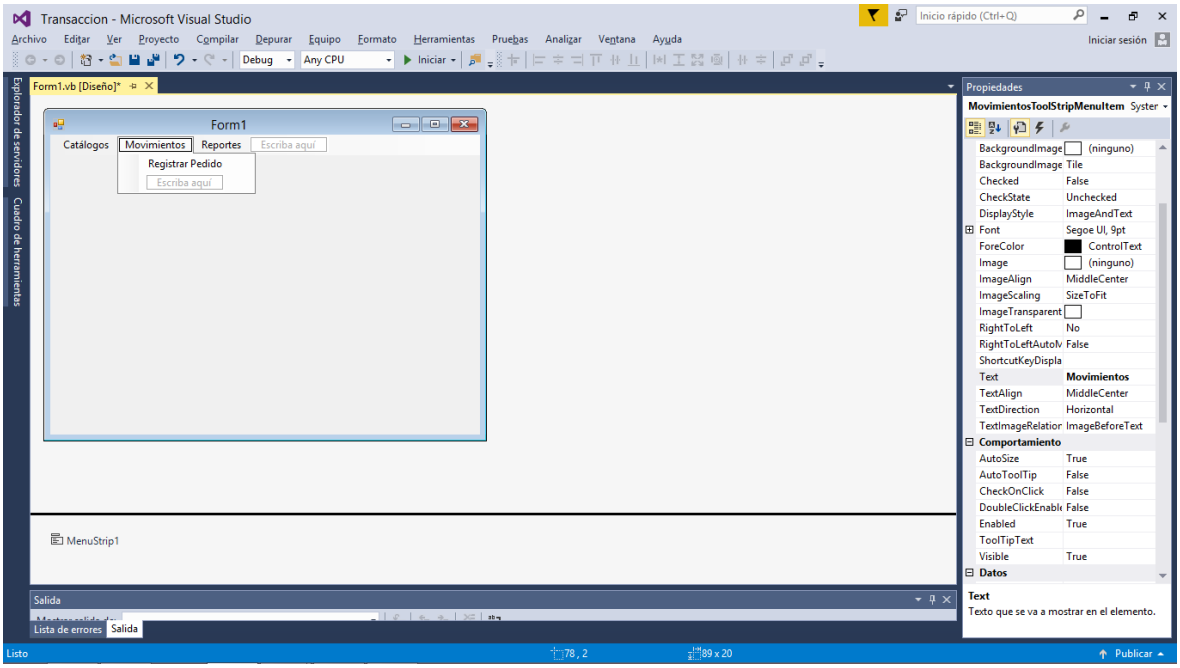
Nombre el nuevo proyecto y de clic en “Aceptar”, la interfaz debe mostrarse como la Figura 46.

Figura 46. Asignación de nombre al nuevo proyecto.



Diseñar la aplicación según las necesidades. La Figura 47 muestra una opción de cómo puede visualizarse la interfaz.

Figura 47. Interfaz principal.



Para realizar el movimiento de “Registrar Pedido” añadir al proyecto un Windows Form para comenzar a diseñar y codificar. Seguir el procedimiento de creación como lo muestran las Figuras 48 y 49.

Figura 48. Creación de nuevo Windows Forms.

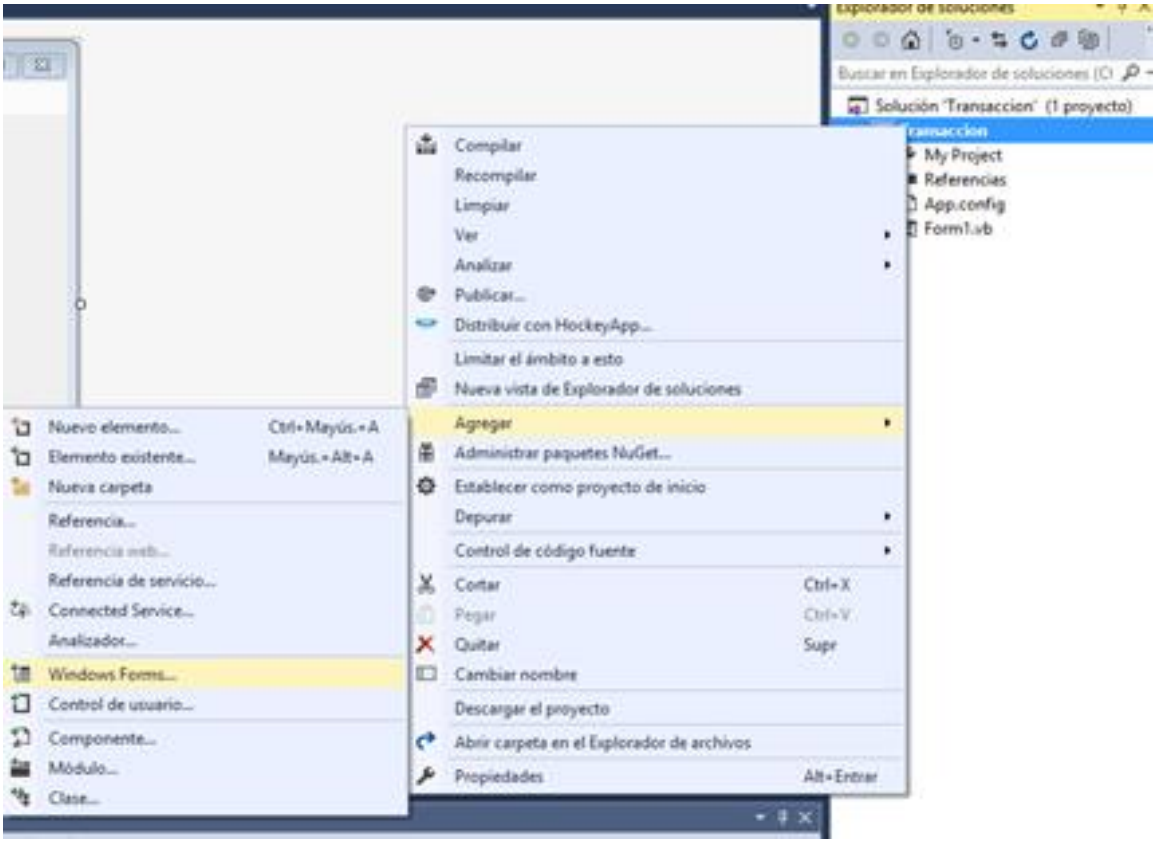
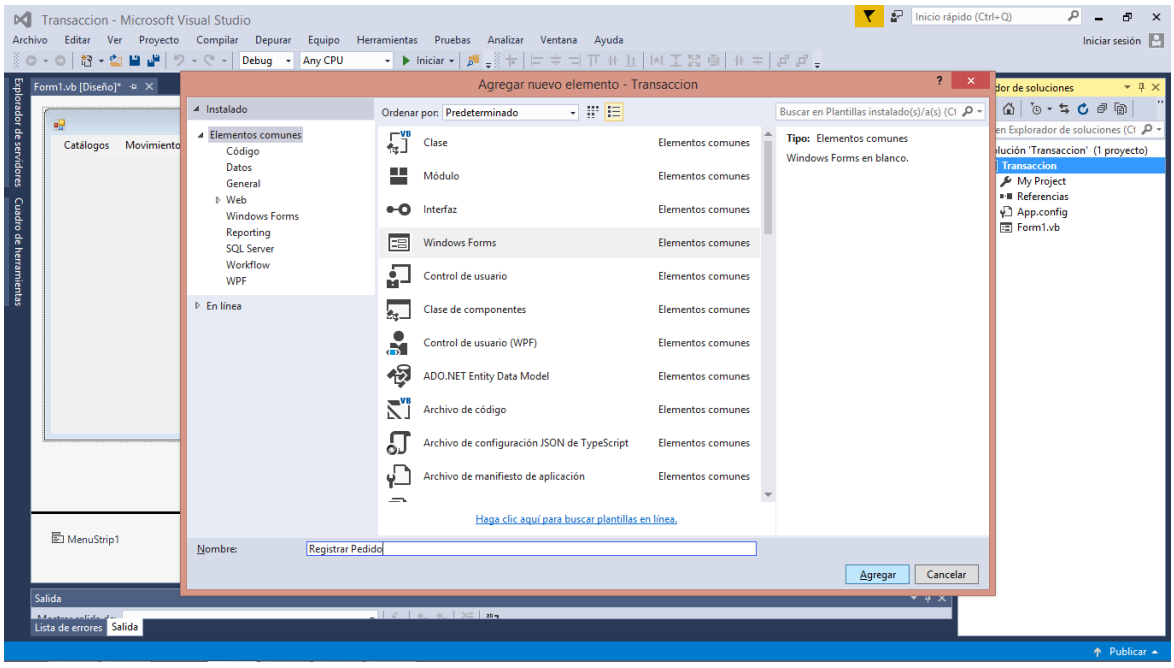


Figura 49. Seguimiento a la creación de nuevo Windows Form.



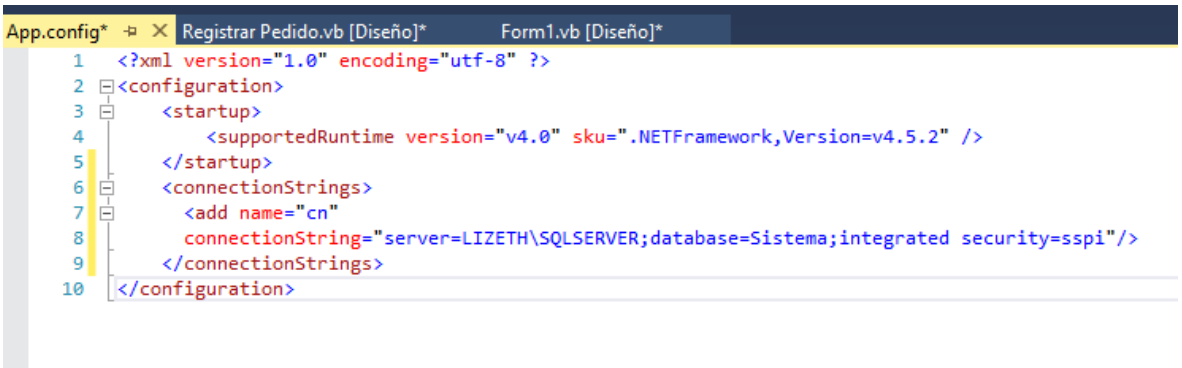
Crear los procedimientos almacenados necesarios en SQL Server, con el código que aparece a continuación.

```
CREATE PROCEDURE LISTACLIENTES
AS
SELECT * FROM Clientes
CREATE PROCEDURE LISTAARTICULOS
AS
SELECT * FROM Articulos
```

Configurar el app.config con el siguiente código de manera que aparezca como en la Figura 50.

```
<?xml version="1.0" encoding="utf-8" ?>
<conFiguration>
   <startup>
      <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2" />
   </startup>
   <connectionStrings>
     <add name="cn"
     connectionString="server=LIZETH\SQLSERVER;database=Sistema;integrated
security=sspi"/>
   </connectionStrings>
</conFiguration>
```

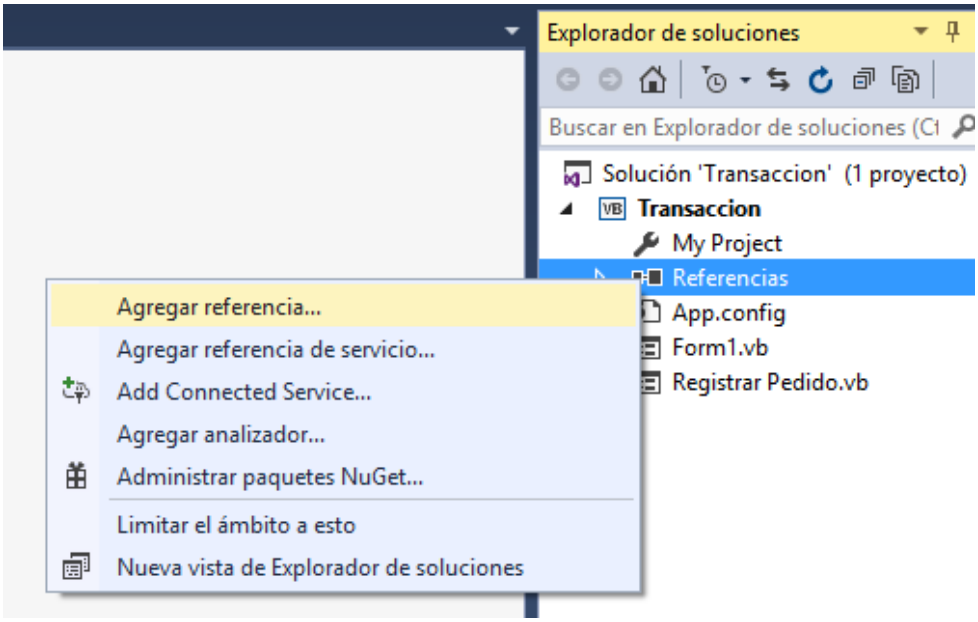
Figura 50. Código para la configuración del archivo app.config.



Para poder trabajar con app.config es necesario agregar la referencia.

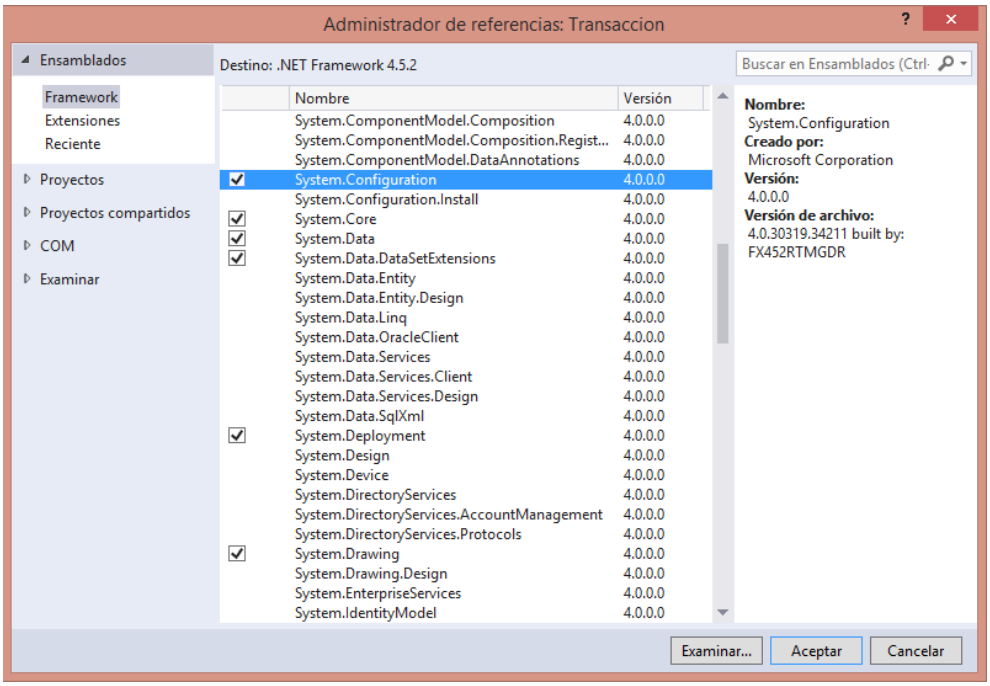
En la ventana de “Explorador de soluciones”, de clic derecho sobre la pestaña “Referencias” – “Agregar referencia”. Realice esto como lo muestra la Figura 51.

Figura 51. Agregar nueva referencia con app.config.



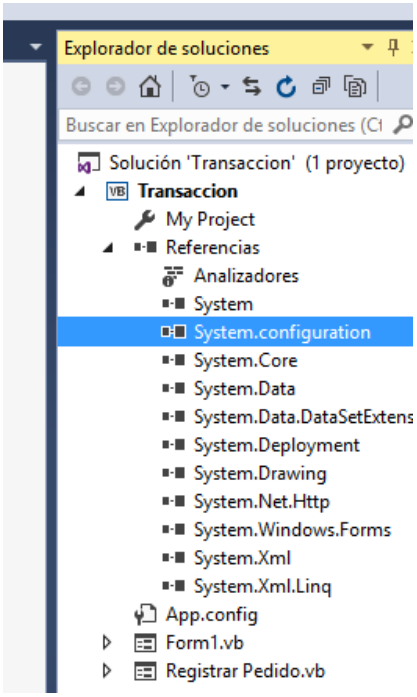
Una vez que abrió la ventana de "Administrador de referencias", activar la casilla de "System.ConFiguration" y de clic en "Aceptar". En la pantalla se mostrará algo como en la Figura 52.

Figura 52. Activación de la referencia "System.Configuration".



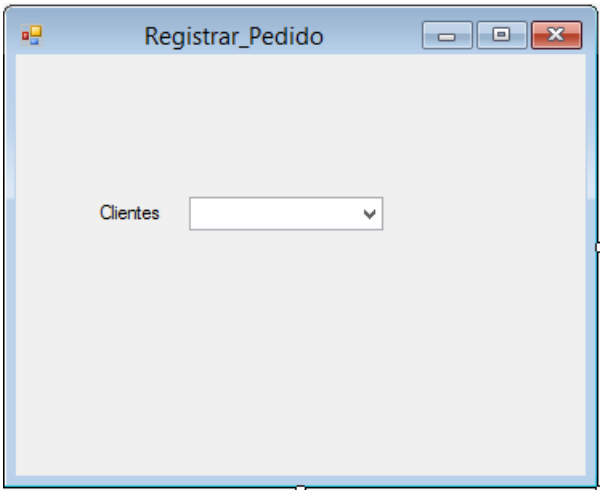
Para verificar que se ha añadido correctamente, se deberá visualizar en el “Explorador de soluciones” (lado derecho) el nuevo elemento agregado tal como aparece en la Figura 53.

Figura 53. Verificación de nueva referencia.



Agregar un “Combo Box” para comprobar que las conexiones a SQL se realizaron correctamente. Véase la Figura 54.

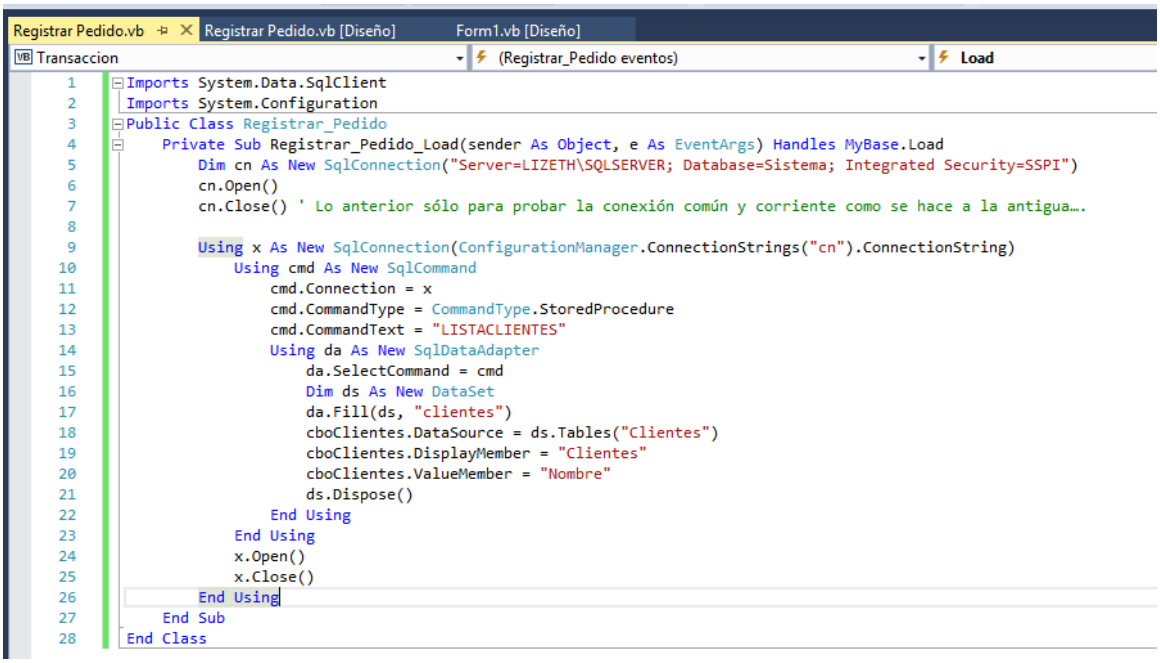
Figura 54. Interfaz para la comprobación de funcionamiento de conexiones.



Teclear el siguiente código en el formulario de la Figura 54. Una vez ya agregado lucirá como la Figura 55.

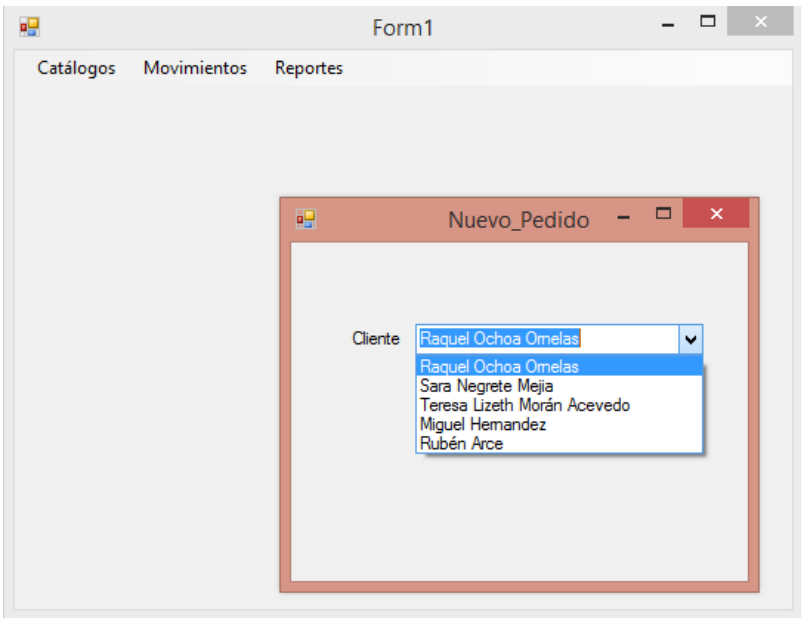
```
Imports System.Data.SqlClient
Imports System.ConFiguration

Public Class Nuevo_Pedido
  Private Sub Nuevo_Pedido_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    Dim cn As New SqlConnection("Server=LIZETH\SQLSERVER; Database=Sistema; Integrated Security=SSPI")
    cn.Open()
    cn.Close() ' Lo anterior sólo para probar la conexión común y corriente como se hace a la antigua….

Using x As New SqlConnection(ConFigurationManager.ConnectionStrings("cn").ConnectionString)
      Using cmd As New SqlCommand
        cmd.Connection = x
        cmd.CommandType = CommandType.StoredProcedure
        cmd.CommandText = "LISTACLIENTES"
        Using da As New SqlDataAdapter
          da.SelectCommand = cmd
          Dim ds As New DataSet
          da.Fill(ds, "clientes")
          cboClientes.DataSource = ds.Tables("Clientes")
          cboClientes.DisplayMember = "Clientes"
          cboClientes.ValueMember = "Nombre"
          ds.Dispose()
        End Using
      End Using
      x.Open()
      x.Close()
    End Using

  End Sub
End Class
```

Figura 55. Interfaz con código para el ComboBox de los clientes.



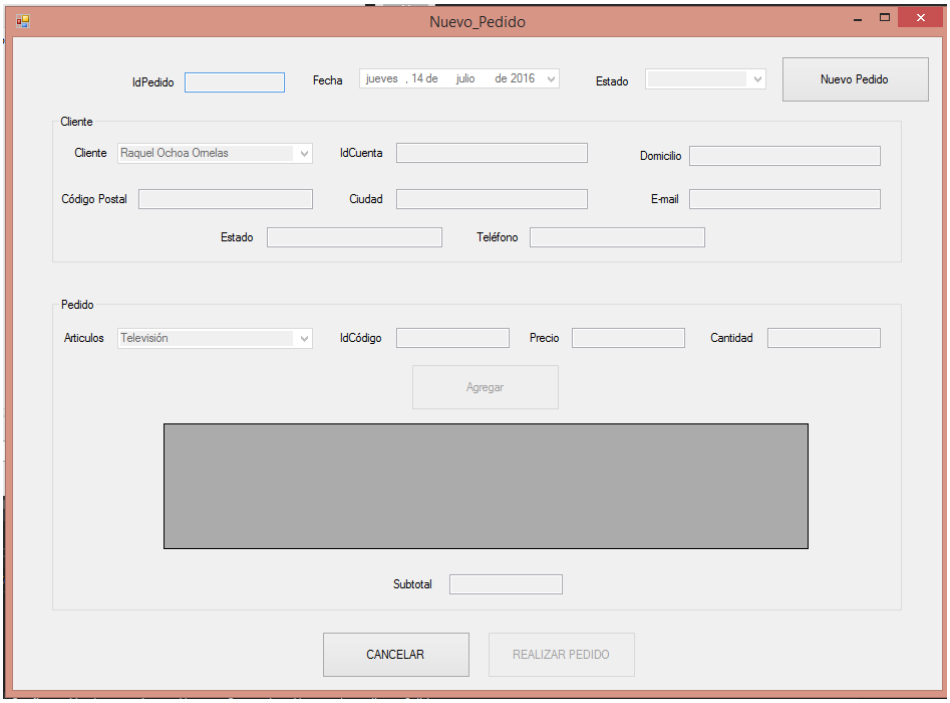
En la Figura 56 se muestra la comprobación del correcto funcionamiento de las conexiones y el código antes agregado.

Figura 56. Ejecución del formulario “Nuevo Pedido”.



Una vez que ya se ha verificado la funcionalidad de las conexiones a SQL Server, continuar con el diseño de pantalla para completar el movimiento “Realizar Pedido”. Se puede tomar de ejemplo el formulario de la Figura 57.

Figura 57. Formulario para agregar “Nuevo Pedido”.



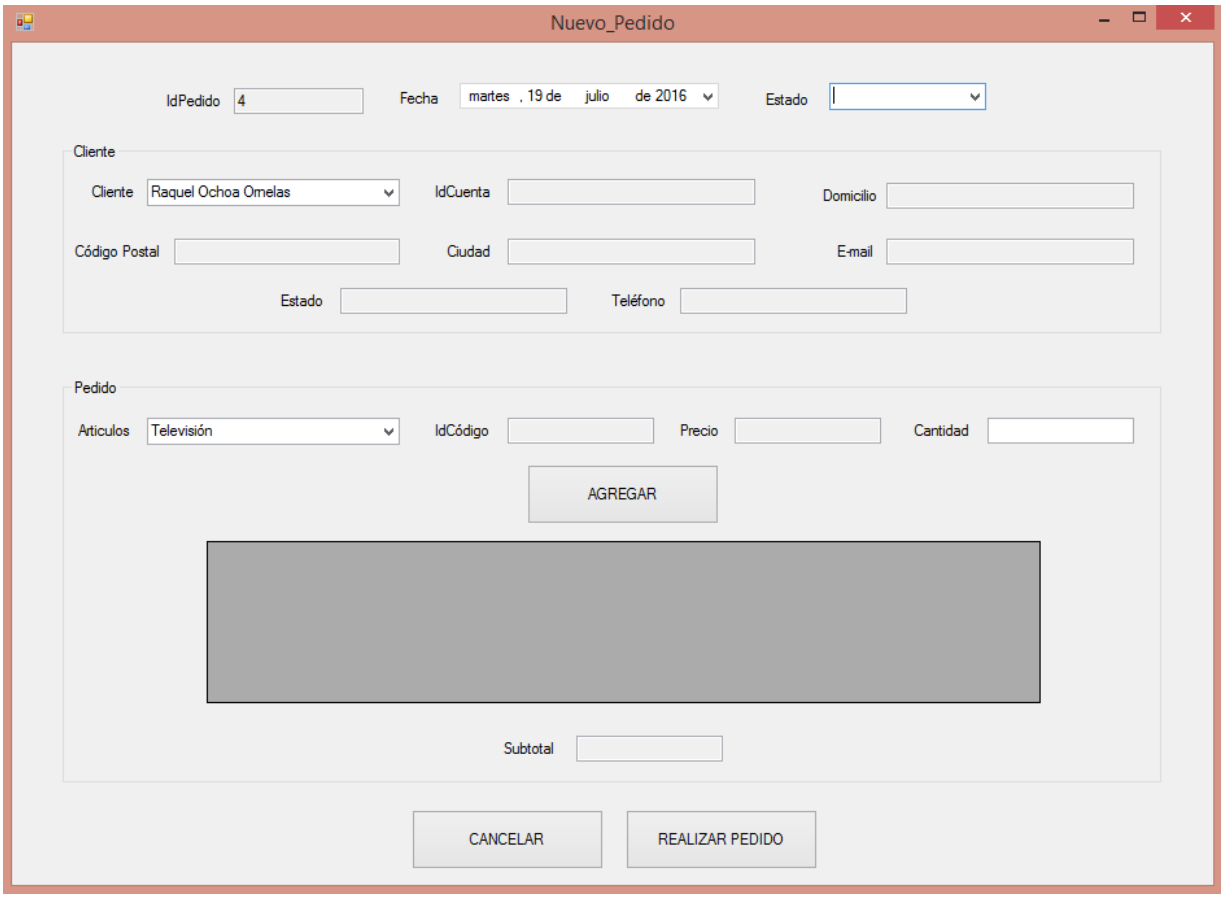
Para asignar el IdPedido de manera automática (es decir, que el usuario no sea quien lo asigne, si no el propio sistema), es necesario hacer un acceso a la tabla Pedidos, sacar el último id registrado, y sumarle 1. El código quedaría de la siguiente manera:

```
Using x As New
SqlConnection(ConFigurationManager.ConnectionStrings("cn").ConnectionString)
        x.Open()
        Using cmd As New SqlCommand
           cmd.Connection = x
           cmd.CommandType = CommandType.Text
           cmd.CommandText = "Select COUNT(*) from Pedidos"
           Using da As New SqlDataAdapter
              da.SelectCommand = cmd
              Dim ds As New DataSet
              da.Fill(ds, "pedidos")
              Dim cantidadRegistros As Integer = cmd.ExecuteScalar()
              If cantidadRegistros = 0 Then
                 txtIdPedido.Text = 1
              Else
```

```
                txtIdPedido.Text = cantidadRegistros + 1
            End If
        End Using
    End Using
    x.Close()
End Using
```

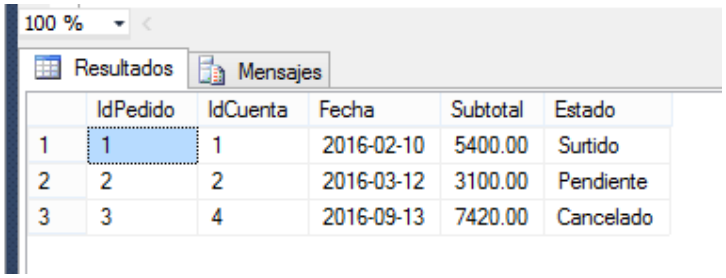
La Figura 58 muestra la ejecución de la aplicación para comprobar la asignación del IdPedido.

Figura 58. Ejecución de aplicación para comprobar asignación de IdPedido.



En la Figura 59 se muestra la verificación de los datos en SQL Server (En este caso ya hay 3 registros almacenados en la tabla "Pedidos" por lo que en el TextBox del idPedido aparece 4).

Figura 59. Datos existentes en SQL Server.

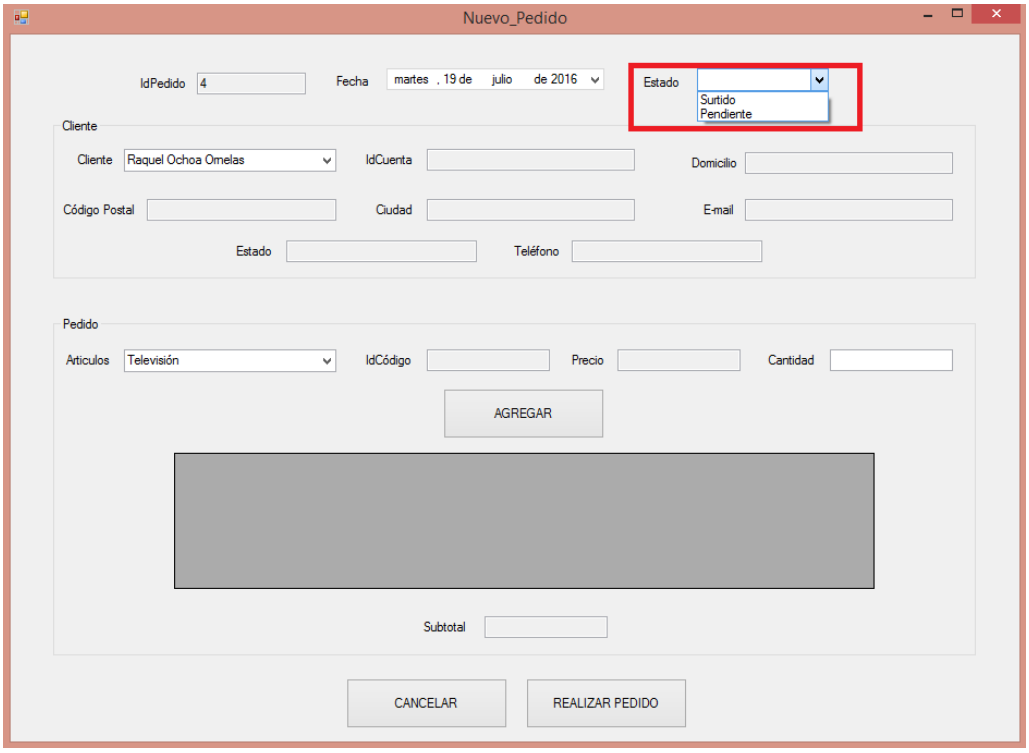


	IdPedido	IdCuenta	Fecha	Subtotal	Estado
1	1	1	2016-02-10	5400.00	Surtido
2	2	2	2016-03-12	3100.00	Pendiente
3	3	4	2016-09-13	7420.00	Cancelado

Respecto al estado del pedido, son datos que no están contenidos en la base de datos por lo que hay que agregarlos manualmente (teniendo en cuenta que entre menos información teclee el usuario, menos errores se tendrán que validar). En la Figura 60 se muestra el código y la demostración en ejecución.

```
cboEstadoPedido.Items.Add("Surtido")
cboEstadoPedido.Items.Add("Pendiente")
```

Figura 60. Demostración de los datos agregados al ComboBox del estado.



En la sección de Cliente, es necesario rellenar los cuadros de texto con la información referente al cliente que se seleccione en el combo por lo que es necesario teclear el siguiente código.

```
Private Sub cboClientes_SelectedIndexChanged(sender As Object, e As EventArgs) Handles cboClientes.SelectedIndexChanged
    Using x As New SqlConnection(ConFigurationManager.ConnectionStrings("cn").ConnectionString)
        x.Open()
        Using cmd As New SqlCommand
            cmd.Connection = x
            cmd.CommandType = CommandType.Text
            cmd.CommandText = "Select * from Clientes where Nombre = '" & cboClientes.Text & "'"
            Lector = cmd.ExecuteReader
            While Lector.Read()
                'Llena los datos de los TextBox
                txtIdCuenta.Text = Lector(0)
                txtDomicilio.Text = Lector(2)
                txtCP.Text = Lector(3)
                txtCiudad.Text = Lector(4)
                txtEmail.Text = Lector(5)
                txtEstadoCliente.Text = Lector(6)
                txtTelefono.Text = Lector(7)
            End While
        End Using
        x.Close()
    End Using
End Sub
```

Como ejemplo ver las Figuras 61, 62 y 63.

Figura 61. Ejecución de proyecto para visualizar los clientes en el ComboBox.

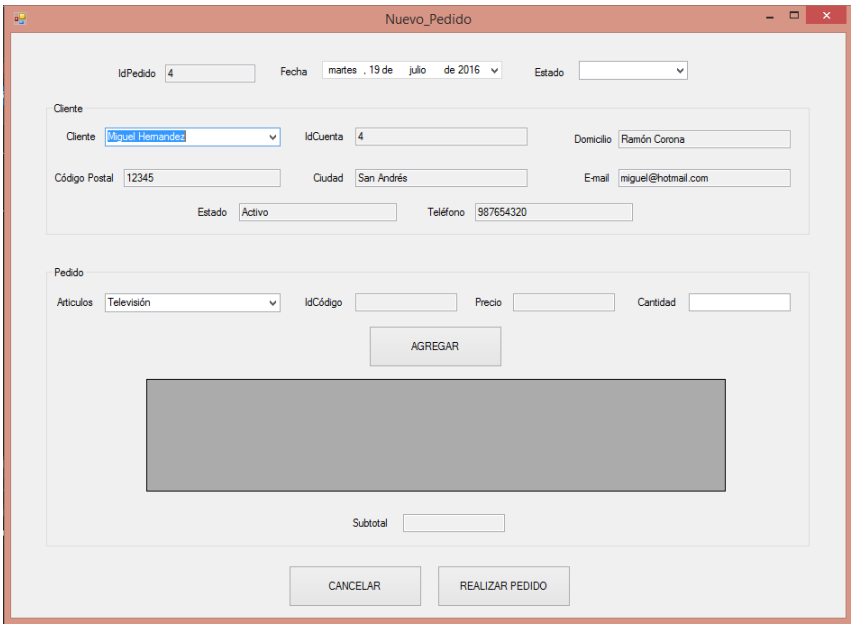


Figura 62. Cambio de Cliente (Comprobación 1).

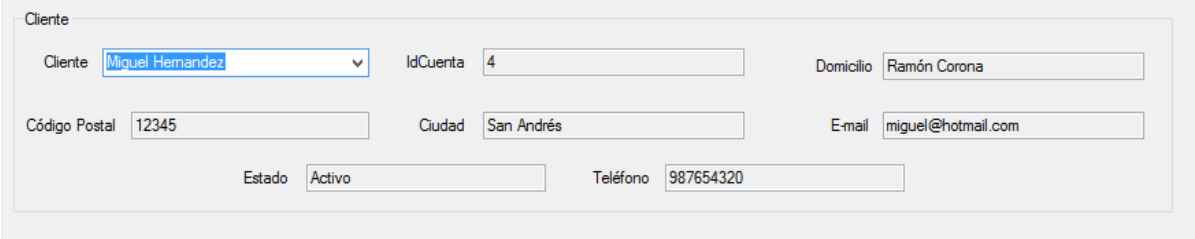
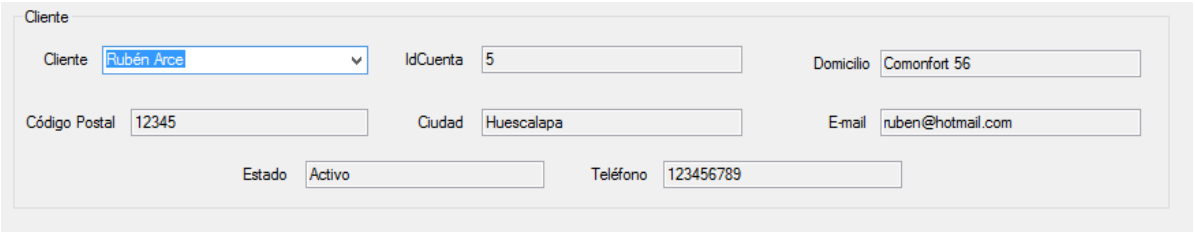
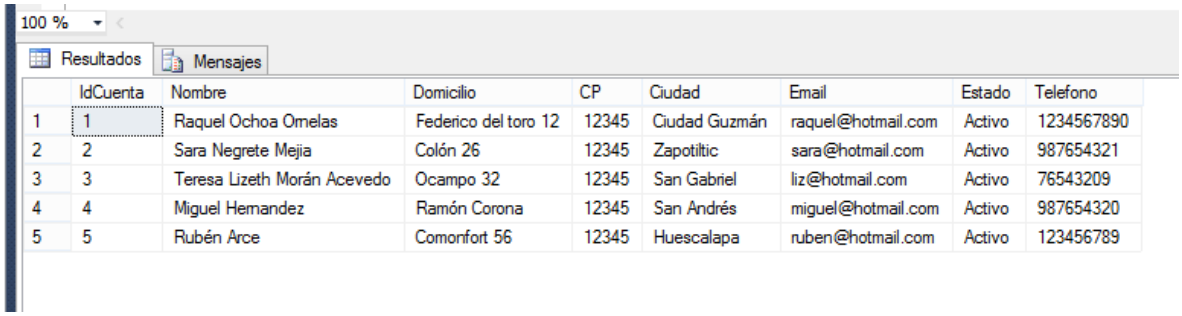


Figura 63. Cambio de Cliente (Comprobación 2).



Verificar la información con la base de datos como lo muestra la Figura 64.

Figura 64. Verificación de información en la base de datos en SQL Server.



	IdCuenta	Nombre	Domicilio	CP	Ciudad	Email	Estado	Telefono
1	1	Raquel Ochoa Omelas	Federico del toro 12	12345	Ciudad Guzmán	raquel@hotmail.com	Activo	1234567890
2	2	Sara Negrete Mejia	Colón 26	12345	Zapotiltic	sara@hotmail.com	Activo	987654321
3	3	Teresa Lizeth Morán Acevedo	Ocampo 32	12345	San Gabriel	liz@hotmail.com	Activo	76543209
4	4	Miguel Hernandez	Ramón Corona	12345	San Andrés	miguel@hotmail.com	Activo	987654320
5	5	Rubén Arce	Comonfort 56	12345	Huescalapa	ruben@hotmail.com	Activo	123456789

En la sección del Pedido, es necesario rellenar los cuadros de texto con los datos del artículo seleccionado en el ComboBox. El código quedaría de la siguiente manera. Comprobando su ejecución en la Figura 65 y 66.

```
Private Sub cboArticulos_SelectedIndexChanged(sender As Object, e As EventArgs)
Handles cboArticulos.SelectedIndexChanged
    Using x As New
SqlConnection(ConFigurationManager.ConnectionStrings("cn").ConnectionString)
        x.Open()
        Using cmd As New SqlCommand
            cmd.Connection = x
```

```
            cmd.CommandType = CommandType.Text
            cmd.CommandText = "Select * from Articulos where Nombre = '" &
cboArticulos.Text & "'"
            cmd.CommandText = "select IdCodigo,Tipos.Concepto,precio from Articulos
inner join  Tipos on Tipos.IdTipo = Articulos.IdTipo where Articulos.Descripcion = '" &
cboArticulos.Text & "'"
            Lector = cmd.ExecuteReader
            While Lector.Read()
               'Llena los datos de los TextBox
               txtIdCodigo.Text = Lector(0)
               txtTipoArticulo.Text = Lector(1)
               txtPrecio.Text = Lector(2)
            End While
        End Using
        x.Close()
    End Using
End Sub
```

Figura 65. Comprobación 1 de llenado de TextBox de acuerdo a selección en ComboBox.

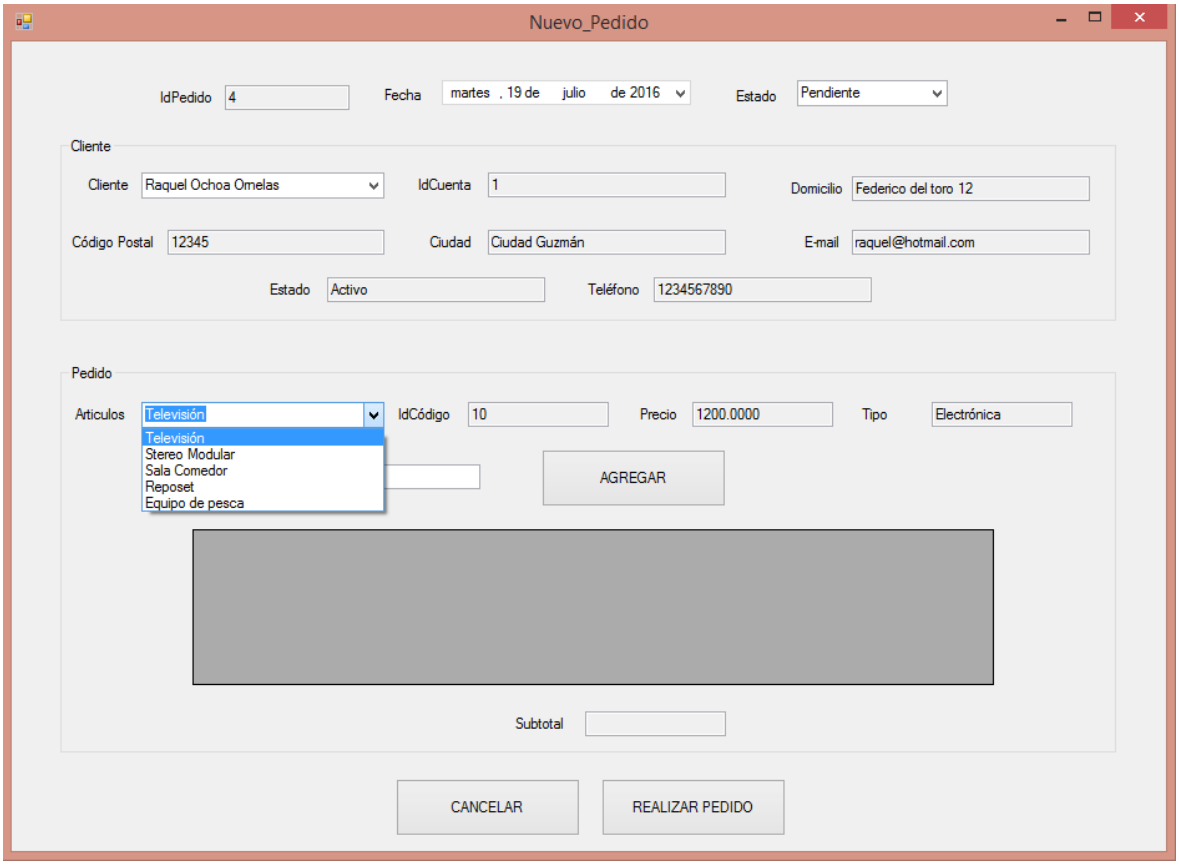
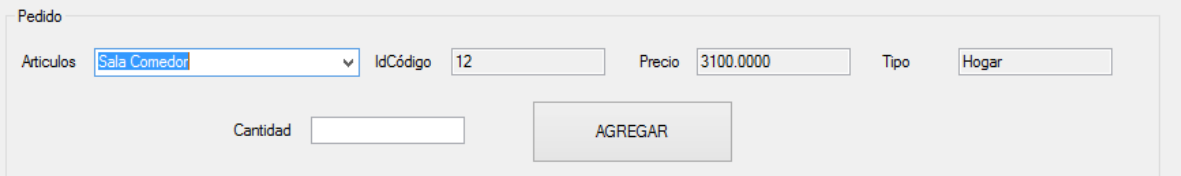
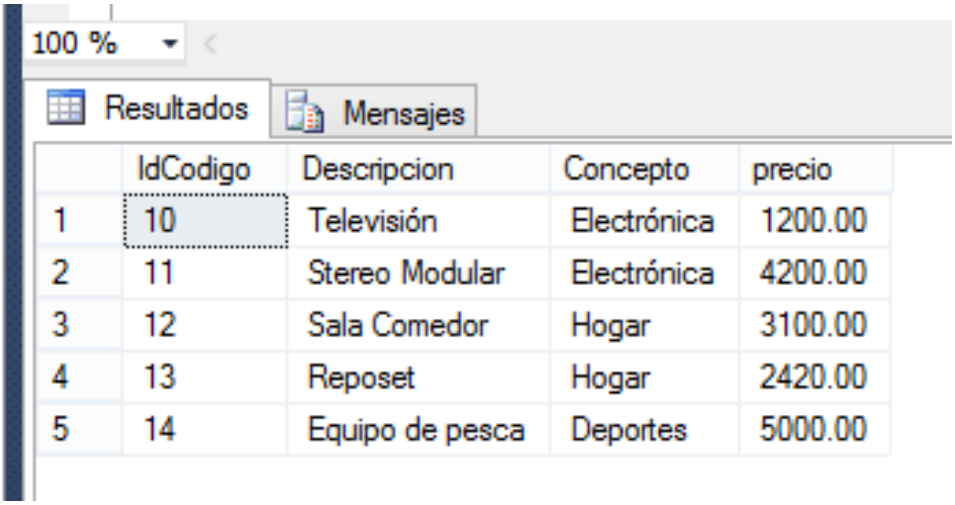


Figura 66. Comprobación 2 de llenado de TextBox de acuerdo a selección en ComboBox.



Verificar la información que aparece en el formulario, con la que está almacenada en la base de datos en SQL Server tal y como se muestra en la Figura 67.

Figura 67. Comprobación de datos en SQL Server.

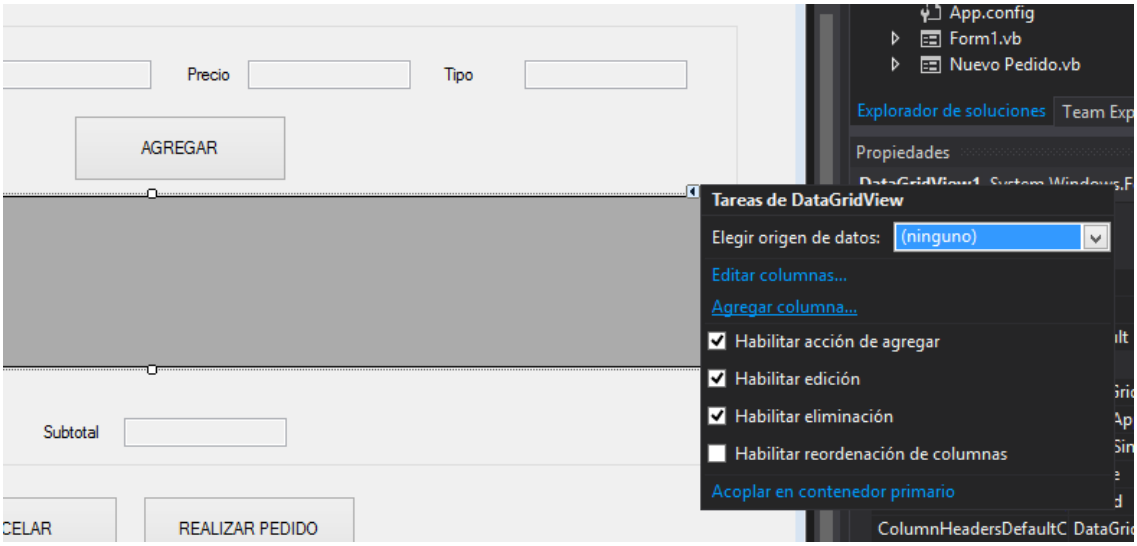


	IdCodigo	Descripcion	Concepto	precio
1	10	Televisión	Electrónica	1200.00
2	11	Stereo Modular	Electrónica	4200.00
3	12	Sala Comedor	Hogar	3100.00
4	13	Reposet	Hogar	2420.00
5	14	Equipo de pesca	Deportes	5000.00

El DataGridView va a fungir como listado de artículos según se vayan agregando al dar clic al botón “Agregar”, por lo que hay que configurarlo de la siguiente manera:

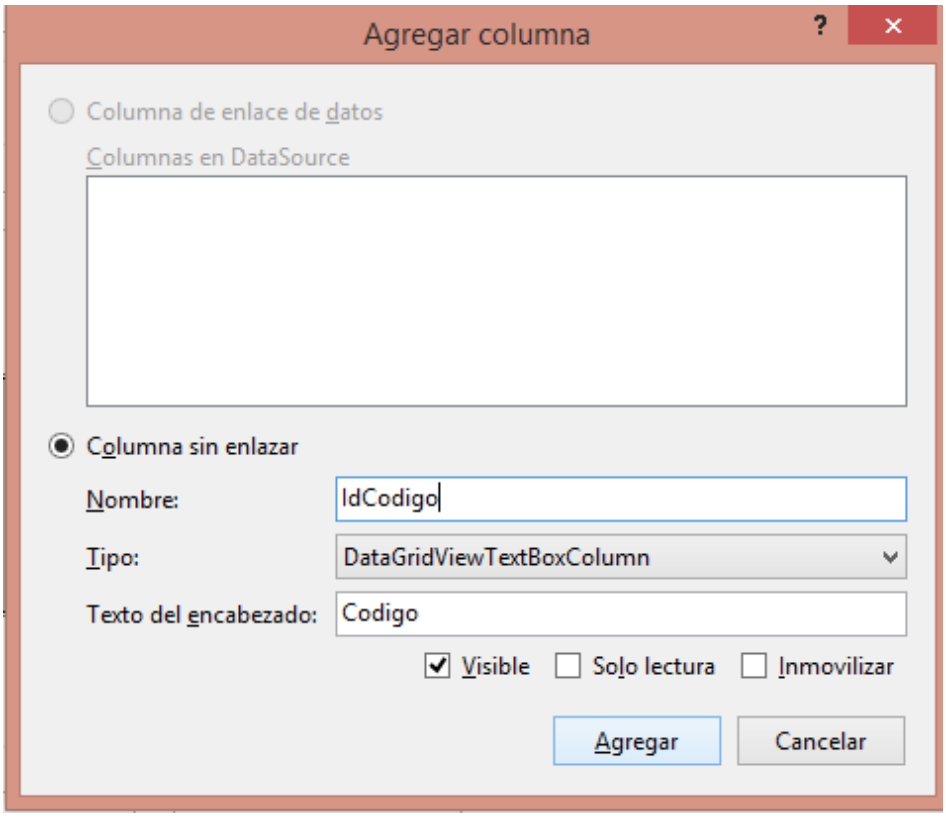
Previamente hay que agregar las columnas al DataGridView, dando clic en la flechita superior del mismo y seleccionando la opción “Agregar Columna”, tal y como se muestra en la Figura 68.

Figura 68. Opción para agregar columnas en DataGridView.



Una vez dando clic en "Agregar columnas" agregar las necesarias. Tomar como ejemplo la Figura 69. (Hacer lo mismo para todas las columnas que se deseen agregar al DataGridView).

Figura 69. Interfaz para agregar la columna IdCodigo.

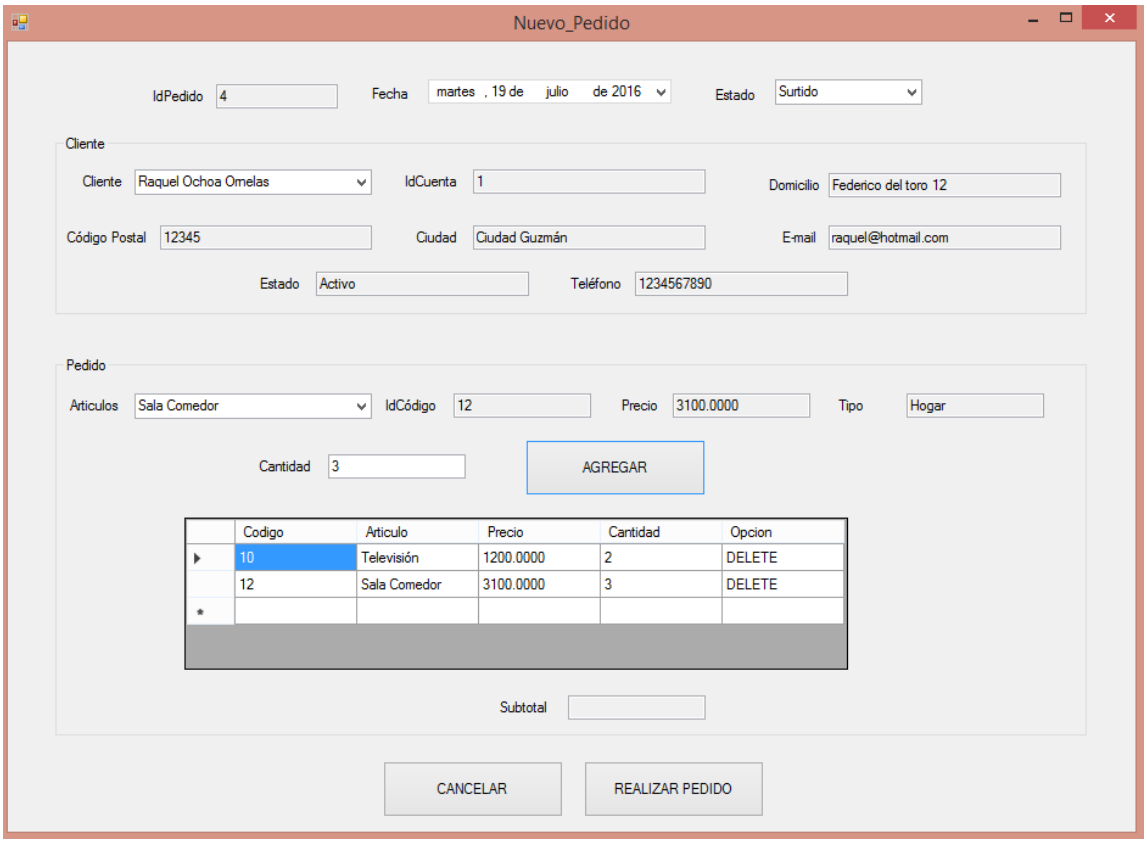


Una vez realizado lo anterior, es necesario codificar el botón de “Agregar” del formulario “Nuevo Pedido” con el siguiente código:

```
Private Sub cmdAgregar_Click(sender As Object, e As EventArgs) Handles cmdAgregar.Click
    DataGridView1.Rows.Add(txtIdCodigo.Text, cboArticulos.Text, txtPrecio.Text, txtCantidad.Text, "DELETE")
End Sub
```

La Figura 70 muestra la ejecución del proyecto en la cual se observan los datos que son agregados al DataGridView desde el botón “Agregar”.

Figura 70. Ejecución de proyecto para verificación de artículos a DataGridView.



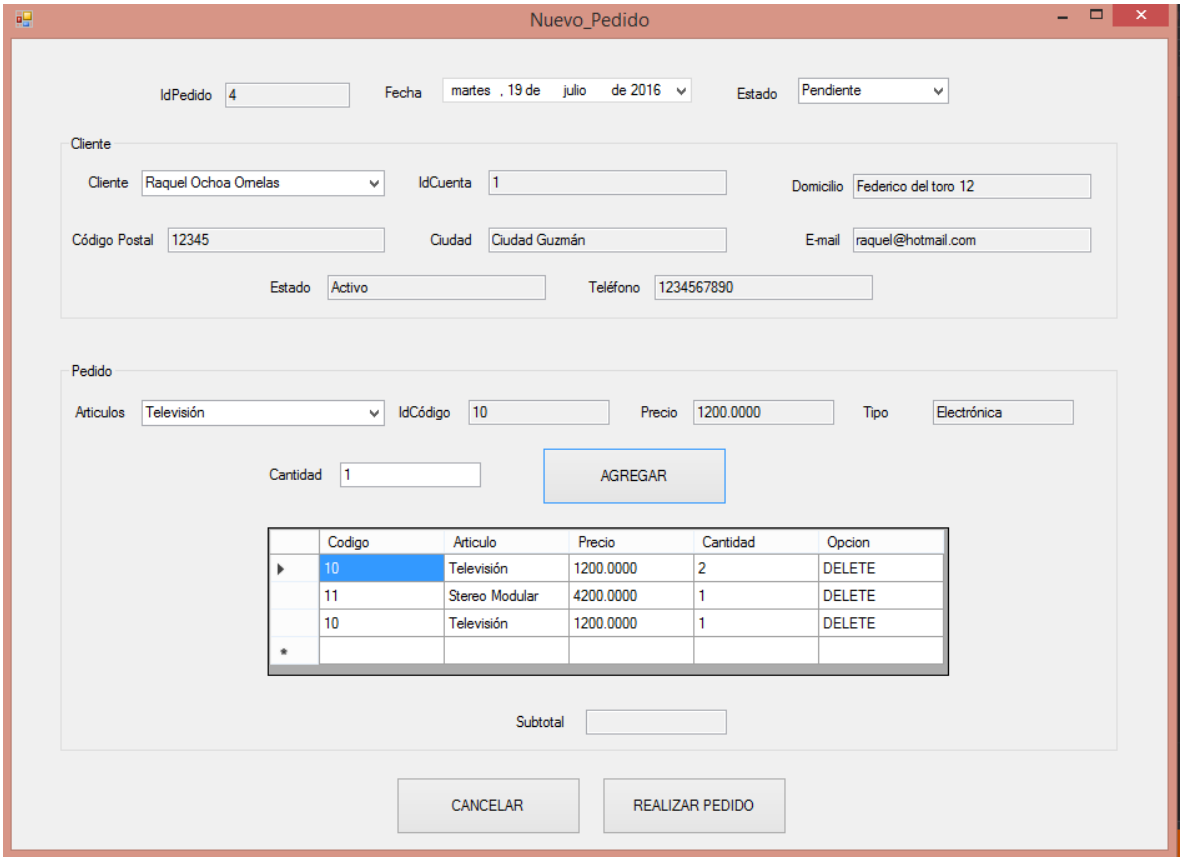
Ahora hay que darle la opción al usuario de eliminar productos de esa lista, para esto es conveniente agregar el siguiente código como método independiente.

```
Private Sub eliminar(sender As Object, e As DataGridViewCellEventArgs) Handles DataGridView1.CellClick
    If DataGridView1.CurrentCell.ColumnIndex = 4 And DataGridView1.CurrentRow.Index <> -1 Then
        DataGridView1.Rows.Remove(DataGridView1.CurrentRow)
```

```
        End If

        If DataGridView1.Rows.Count > 0 Then
            DataGridView1.Enabled = True
        Else
            DataGridView1.Enabled = False
        End If
    End Sub
```

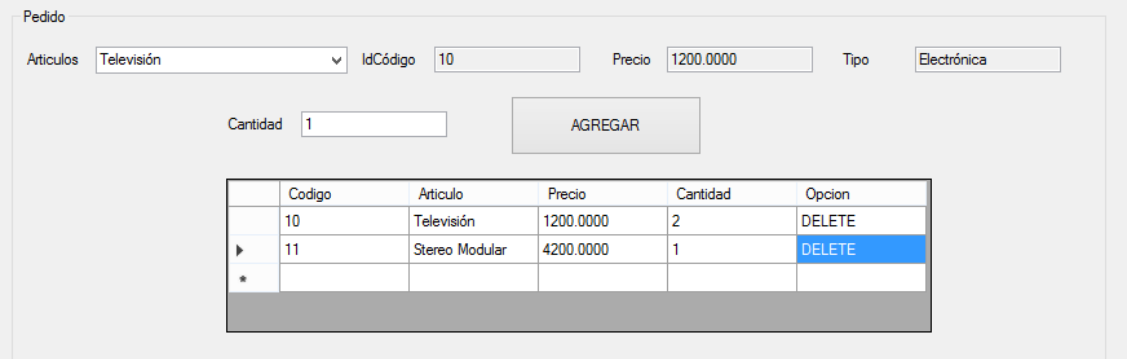
Posteriormente hay que ejecutar para comprobar la funcionalidad del método (véase la Figura 71 y 72).

Figura 71. Ejecución de proyecto para comprobación del método eliminar (1).



En este listado se tienen 3 productos pero una vez que se detecte clic en cualquier celda que contenga la frase “DELETE” se eliminará todo el registro del DataGridView.

Figura 72. Ejecución de proyecto para comprobación del método eliminar (2).



Una vez que se configuró el "Listado" hay que hacer que en el TextBox de Subtotal se actualice cada vez que se agreguen datos en el DataGridView. El código es el siguiente en el evento Click del botón correspondiente:

Previamente hay que declarar a nivel de clase la variable suma de la siguiente manera: Dim suma As Double.

```
Private Sub cmdAgregar_Click(sender As Object, e As EventArgs) Handles cmdAgregar.Click
    DataGridView1.Rows.Add(txtIdCodigo.Text, cboArticulos.Text, txtPrecio.Text, txtCantidad.Text, "DELETE")
    txtCantidad.Text = ""
    Dim fila As Integer
    fila = DataGridView1.Rows.Count - 2

    If (txtSubtotal.Text = "") Then
        suma = (DataGridView1.Rows(fila).Cells(2).Value * DataGridView1.Rows(fila).Cells(3).Value)
    Else
        suma = suma + (DataGridView1.Rows(fila).Cells(2).Value * DataGridView1.Rows(fila).Cells(3).Value)
    End If
    txtSubtotal.Text = suma

End Sub
```

En la Figura 73 y 74 se muestra la ejecución para la opción de “Borrado” desde el DataGridView.

Figura 73. Ejecución del proyecto probando el borrado desde DataGridView (1).

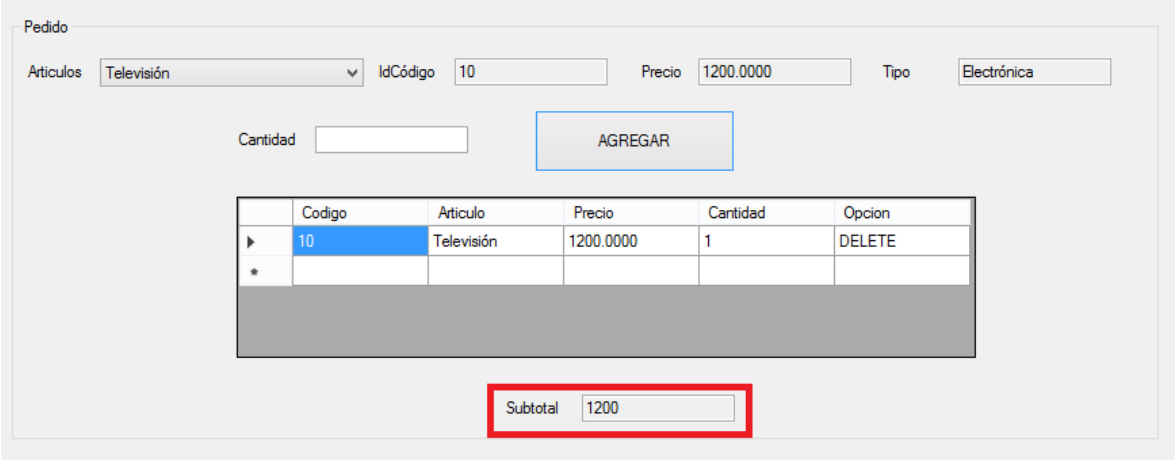
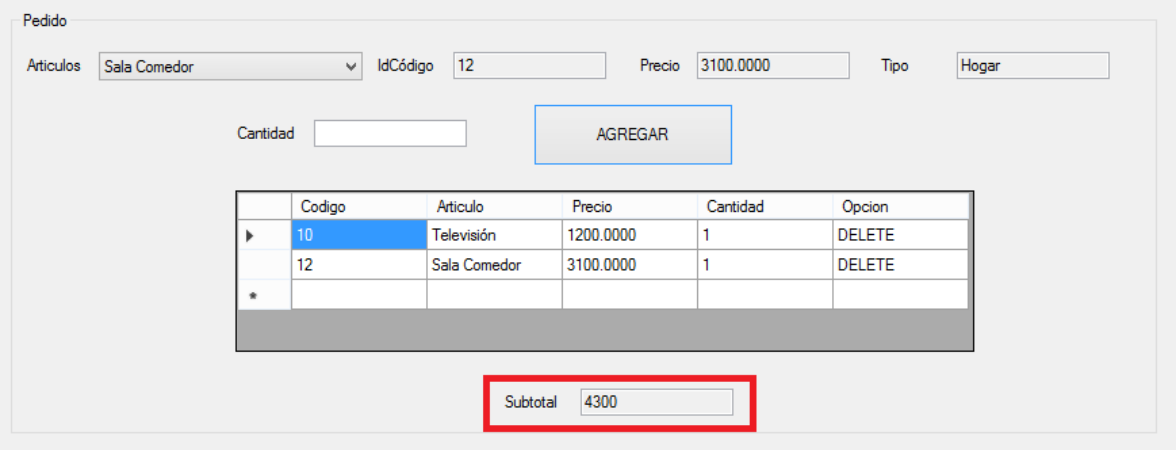


Figura 74. Ejecución del proyecto probando el borrado desde DataGridView (2).



En el caso de que el usuario necesite eliminar una opción del DataGridView, se tendrá que **actualizar el Subtotal** después de la eliminación, por lo que se modificará el código del método “eliminar” (anteriormente agregado) quedando de la siguiente manera:

```
Private Sub eliminar(sender As Object, e As DataGridViewCellEventArgs) Handles DataGridView1.CellClick
    Dim resta As Double
```

```
        If DataGridView1.CurrentCell.ColumnIndex = 4 And
DataGridView1.CurrentRow.Index <> -1 Then
            'Se toman los valores de las celdas para realizar la resta al subtotal
            resta = (DataGridView1.Rows(e.RowIndex).Cells(2).Value *
DataGridView1.Rows(e.RowIndex).Cells(3).Value)
            suma = suma - resta
            txtSubtotal.Text = suma
            'Se elimina el registro del DataGridView
            DataGridView1.Rows.Remove(DataGridView1.CurrentRow)
        End If

        If DataGridView1.Rows.Count > 0 Then
            DataGridView1.Enabled = True
        Else
            DataGridView1.Enabled = False
        End If
    End Sub
```

Para verificar que realmente se eliminó la cantidad del subtotal, ejecutar la aplicación como se muestra en la Figura 75, 76 y 77.

Figura 75. Prueba de eliminación de cantidad al subtotal (1).

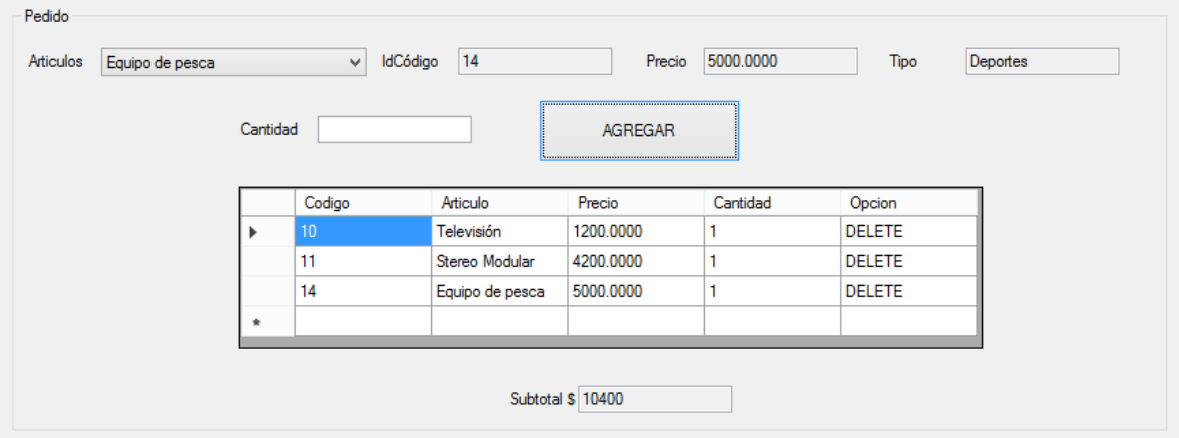


Figura 76. Prueba de eliminación de cantidad al subtotal (2).

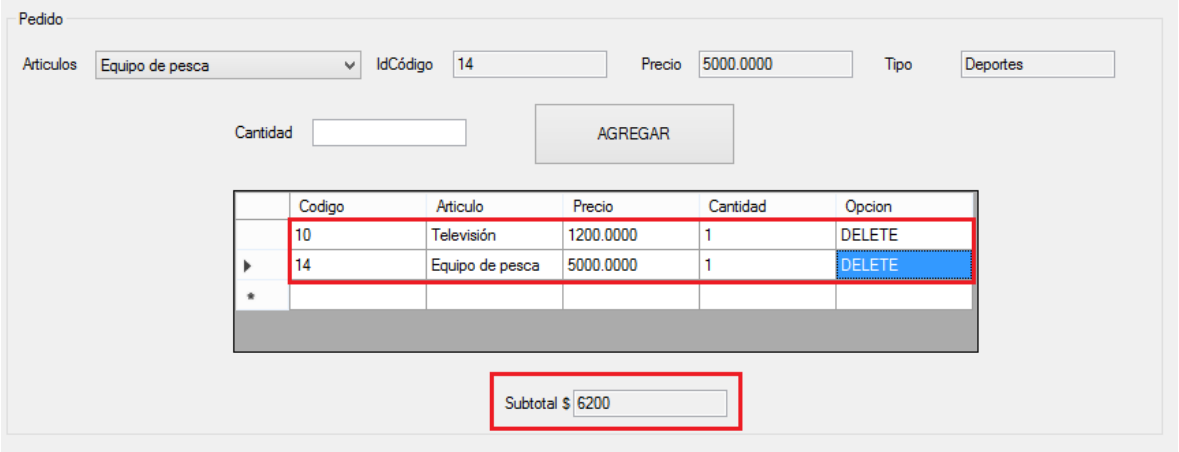
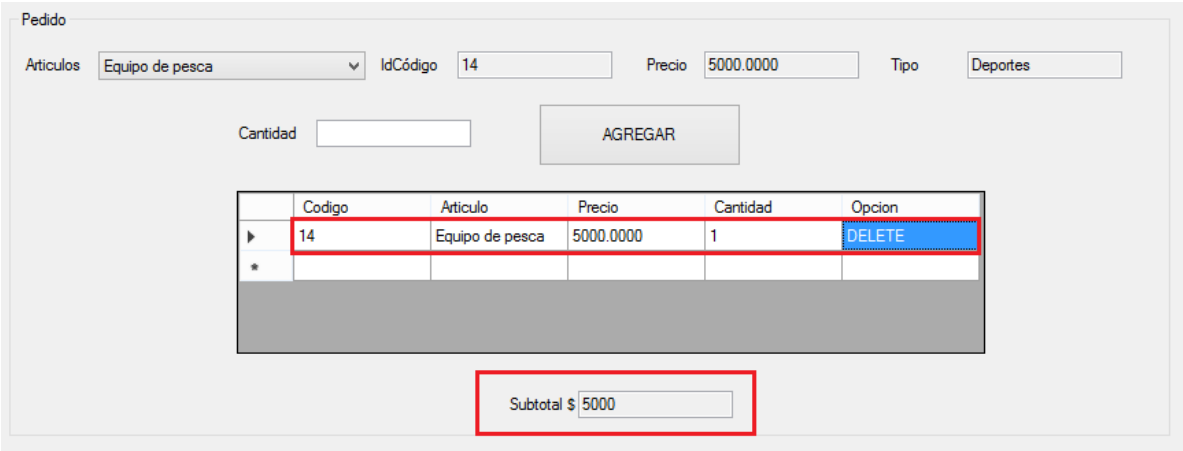


Figura 77. Prueba de eliminación de cantidad al subtotal (3).



Una vez que ya se tiene todo funcional, es necesario realizar Transacciones, para en el caso de que por alguna razón se interrumpa el registro, no registrar cambios en la base de datos y se presenten inconsistencias en la información.

Para realizar la **TRANSACCIÓN** hay que codificar lo siguiente en el evento click del botón de "Realizar Pedido":

(Previamente hay que declarar a nivel de clase la bandera: Dim bandera As Integer = 0)

```
Private Sub cmdRealizarPedido_Click(sender As Object, e As EventArgs) Handles cmdRealizarPedido.Click
    Using x As New SqlConnection(ConFigurationManager.ConnectionStrings("cn").ConnectionString)
```

```
        x.Open()
        Using cmd As New SqlCommand
            cmd.CommandType = CommandType.Text
            Dim transaccion As SqlTransaction

            'Iniciar transaccion local
            transaccion = x.BeginTransaction("Registrar Pedido")

            'Asignacion  del objeto  de la transaccion  y la conexion a un comando para la transaccion  local pendiente
            cmd.Connection = x
            cmd.Transaction = transaccion


            Try

                'Sentencia para agregar nuevo pedido
                cmd.CommandText = "Insert into Pedidos(IdPedido,IdCuenta,Fecha,Subtotal,Estado) values('" & txtIdPedido.Text & "','" & txtIdCuenta.Text & "','" & dtpFechaPedido.Value.Date & "','" & txtSubtotal.Text & "','" & cboEstadoPedido.Text & "')"
                cmd.ExecuteNonQuery()

                'Recorre el DataGridView para almacenar cada registro del DetallePedido
                For i = 0 To DataGridView1.Rows.Count - 2
                    cmd.CommandText = "Insert into DetallePedido(IdPedido,IdCodigo,Precio,Cantidad) values('" & txtIdPedido.Text & "','" & DataGridView1.Item(0, i).Value & "','" & DataGridView1.Item(2, i).Value & "','" & DataGridView1.Item(3, i).Value & "')"
                    cmd.ExecuteNonQuery()
                Next

                'Intenta ejecutar la transacción
                If MsgBox("Desea ejecutar la transacción", MsgBoxStyle.YesNo, "ejecutar") = MsgBoxResult.Yes Then
                    transaccion.Commit()
                    'MsgBox("Ok")
                    bandera = 1
                Else
                    transaccion.Rollback()
                End If

            Catch ex As Exception
                MsgBox("Commit Exception Type: {0} No se pudo insertar por error")
                MsgBox("Error: " & ex.Message)
                Try
                    transaccion.Rollback()
```

```
            Catch ex2 As Exception
                'Este bloque  de catch manejará los errores que pueden  ser ocurridos en el
servidor
                ' y que podrían causar  el rollback tal como una conexion cerrada
                MsgBox("Error Rollback")
            End Try
        End Try
    End Using
End Using

If (bandera = 1) Then
    MsgBox("PEDIDO REGISTRADO")
Else
    MsgBox("PEDIDO CANCELADO")
End If
Me.Close()
End Sub
```

Las Figuras 78, 79 y 80 muestran el funcionamiento de las transacciones en el proyecto.

Figura 78. Pantalla de ejecución para prueba de transacciones en el proyecto (1).

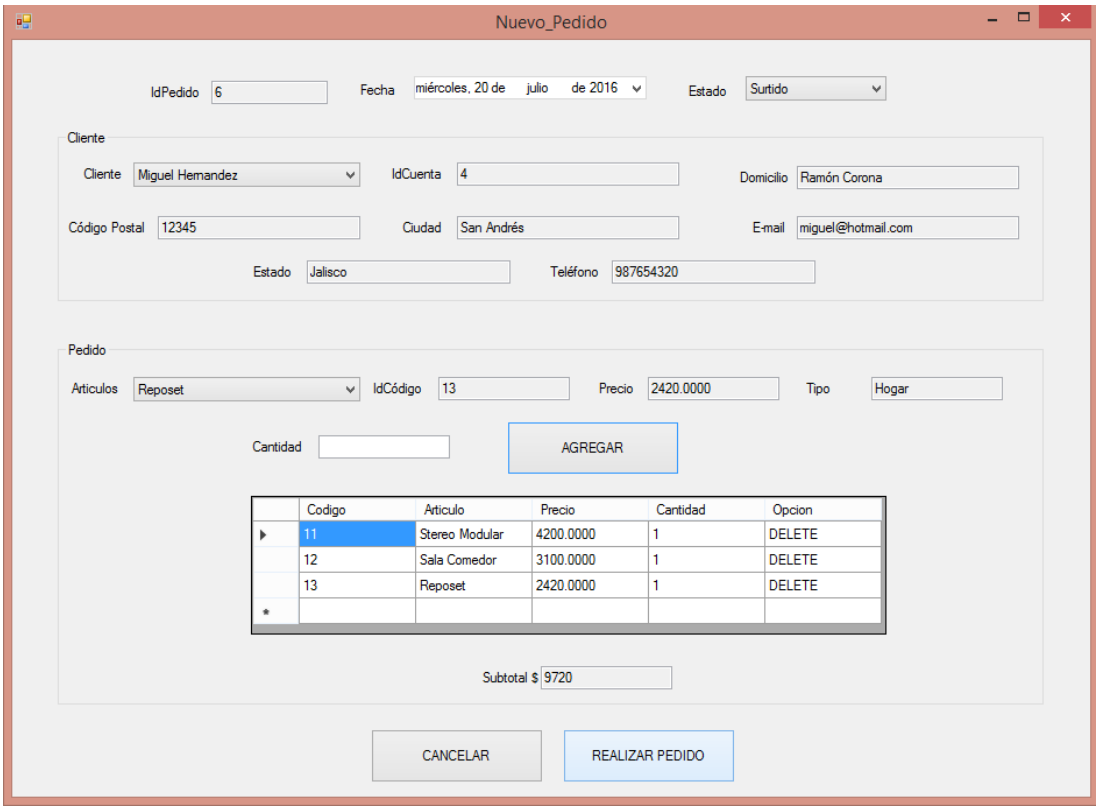


Figura 79. Pantalla de ejecución para prueba de transacciones en el proyecto (2).

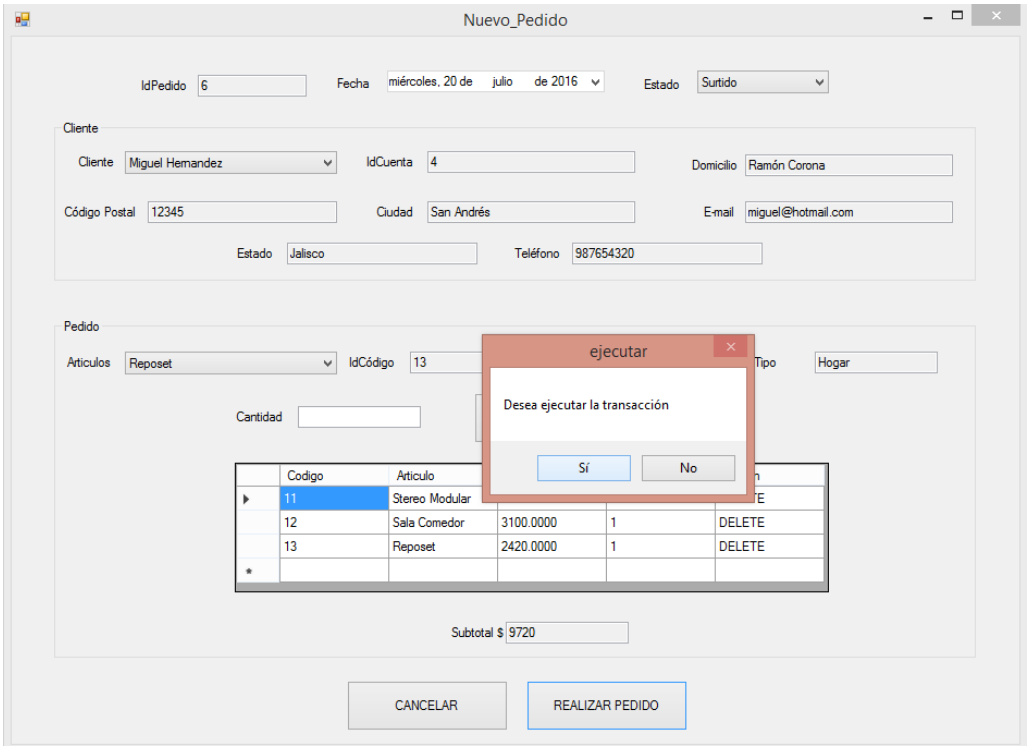
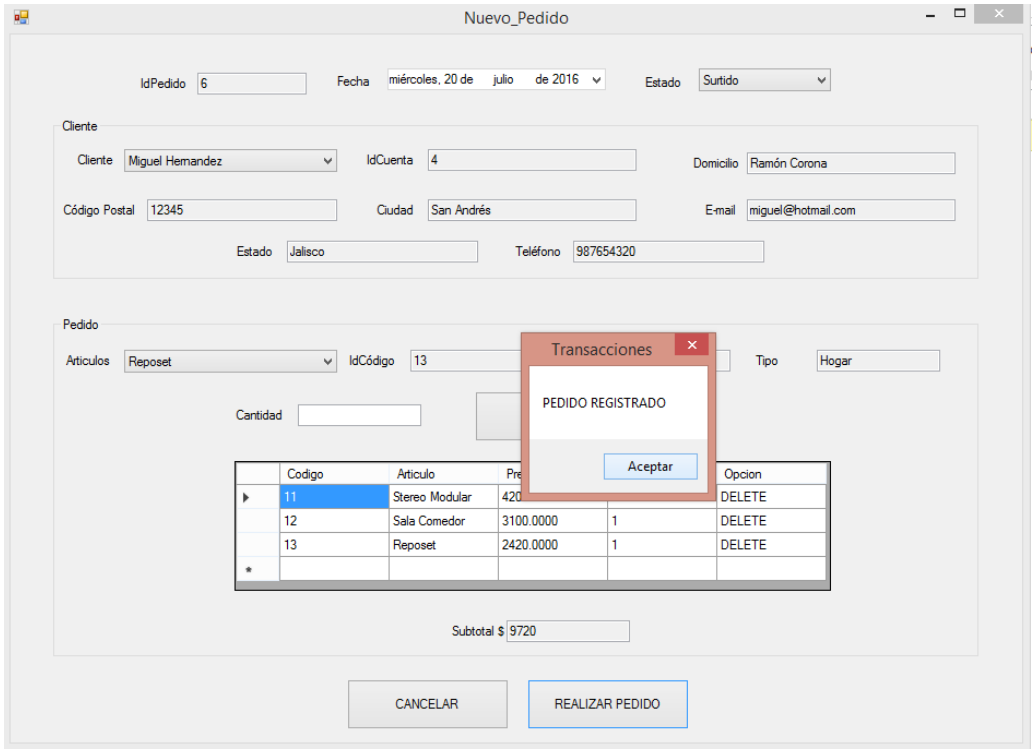
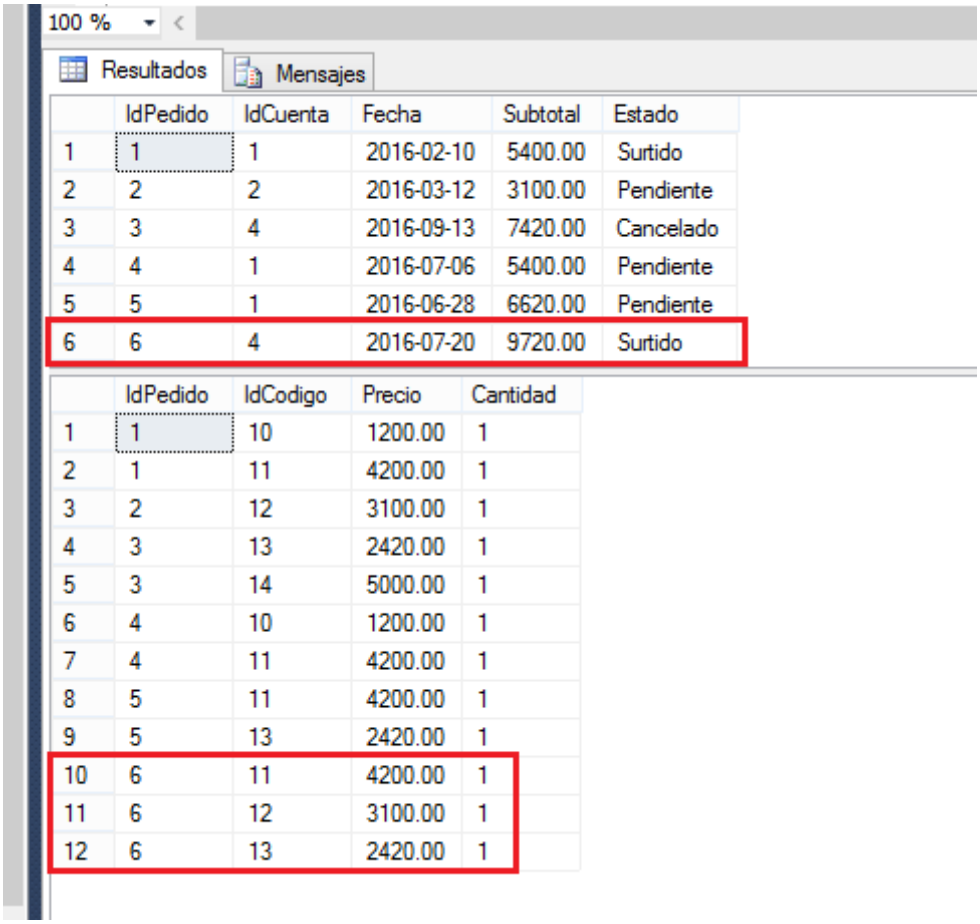


Figura 80. Pantalla de ejecución para prueba de transacciones en el proyecto (3).



Para comprobar que realmente la transacción se realizó y que la información fue correcta, revisar los registros en la base de datos, como lo muestra la Figura 81. Con esto se comprueban los datos recién almacenados con el IdPedido 6.

Figura 81. Comprobación de correcta ejecución de la transacción.



	IdPedido	IdCuenta	Fecha	Subtotal	Estado
1	1	1	2016-02-10	5400.00	Surtido
2	2	2	2016-03-12	3100.00	Pendiente
3	3	4	2016-09-13	7420.00	Cancelado
4	4	1	2016-07-06	5400.00	Pendiente
5	5	1	2016-06-28	6620.00	Pendiente
6	6	4	2016-07-20	9720.00	Surtido

	IdPedido	IdCodigo	Precio	Cantidad
1	1	10	1200.00	1
2	1	11	4200.00	1
3	2	12	3100.00	1
4	3	13	2420.00	1
5	3	14	5000.00	1
6	4	10	1200.00	1
7	4	11	4200.00	1
8	5	11	4200.00	1
9	5	13	2420.00	1
10	6	11	4200.00	1
11	6	12	3100.00	1
12	6	13	2420.00	1

Ahora hay que comprobar que al no ejecutar la transacción no realiza ningún cambio en la base de datos. Ver las Figuras 82, 83 y 84.

Figura 82. Ejecución de la aplicación cancelando la transacción (1).

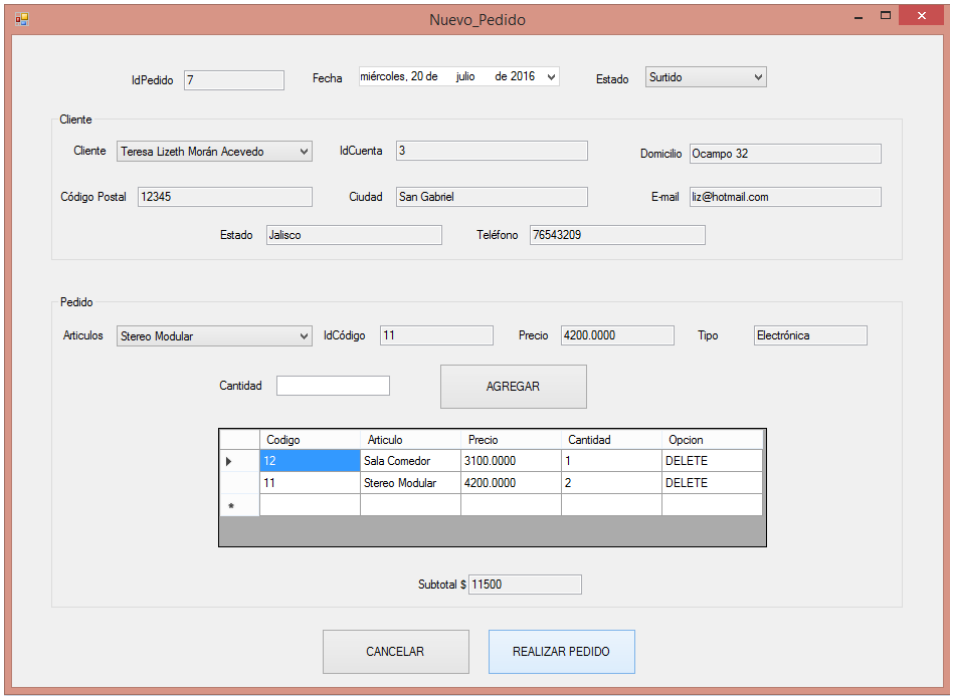


Figura 83. Ejecución de la aplicación cancelando la transacción (2).

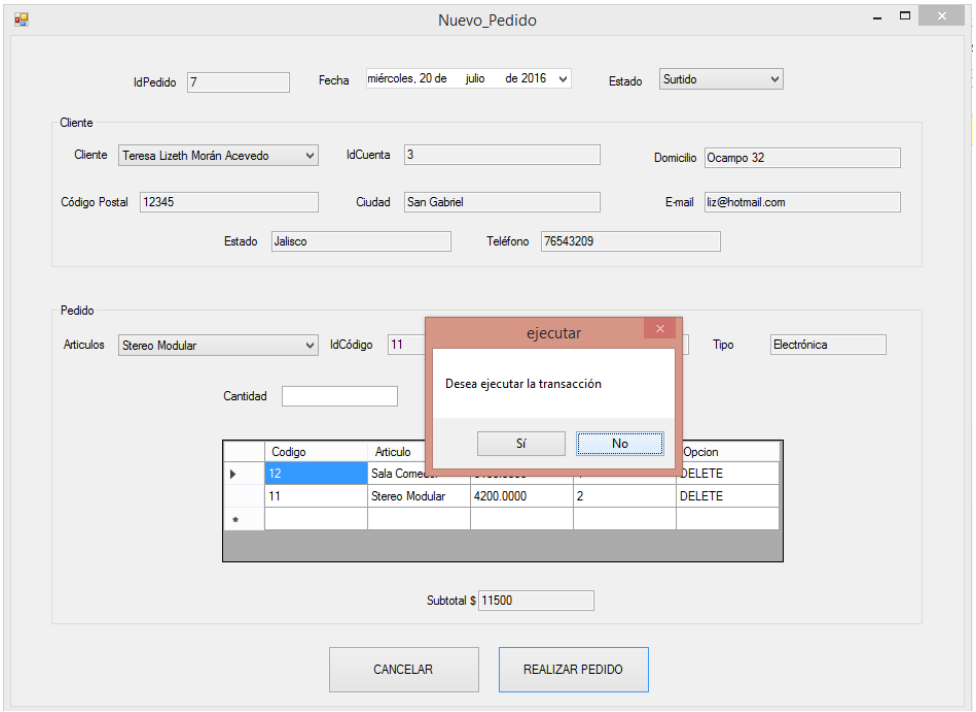
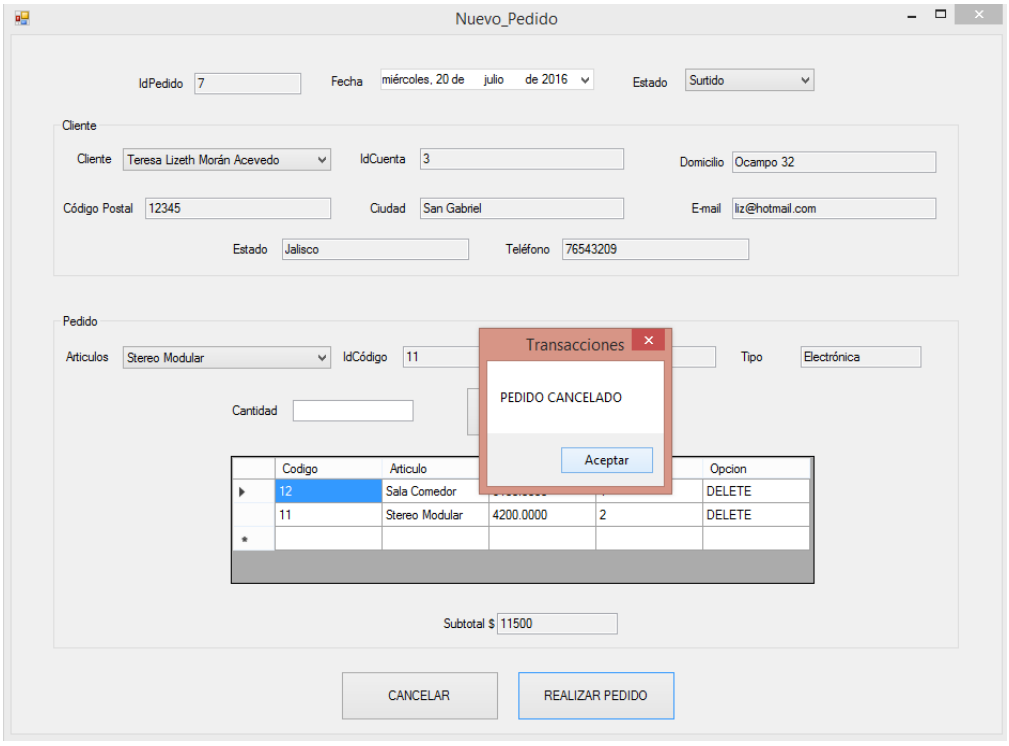
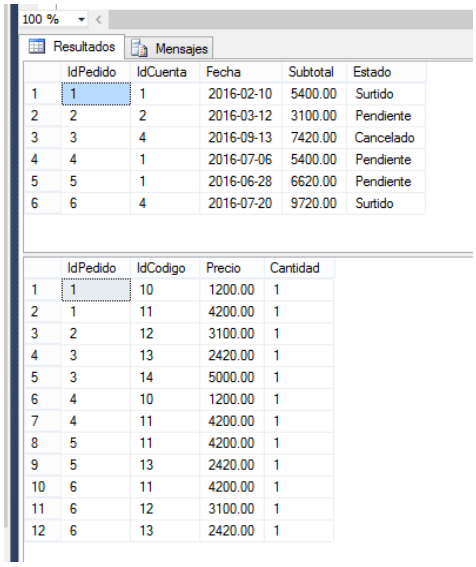


Figura 84. Ejecución de la aplicación cancelando la transacción (3).



La Figura 85 muestra la verificación de que no haya datos almacenados con registro IdPedido 7 (era el id del pedido que se canceló).

Figura 85. Pantalla en SQL Server sin datos de pedido cancelado.



	IdPedido	IdCuenta	Fecha	Subtotal	Estado
1	1	1	2016-02-10	5400.00	Surtido
2	2	2	2016-03-12	3100.00	Pendiente
3	3	4	2016-09-13	7420.00	Cancelado
4	4	1	2016-07-06	5400.00	Pendiente
5	5	1	2016-06-28	6620.00	Pendiente
6	6	4	2016-07-20	9720.00	Surtido

	IdPedido	IdCodigo	Precio	Cantidad
1	1	10	1200.00	1
2	1	11	4200.00	1
3	2	12	3100.00	1
4	3	13	2420.00	1
5	3	14	5000.00	1
6	4	10	1200.00	1
7	4	11	4200.00	1
8	5	11	4200.00	1
9	5	13	2420.00	1
10	6	11	4200.00	1
11	6	12	3100.00	1
12	6	13	2420.00	1

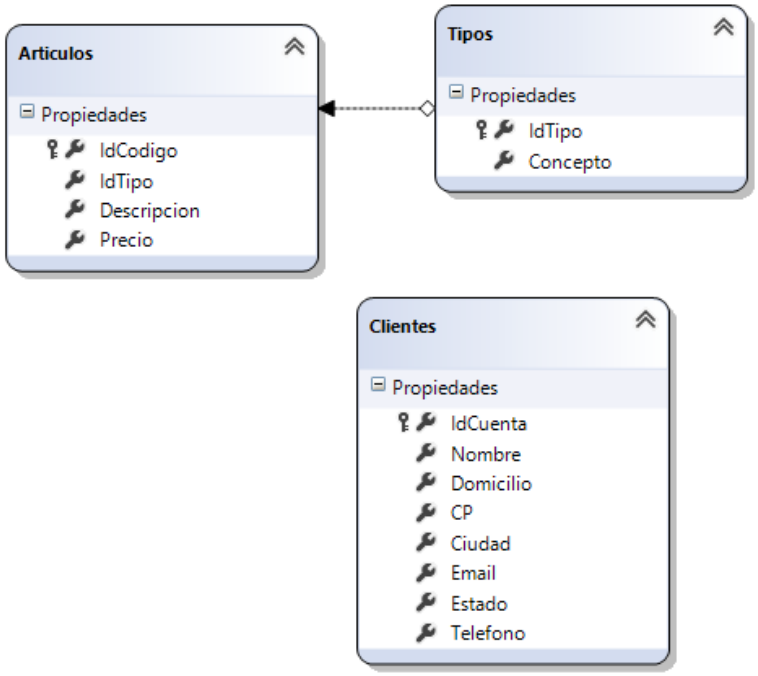
5.3 CONSULTA DE PEDIDOS POR CLIENTE Y AÑO CON LINQ TO SQL Y PROCEDIMIENTOS ALMACENADOS EN SQL SERVER.

En esta aplicación se enlistan los pedidos realizados por un determinado cliente y año, ambos seleccionados desde cuadros combinados, considerar que para llenar los nombres de los clientes lo tendrá que realizar por medio de LinQ y en el caso de los años implementar en SQL Server un procedimiento almacenado.

Paso 1: En el LinQ DataSistema se agregan las tablas que mantengan relación con la que se vaya a utilizar. Ver la Figura 86.

NOTA: En este caso la tabla Clientes no mantiene relación directa con otras tablas por lo que no es necesario agregar alguna al DataSistema.

Figura 86. Pantalla de LinQ DataSistema.



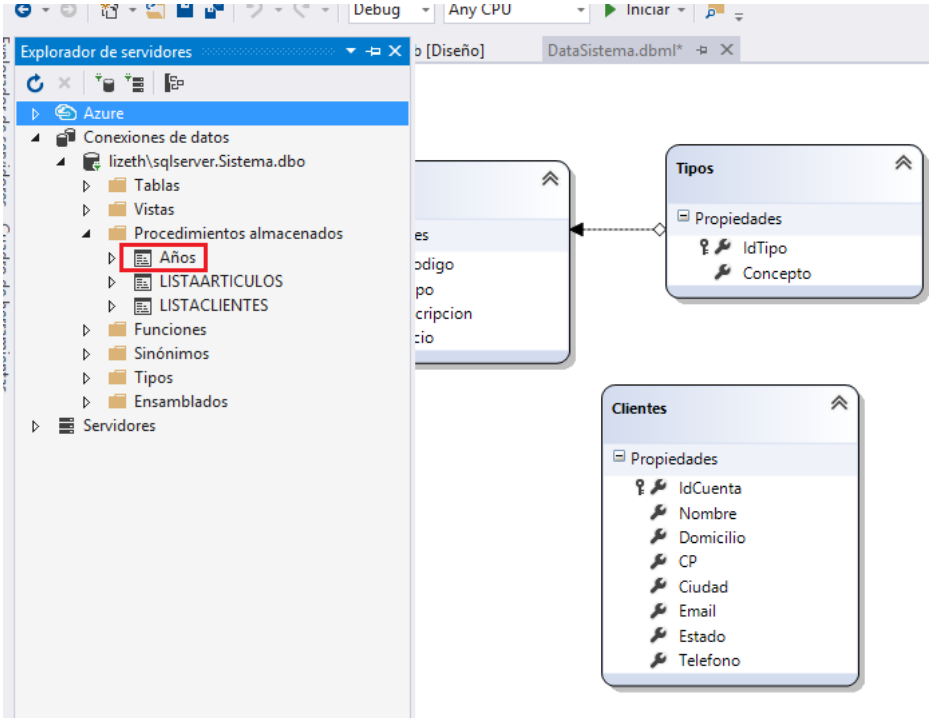
Paso 2: Ejecutar el siguiente script dentro del editor de SQL Server para poder enviar los años de los pedidos al cuadro combinado.

```
create procedure Años
As
        select YEAR(P.Fecha) As AÑO
        From Pedidos P
        Group By YEAR(P.Fecha)
Go
```

Paso 3: Arrastrar el procedimiento almacenado desde el explorador de servidores a LinQ como se muestra en la Figura 87.

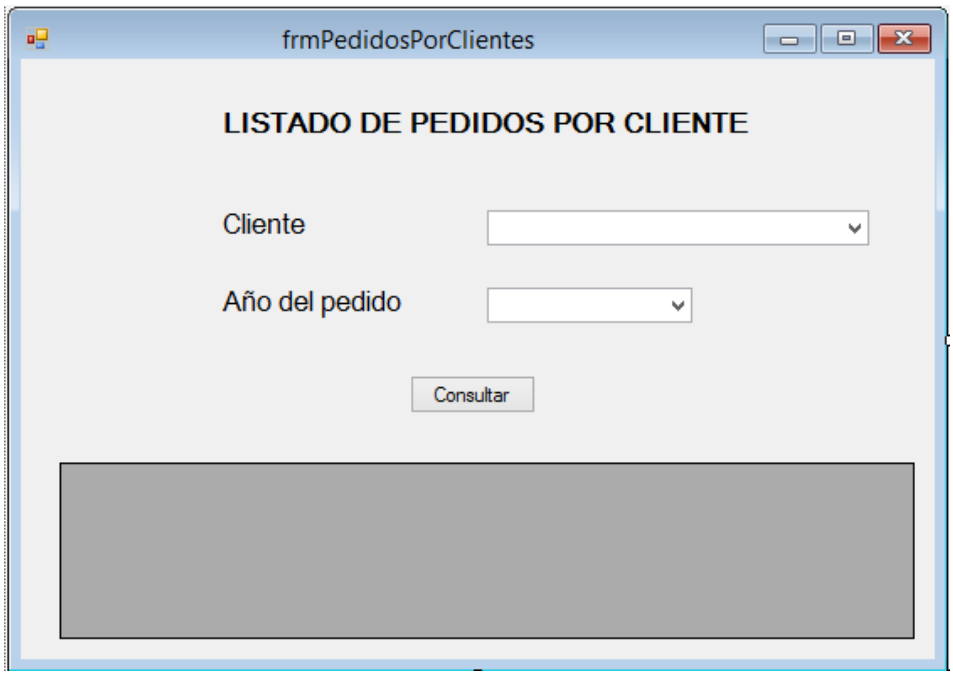
NOTA: Una vez que se ha arrastrado el procedimiento almacenado, en caso de que exista relación directa con otras tablas se notará la relación misma en el LinQ (dbml).

Figura 87. Pantalla de LinQ DataSistema con el procedimiento almacenado (izquierda).



Paso 4: Agregar un nuevo Windows Forms y los elementos necesarios para el listado de pedidos adecuado como se muestra en la Figura 88.

Figura 88. Interfaz para frmPedidosPorClientes.



Paso 5: Copiar el siguiente código en el formulario creado.

```
Public Class frmPedidosPorClientes

    Dim dcSist As New DataSistemaDataContext

    Private Sub frmPedidosPorClientes_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        Call llenaClientes()
        Call llenaAños()
    End Sub

    Sub llenaClientes()
        Dim cliente = From p In dcSist.Clientes
        cboCliente.DataSource = cliente
        cboCliente.DisplayMember = "Nombre"
        cboCliente.ValueMember = "IdCuenta"
    End Sub

    Sub llenaAños()
        cboAño.DataSource = dcSist.Años
        cboAño.DisplayMember = "AÑO"
        cboAño.ValueMember = "AÑO"
```

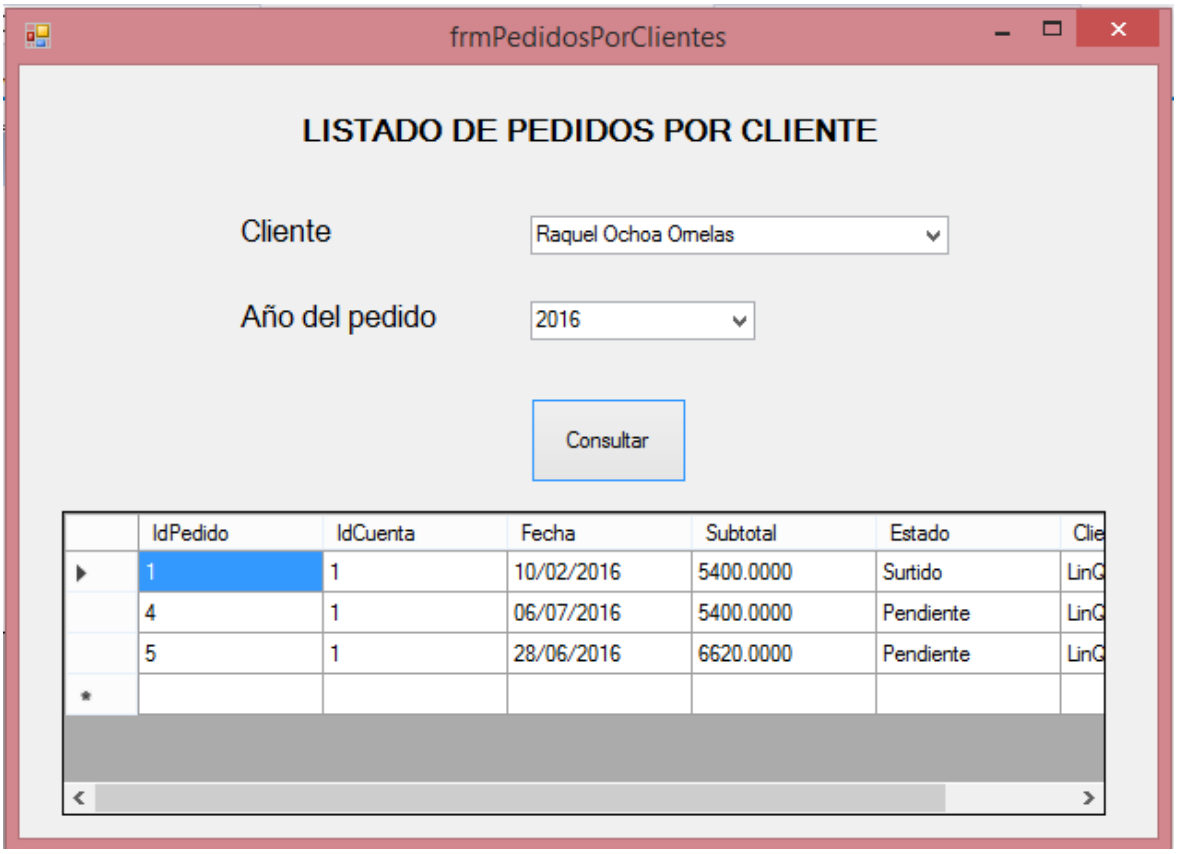
```
    End Sub

    Private Sub btnConsultar_Click(sender As Object, e As EventArgs) Handles
btnConsultar.Click
        Dim pedidos = From p In dcSist.Pedidos
                      Where p.IdCuenta = cboCliente.SelectedValue.ToString _
                      And p.Fecha > "1 / 1 / " & cboAño.Text _
                      And p.Fecha < " 31 / 12 / " & cboAño.Text
        dgvPedidos.DataSource = pedidos

    End Sub

End Class
```

Paso 6: Agregar el código para que desde el Form1 podamos abrir frmPedidosPorClientes.

```
Private Sub ListadoDePedidosPorClienteToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles ListadoDePedidosPorClienteToolStripMenuItem.Click
    frmPedidosPorClientes.Show()
End Sub
```

Paso 7: Ejecutar la aplicación y observar los datos presentados como lo muestra la Figura 89.

Figura 89. Pantalla en ejecución de la aplicación.

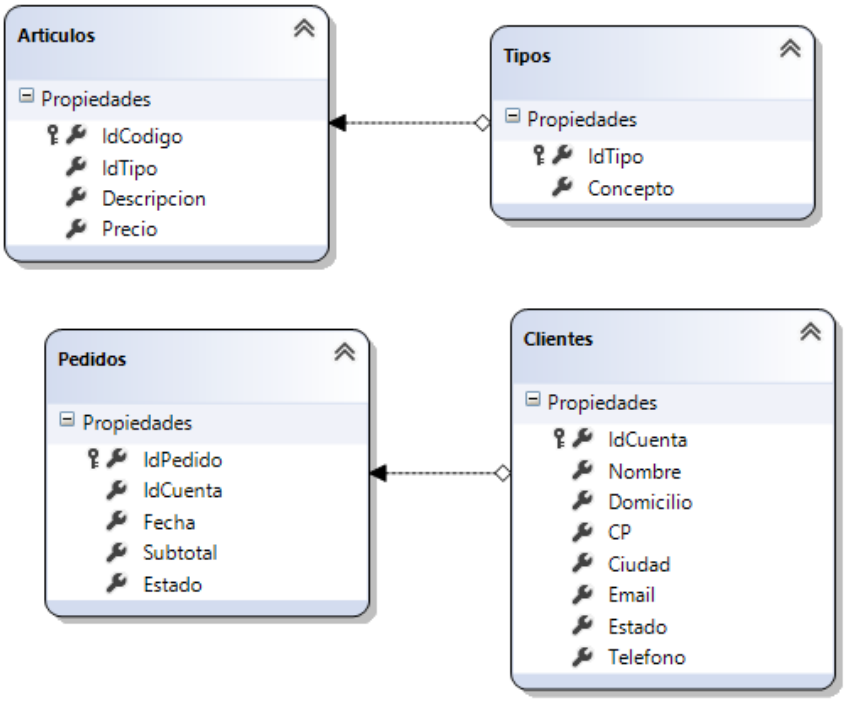


5.4 CONSULTA DE ARTICULOS SEGÚN SU TIPO CON LINQ TO SQL

En esta aplicación se permite mostrar los registros de la tabla Articulos según su Tipo. Se desarrollará haciendo uso de LinQ.

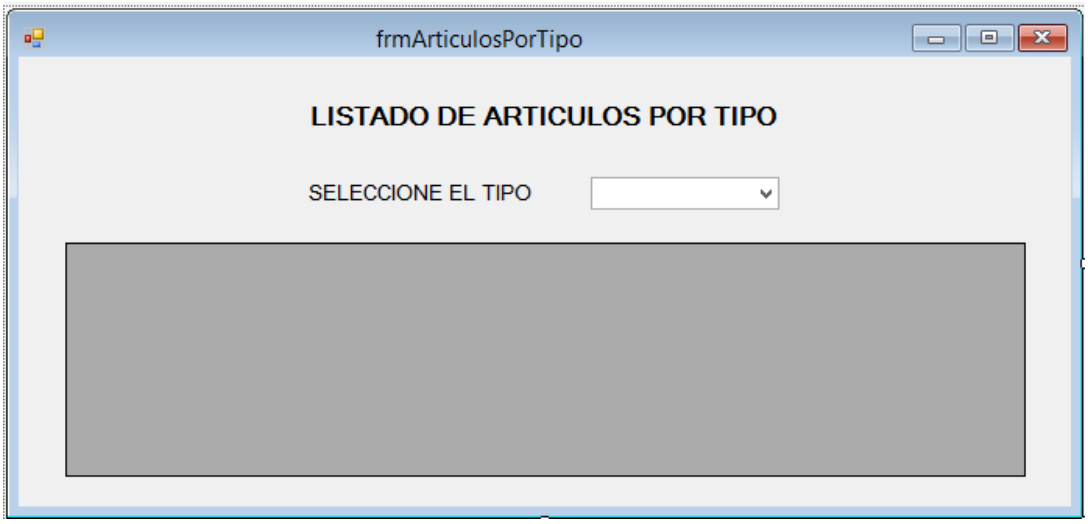
Paso 1: En el LinQ agregar la tabla Articulos, Tipos y Pedidos arrastrándolas desde el explorador de soluciones. Quedando como lo muestra la Figura 90.

Figura 90. Pantalla de LinQ DataSistema.



Paso 2: Agregar un nuevo Windows Form al proyecto LinQTo y agregar los elementos necesarios para la consulta como lo muestra la Figura 91.

Figura 91. Interfaz para frmArticulosPorTipo.



Paso 3: Agregar el código siguiente a frmArticulosPorTipo.

```
Public Class frmArticulosPorTipo

    Dim daAge As New DataSistemaDataContext

    Private Sub frmArticulosPorTipo_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        Call llenaTipos()
    End Sub

    Sub llenaTipos()
        Dim Tipos = From t In daAge.Tipos
        Dim Arti = From a In daAge.Articulos

        cboTipo.DataSource = Tipos
        cboTipo.DisplayMember = "Concepto"
        cboTipo.ValueMember = "Concepto"
    End Sub

    Private Sub cboTipo_SelectedIndexChanged(sender As Object, e As EventArgs)
Handles cboTipo.SelectedIndexChanged

        Dim valorCombo = cboTipo.SelectedValue.ToString

        Dim articulos = From t In daAge.Tipos Join a In daAge.Articulos On t.IdTipo Equals
a.IdTipo
                        Where t.Concepto = valorCombo Select a.IdCodigo, a.IdTipo,
a.Descripcion, a.Precio
```

```
        dgArticulos.DataSource = articulos

    End Sub
End Class
```

Paso 4: Agregar el código para que desde el Form1se pueda invocar a frmArticulosPorTipo.

```
Private Sub ListadoDeArticulosPorTipoToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles ListadoDeArticulosPorTipoToolStripMenuItem.Click
        frmArticulosPorTipo.Show()
    End Sub
```

Paso 5: Ejecutar la aplicación y obtener los resultados como los muestra la Figura 92.

Figura 92. Pantalla en ejecución de la aplicación.

5.5 PROGRAMACIÓN EN N-CAPAS

En esta aplicación se permitirá hacer uso de la arquitectura en n-capas, específicamente presentación, datos, lógica y entidad, para administrar los datos de pedidos.

Paso 1: Implementar los siguientes scripts dentro del editor de SQL Server (considerando que la base de datos ya se encuentra en el servidor).

```
Use Sistema
Go

--Procedimiento almacenado que permite mostrar un listado de los pedidos
IF OBJECT_ID('LISTADOPEDIDOS') IS NOT NULL
	DROP PROC LISTADOPEDIDOS
GO

CREATE PROC LISTADOPEDIDOS
AS
BEGIN
	SELECT Pedidos.IdPedido AS idPedido,
		Pedidos.IdCuenta AS idCuenta,
		Pedidos.Fecha AS fecha,
		Pedidos.Subtotal AS subtotal,
		Pedidos.Estado AS estado
		FROM Pedidos
		JOIN Clientes ON Pedidos.IdCuenta = Clientes.IdCuenta
END
GO

--Procedimiento almacenado para actualizar el estado del pedido
Use Sistema
IF OBJECT_ID('ACTUALIZAESTADO') IS NOT NULL
	DROP PROC ACTUALIZAESTADO
GO

CREATE PROC ACTUALIZAESTADO(
	@idPedido int,
	@estado varchar(10)
)
AS
	UPDATE Pedidos SET Estado = @estado WHERE IdPedido = @idPedido
GO


--Procedimiento almacenado que permitirá consultar los id de los pedidos
--para los combios
IF OBJECT_ID('CONSULTARIDPEDIDO') IS NOT NULL
	DROP PROC CONSULTARIDPEDIDO
GO

CREATE PROC CONSULTARIDPEDIDO
AS
	SELECT IdPedido from pedidos ORDER BY IdPedido ASC
GO
```

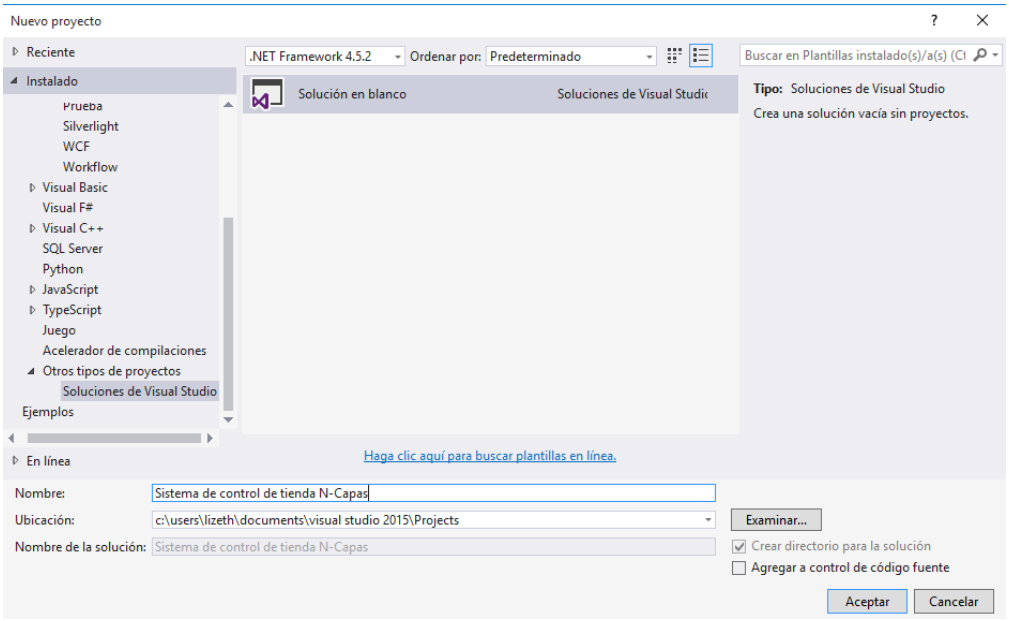
```
--Procedimiento almacenado para consultar el detalle del pedido
IF OBJECT_ID('CONSULTARDETALLEPEDIDO') IS NOT NULL
	DROP PROC CONSULTARDETALLEPEDIDO
GO

CREATE PROC CONSULTARDETALLEPEDIDO(@IdPedido int)
AS
	SELECT DetallePedido.IdCodigo, Articulos.Descripcion, DetallePedido.Precio,
DetallePedido.Cantidad
	FROM DetallePedido Join Articulos On DetallePedido.IdCodigo =
Articulos.IdCodigo WHERE IdPedido = @IdPedido
GO

--Procedimiento almacenado para consultar los datos del pedido

If OBJECT_ID('CONSULTARPEDIDO')IS NOT NULL
	DROP PROC CONSULTARPEDIDO
GO

CREATE PROC CONSULTARPEDIDO(@idPedido int)
AS
	SELECT Pedidos.IdPedido, Pedidos.IdCuenta, Pedidos.Estado,
Pedidos.Fecha,Pedidos.Subtotal,Clientes.Nombre FROM Pedidos Join Clientes
	On Pedidos.IdCuenta = Clientes.IdCuenta WHERE IdPedido = @IdPedido
GO
```

Paso 2: Crear en Visual Basic Community 2015 el proyecto completo de la aplicación en:

Archivo > Nuevo Proyecto > Seleccionar "Otros tipos de proyectos" > Soluciones de Visual Studio > asignar el nombre "**Sistema de control de tienda n-capas**", tal como se muestra en la Figura 93.

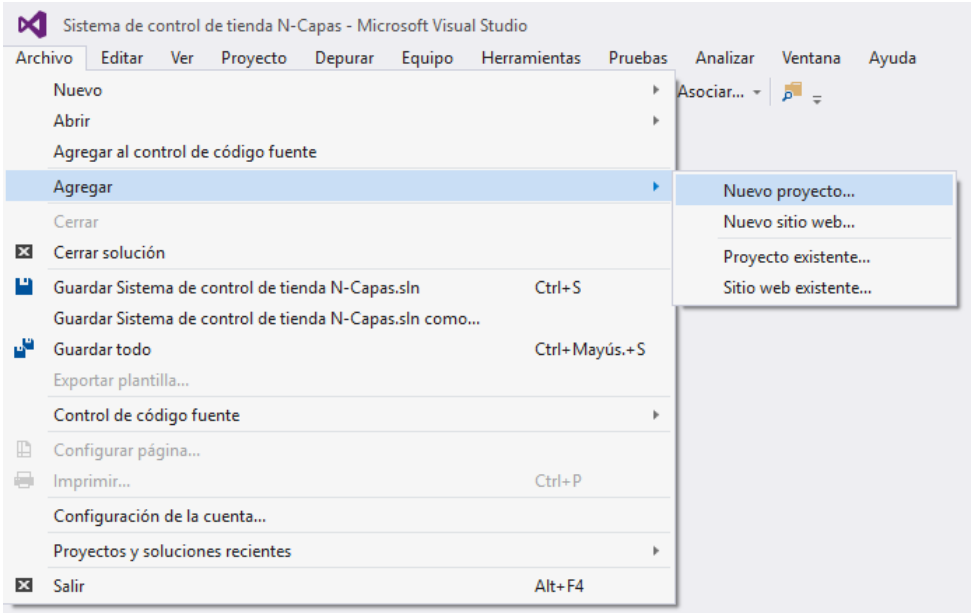
Nota: La ruta que se seleccione será la ubicación en la que se almacenará el proyecto.

Figura 93. Creación de nuevo proyecto para arquitectura n-capas.



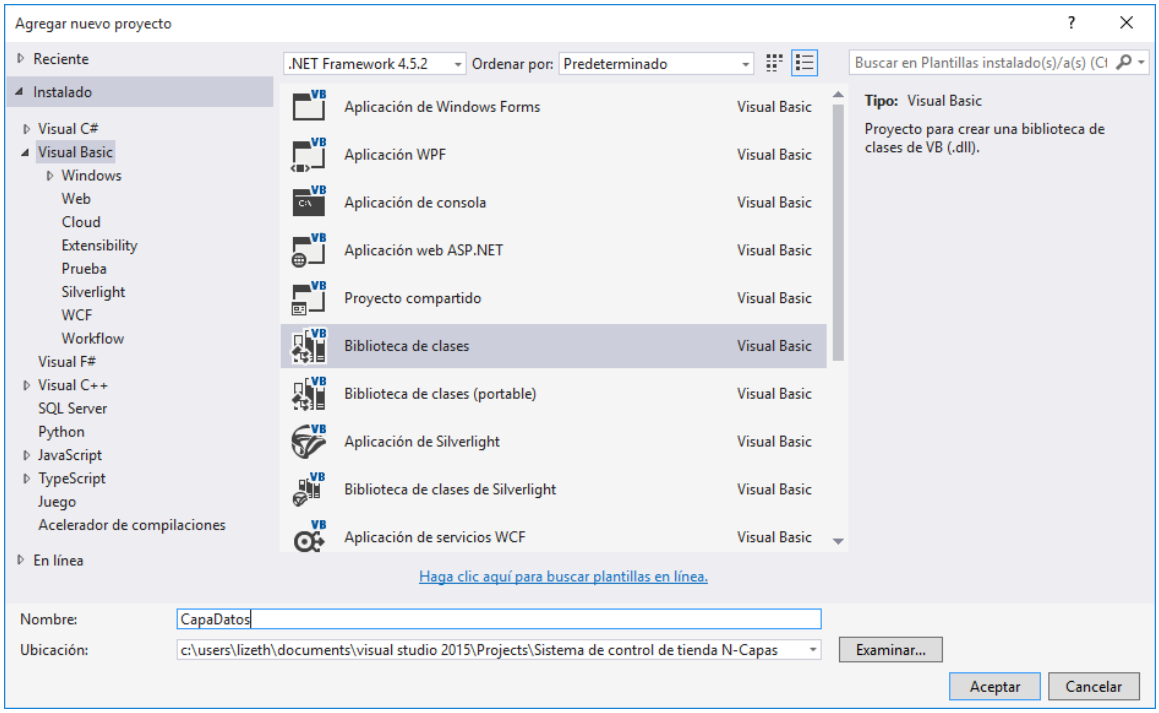
Paso 3: A la solución anterior agregar un nuevo proyecto en Archivo > Agregar > Nuevo Proyecto como se muestra en la Figura 94.

Figura 94. Nuevo Proyecto.



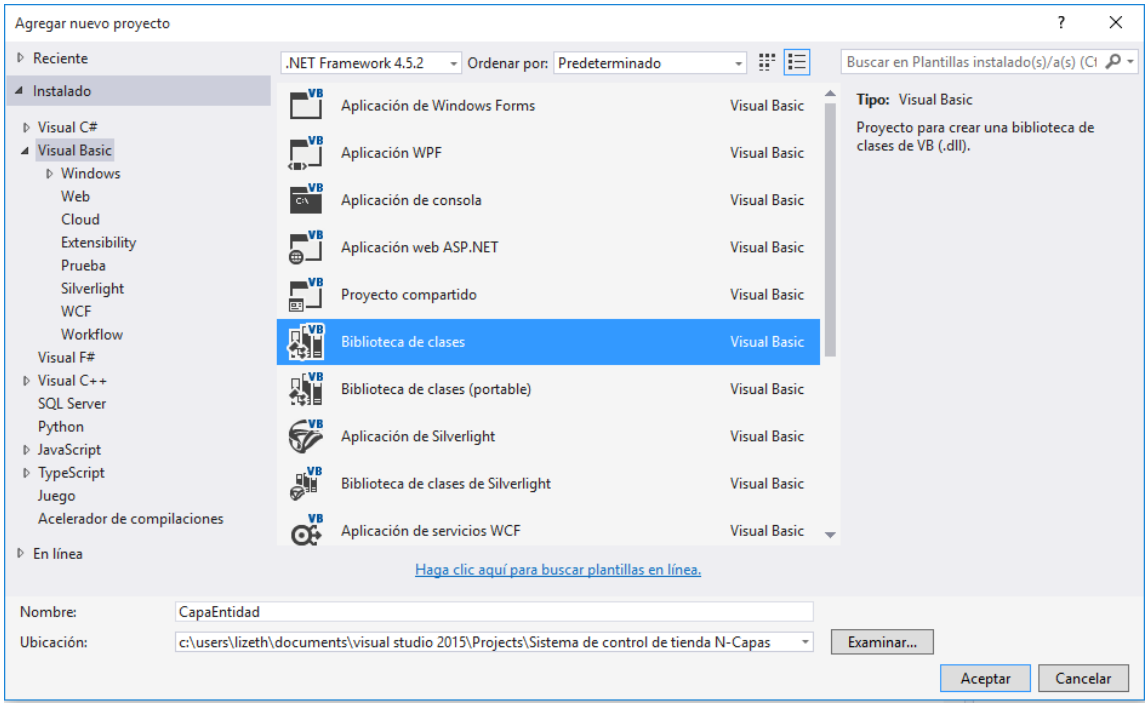
Paso 4: Seleccionar Visual Basic > Biblioteca de Clase y asignar el nombre de **"Capa de datos"** como lo muestra la Figura 95.

Figura 95. Nueva Biblioteca de clases como CapaDatos.



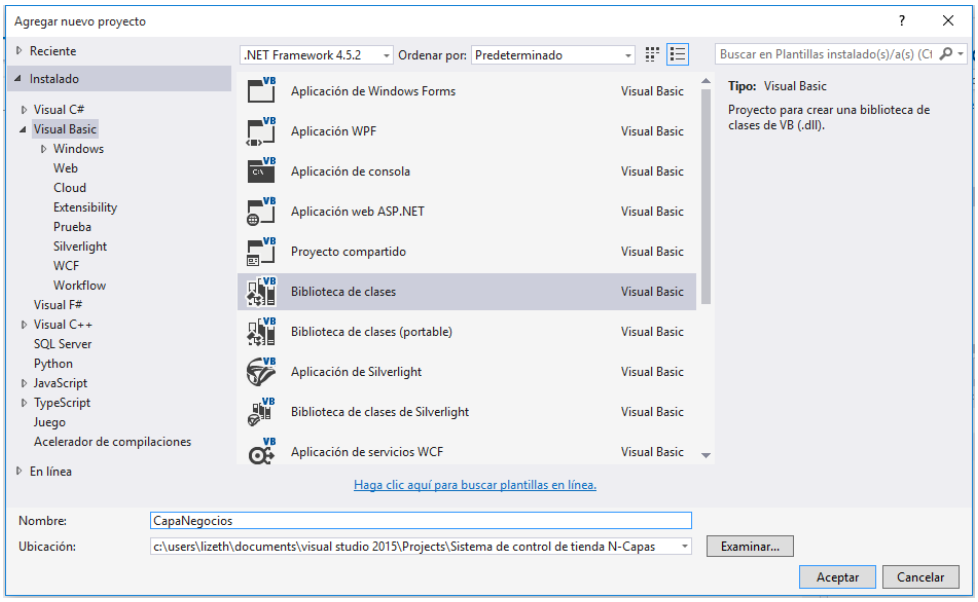
Paso 5: Nuevamente seleccionar Archivo > Agregar > Nuevo Proyecto > Visual Basic > Biblioteca de Clases > asignar el nombre "**CapaEntidad"** como lo muestra la Figura 96.

Figura 96. Nueva Biblioteca de clases como CapaEntidad.



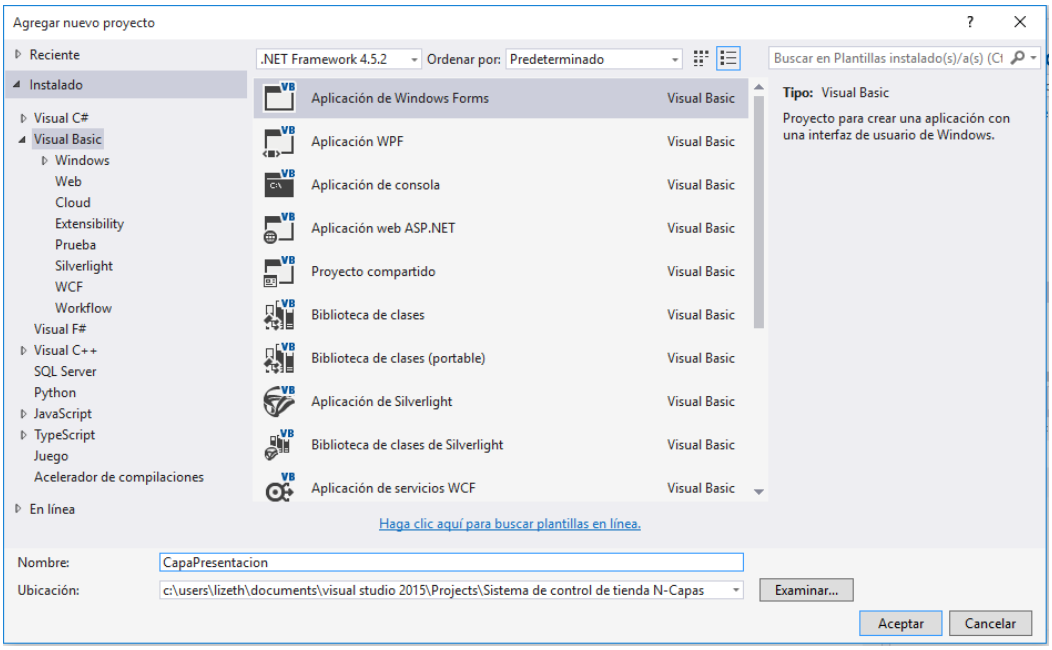
Paso 6: Nuevamente seleccionar Archivo > Agregar > Nuevo Proyecto > Visual Basic > Biblioteca de Clases > asignar el nombre "**CapaNegocios"** como lo muestra la Figura 97.

Figura 97. Nueva Biblioteca de clases como CapaNegocios.



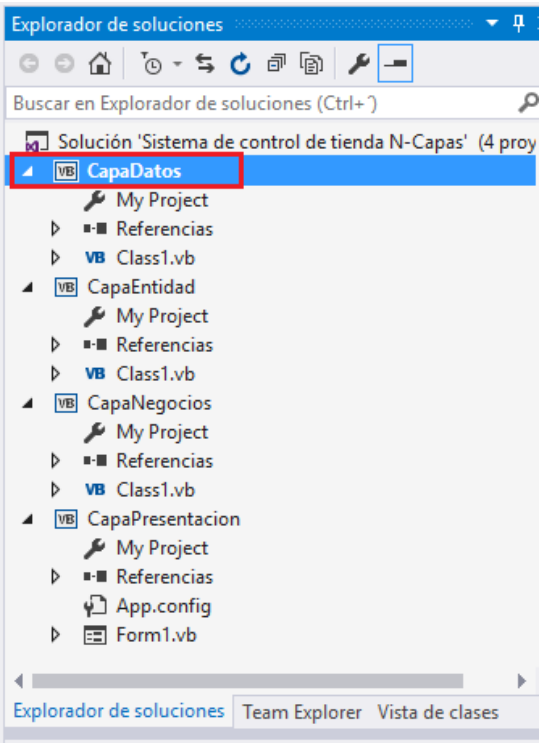
Paso 7: Finalmente, agregar la última capa en Archivo > Agregar > Nuevo Proyecto > Visual Basic > Aplicación de Windows Forms > asignar el nombre "**CapaPresentacion"** como lo muestra la Figura 98.

Figura 98. Nueva Aplicación de Windows Forms como CapaPresentación.



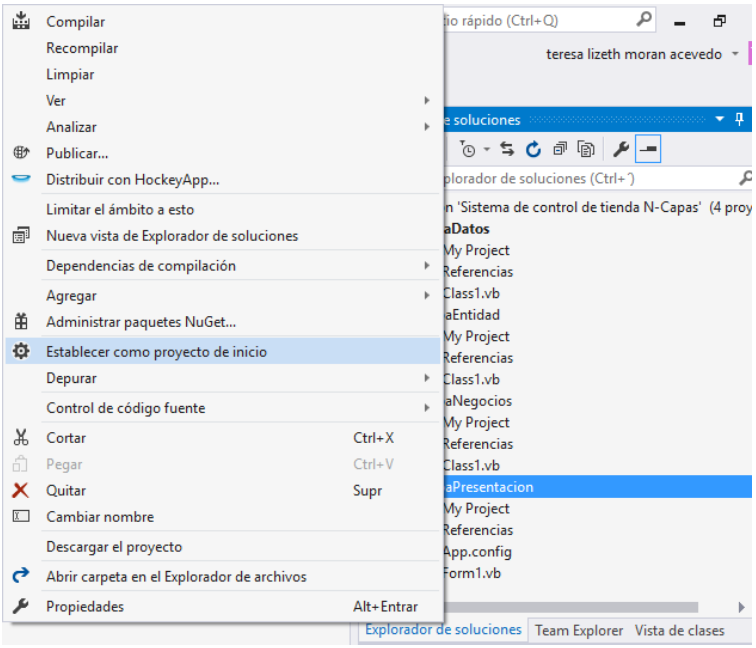
La capa de datos fue la primera que se creó y es actualmente la que tiene el proyecto de inicio, es decir, al ejecutar la aplicación se iniciará con la capa de datos y ese no es el trabajo de dicha capa, por lo que se debe modificar el proyecto de inicio. Se reconoce dicha característica porque todo el proyecto de inicio aparece en negrita, como lo muestra la Figura 99.

Figura 99. Visualización de capa inicial del proyecto.



Paso 8: Para cambiar el proyecto de inicio presionar clic derecho sobre la **CapaPresentacion** y seleccionar “Establecer como proyecto de inicio”, tal y como se muestra en la Figura 100.

Figura 100. Visualización de capa inicial del proyecto.

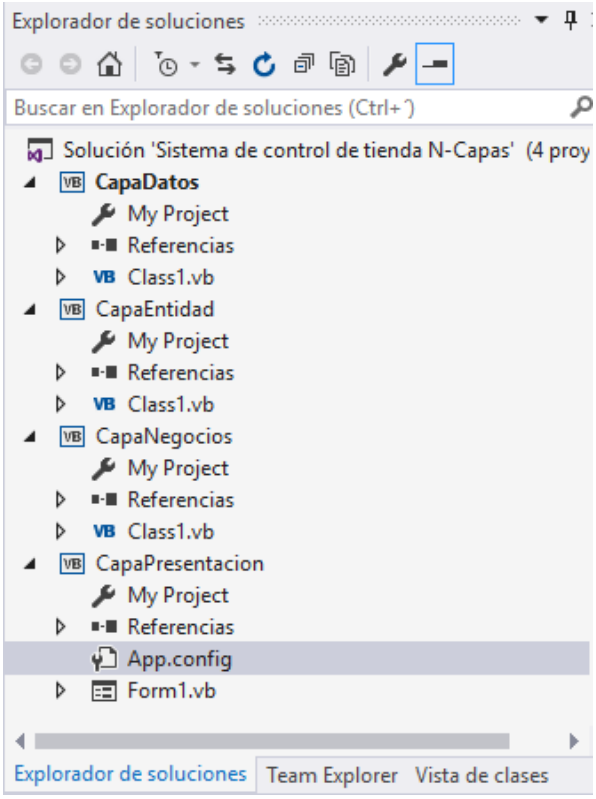


Paso 9: Configurar el app.config de la capa de presentación con el siguiente código:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <startup>
      <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.2" />
   </startup>
   <connectionStrings>
    <add name="cn"
        connectionString ="Server=.;Database=Sistema;Integrated security=SSPI"
    />
   </connectionStrings>
</configuration>
```

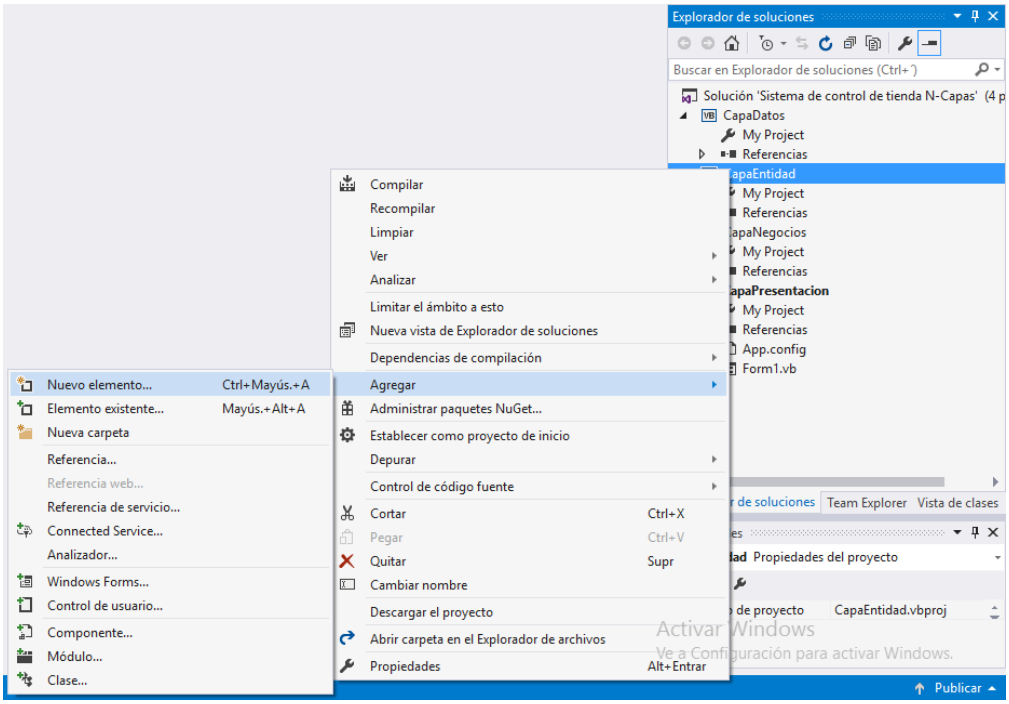
Paso 10: Antes de implementar las clases que comprenderán los elementos de las capas, se deben eliminar las capas creadas en la capa de datos, capa entidad y capa de negocios (como se presenta en la Figura 101). Los cuales llevan por nombre “Class1.vb”.

Figura 101. Antes de la eliminación de la capa de datos, entidad y de negocios.



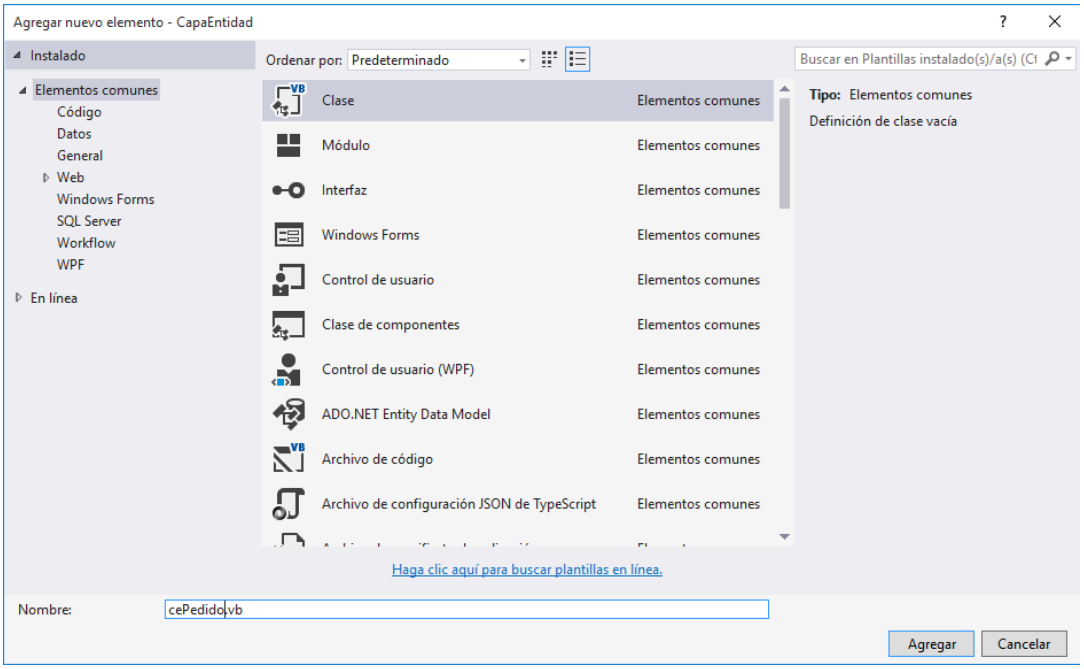
Paso 11: Agregar la clase cePedidos en la capa entidad, ésta tendrá la misión de especificar todos los elementos que componen la clase pedido. Para esto en el Explorador de Soluciones > clic derecho capaEntidad > Agregar > Nuevo elemento, tal como se muestra en la Figura 102.

Figura 102. Agregar nuevo elemento a la capa entidad.



Agregar la clase “cePedido” > Aceptar (Véase la Figura 103).

Figura 103. Nueva clase en capa entidad.



Paso 12: Agregar el siguiente código a la clase cePedido anteriormente creada.

```
Public Class cePedido
    Private _idPedido As Integer
    Private _idCuenta As String
    Private _fecha As String
    Private _subtotal As Double
    Private _estado As String

    Public Property idPedido As Integer
        Get
            Return _idPedido
        End Get
        Set(value As Integer)
            _idPedido = value
        End Set
    End Property

    Public Property idCuenta As String
        Get
            Return _idCuenta
        End Get
        Set(value As String)
            _idCuenta = value
        End Set
    End Property

    Public Property fecha As String
        Get
            Return _idCuenta
        End Get
        Set(value As String)
            _idCuenta = value
        End Set
    End Property

    Private Property subtotal As Double
        Get
            Return _subtotal
        End Get
        Set(value As Double)
            _subtotal = value
        End Set
    End Property

    Public Property estado As String
        Get
```

```
        Return _estado
    End Get
    Set(value As String)
        _estado = value
    End Set
  End Property

End Class
```

Paso 13: Agregar a la capa de datos la referencia al namespace **System.Configuration** como se muestra en la Figura 104 y 105.

Figura 104. Nueva referencia.

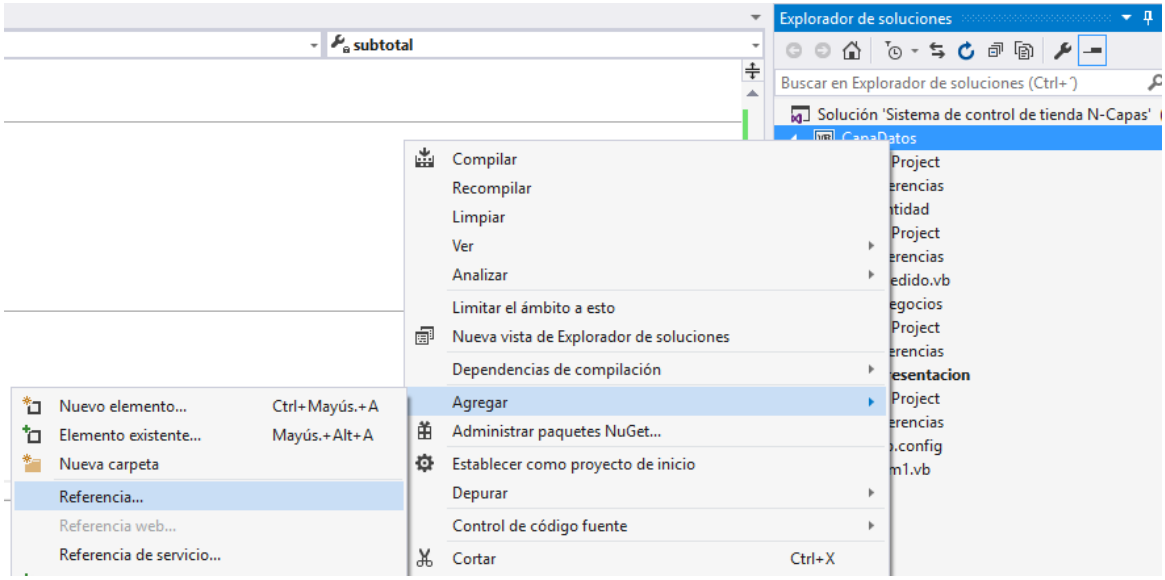
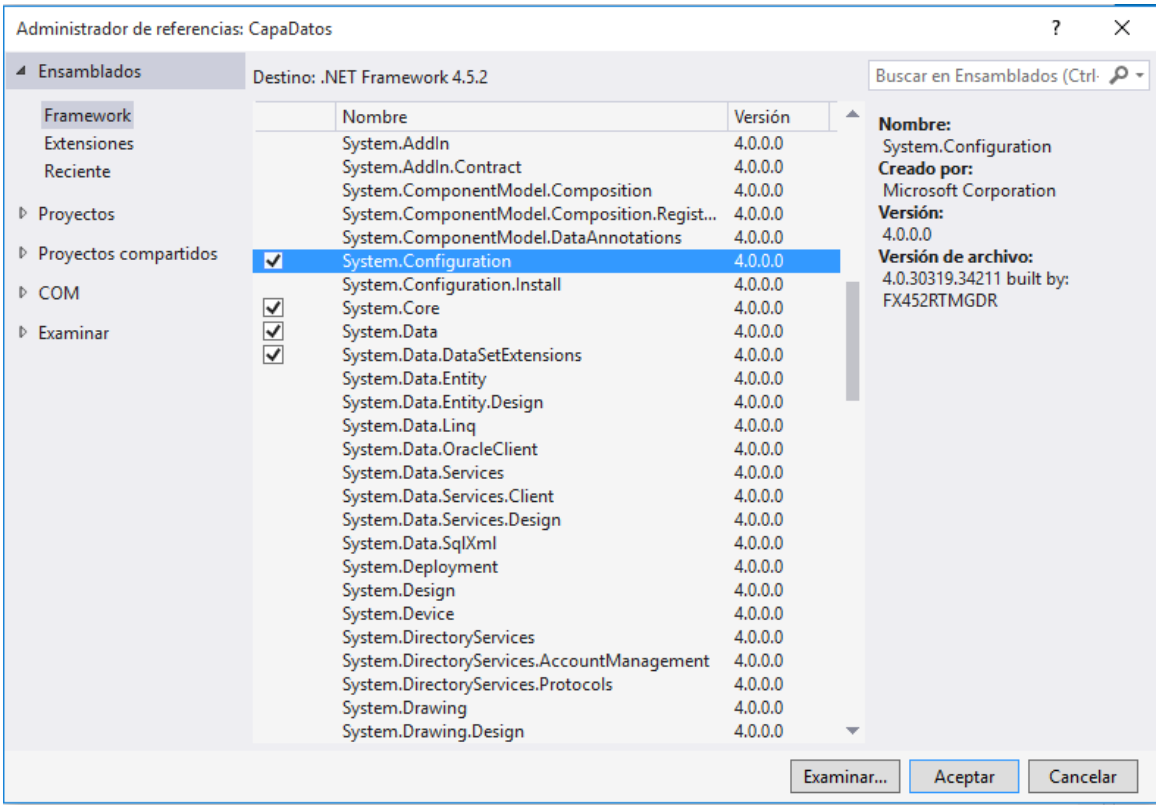


Figura 105. Selección de la referencia “System.Configuration”.



Paso 14: Agregar a la capa de datos la referencia de la Capa Entidad, para lograrlo presionar clic derecho sobre la capa de datos > Solucion > Proyecto > Activar Check de la CapaEntidad, tal y como lo muestra la Figura 106 y 107.

Figura 106. Agregar nueva referencia a la capa de datos.

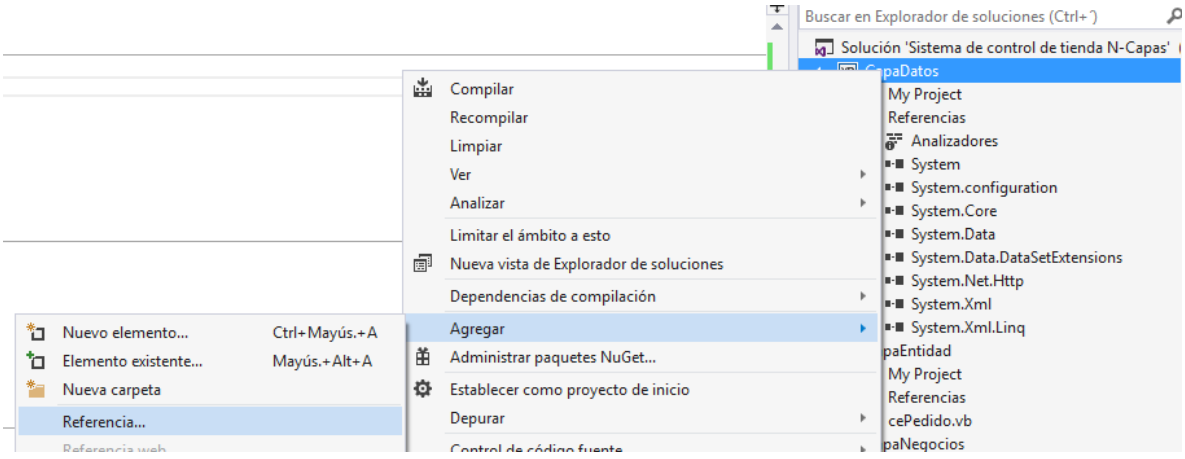
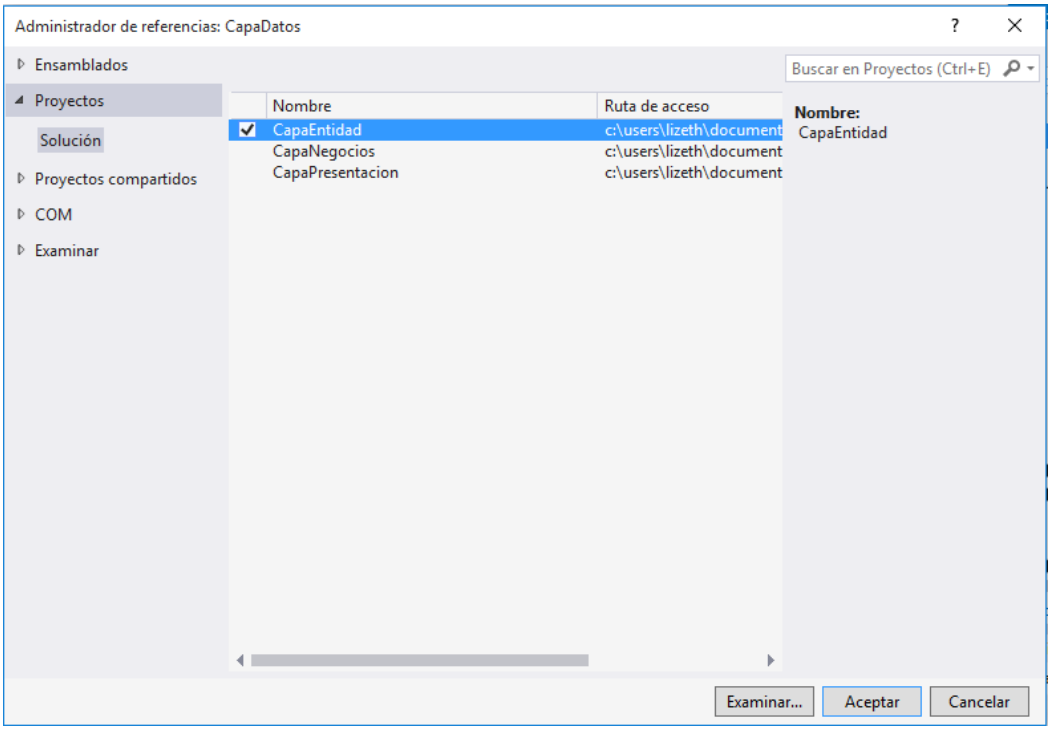


Figura 107. Selección de la referencia de capa entidad.



Paso 15: Agregar dos clases a la CapaDatos: la primera llamada “Conexion” que tendrá la misión de realizar las transacciones necesarias para la conexión a la base de datos “Sistema”, la segunda llamada “PedidoDAO” que tendrá por misión almacenar las funciones y procedimientos propios de la administración de los registros para pedidos. Basarse en las Figuras 108 y 109.

Figura 108. Agregar nuevo elemento (clase Conexion).

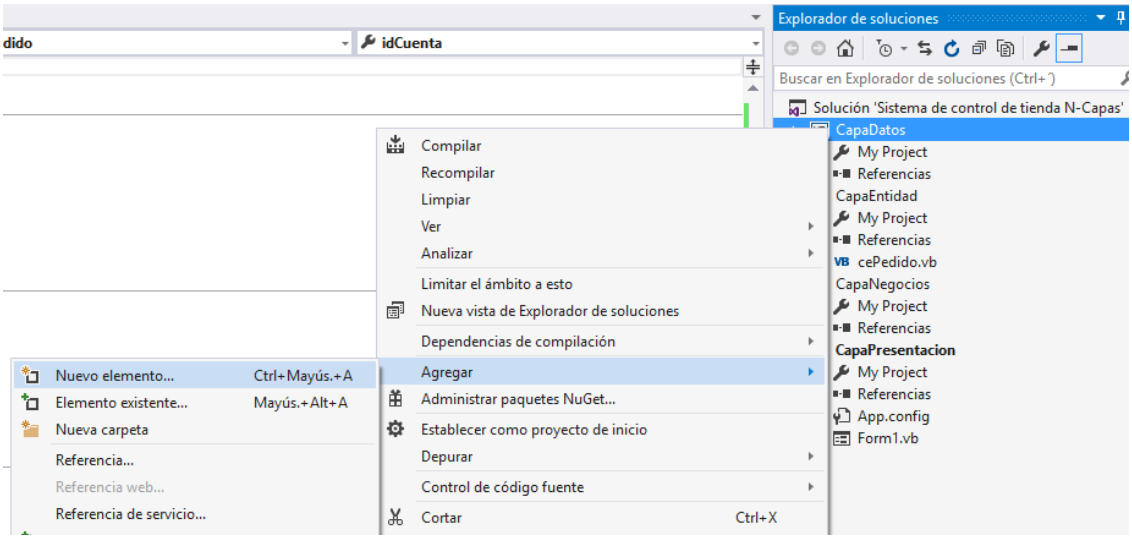
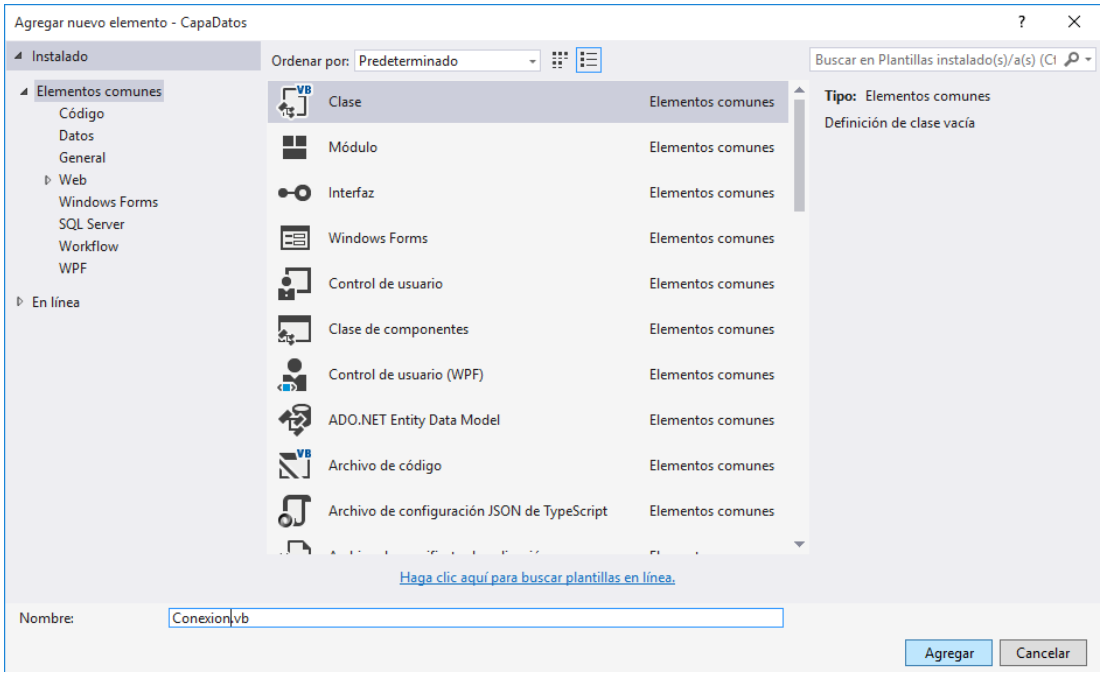


Figura 109. Agregar nuevo elemento (clase PedidoDAO).



Paso 16: Dentro de la clase Conexión, agregar el siguiente código:

```
Imports System.Data.SqlClient
Imports System.Configuration
Public Class Conexion
    Dim conexion As SqlConnection
    Public Function conectar() As SqlConnection
        conexion = New
SqlConnection(ConfigurationManager.ConnectionStrings("cn").ConnectionString)
        Return conexion
    End Function
End Class
```

Paso 17: Dentro de la clase PedidoDAO agregar el siguiente código.

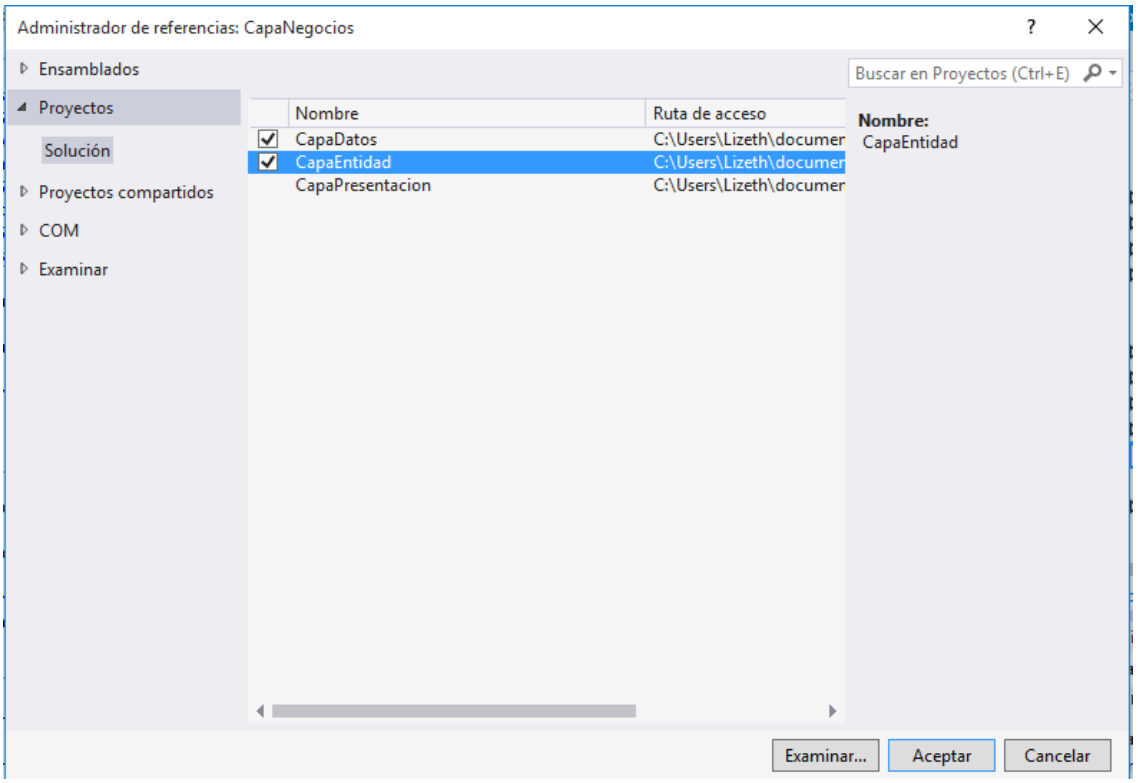
```
Imports System.Data.SqlClient
Imports CapaEntidad
Public Class PedidoDAO
    Dim conexion As New Conexion
    Dim cn As SqlConnection
    Dim da As SqlDataAdapter
    Public Function LISTADOPEDIDOS() As DataSet
        Dim ds As New DataSet
        cn = conexion.conectar
        da = New SqlDataAdapter("LISTADOPEDIDOS", cn)
        da.SelectCommand.CommandType = CommandType.StoredProcedure
        da.Fill(ds, "Pedidos")
```

```
    da.Dispose()
    cn.Dispose()
    Return ds
End Function
Public Function CONSULTARDETALLEPEDIDO(ByVal id As Integer) As DataSet
    Using data As New DataSet
        cn = conexion.conectar
        da = New SqlDataAdapter("CONSULTARDETALLEPEDIDO", cn)
        da.SelectCommand.CommandType = CommandType.StoredProcedure
        Dim pDetalle = New SqlParameter("@IdPedido", SqlDbType.Int)
        pDetalle.Direction = ParameterDirection.Input
        pDetalle.Value = id
        da.SelectCommand.Parameters.Add(pDetalle)
        da.Fill(data, "DETALLEPEDIDO")
        Return data
    End Using
    da.Dispose()
    cn.Dispose()
End Function
Public Function CONSULTARIDPEDIDO() As DataSet
    Dim ds As New DataSet
    cn = conexion.conectar
    da = New SqlDataAdapter("CONSULTARIDPEDIDO", cn)
    da.SelectCommand.CommandType = CommandType.StoredProcedure
    da.Fill(ds, "Pedidos")
    da.Dispose()
    cn.Dispose()
    Return ds
End Function
Public Function CONSULTARPEDIDO(ByVal id As Integer) As DataSet
    Using data As New DataSet
        cn = conexion.conectar
        da = New SqlDataAdapter("CONSULTARPEDIDO", cn)
        da.SelectCommand.CommandType = CommandType.StoredProcedure
        Dim pPedido = New SqlParameter("@IdPedido", SqlDbType.Int)
        pPedido.Direction = ParameterDirection.Input
        pPedido.Value = id
        da.SelectCommand.Parameters.Add(pPedido)
        da.Fill(data, "Pedidos")
        Return data
    End Using
    da.Dispose()
    cn.Dispose()
End Function

Public Sub ACTUALIZAESTADO(ByVal id As Integer, ByVal edo As String)
    cn = conexion.conectar
```

```
        Try
            da = New SqlDataAdapter("ACTUALIZAESTADO", cn)
            da.SelectCommand.CommandType = CommandType.StoredProcedure
            With da.SelectCommand.Parameters
                .Add("@idPedido", SqlDbType.Int).Value = id
                .Add("@estado", SqlDbType.VarChar).Value = edo
            End With
            cn.Open()
            da.SelectCommand.ExecuteNonQuery()
            MsgBox("ESTADO DE PEDIDO ACTUALIZADO")
        Catch ex As Exception
            MsgBox("ERROR AL ACTUALIZAR EL ESTADO DEL PEDIDO" &
ex.Message)
        Finally
            da.Dispose()
            cn.Dispose()
        End Try
    End Sub

End Class
```

Paso 18: Agregar la referencia CapaDatos y CapaEntidad a la **Capa de Negocios** como lo muestra la Figura 110.

Figura 110. Nuevas referencias a CapaNegocios.

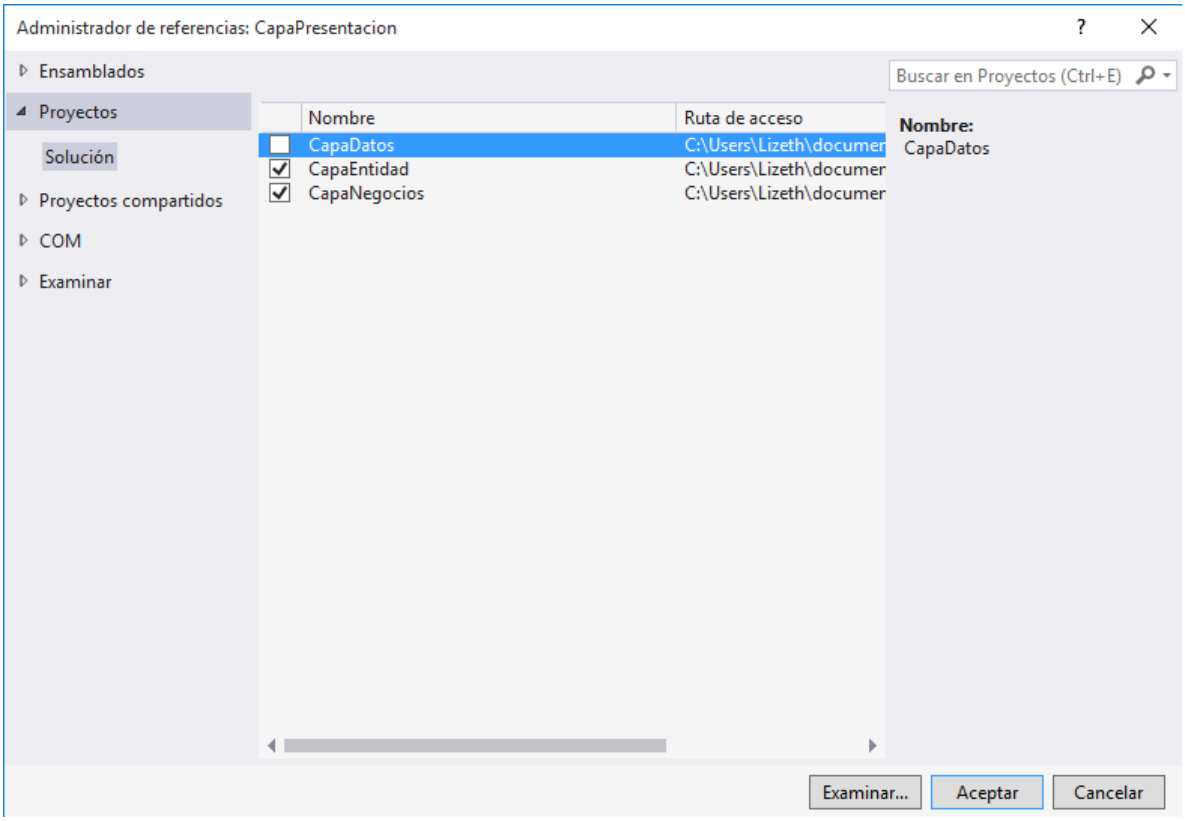


Paso 19: Agregar una clase a la Capa de Negocios y asignar el nombre de “PedidoCN” y colocar el siguiente código:

```
Imports CapaDatos
Imports CapaEntidad
Public Class PedidoCN
    Dim Pedido As New PedidoDAO
    Dim Cliente As New ClienteDAO

    Public Function LISTADOPEDIDOS() As DataSet
        Return pedido.LISTADOPEDIDOS
    End Function
    Public Function CONSULTARIDPEDIDO() As DataSet
        Return Pedido.CONSULTARIDPEDIDO
    End Function

    Public Function CONSULTARDETALLEPEDIDO(ByVal id As Integer) As DataSet
        Return Pedido.CONSULTARDETALLEPEDIDO(id)
    End Function

    Public Function CONSULTARPEDIDO(ByVal id As Integer) As DataSet
        Return Pedido.CONSULTARPEDIDO(id)
    End Function
    Public Sub ACTUALIZAPEDIDO(ByVal id As Integer, ByVal edo As String)
        Pedido.ACTUALIZAESTADO(id, edo)
    End Sub

End Class
```

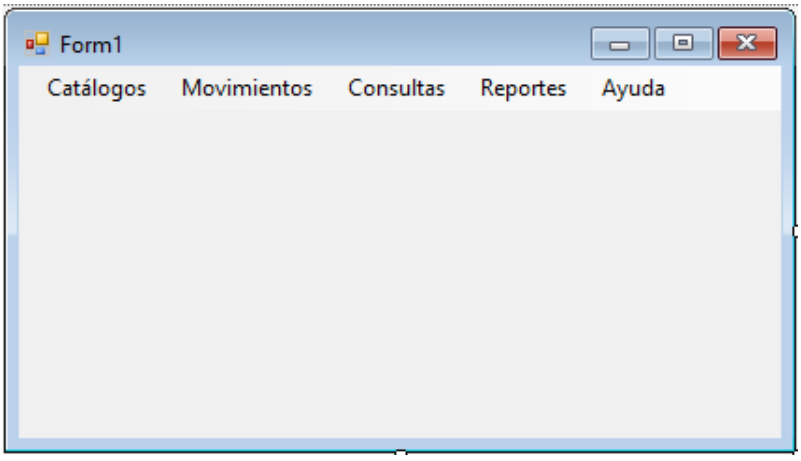
Paso 20: Agregar a la Capa de Presentación la referencia de la Capa de Negocios y la Capa Entidad como lo muestra la Figura 111.

Figura 111. Nuevas referencias a la CapaPresentación.



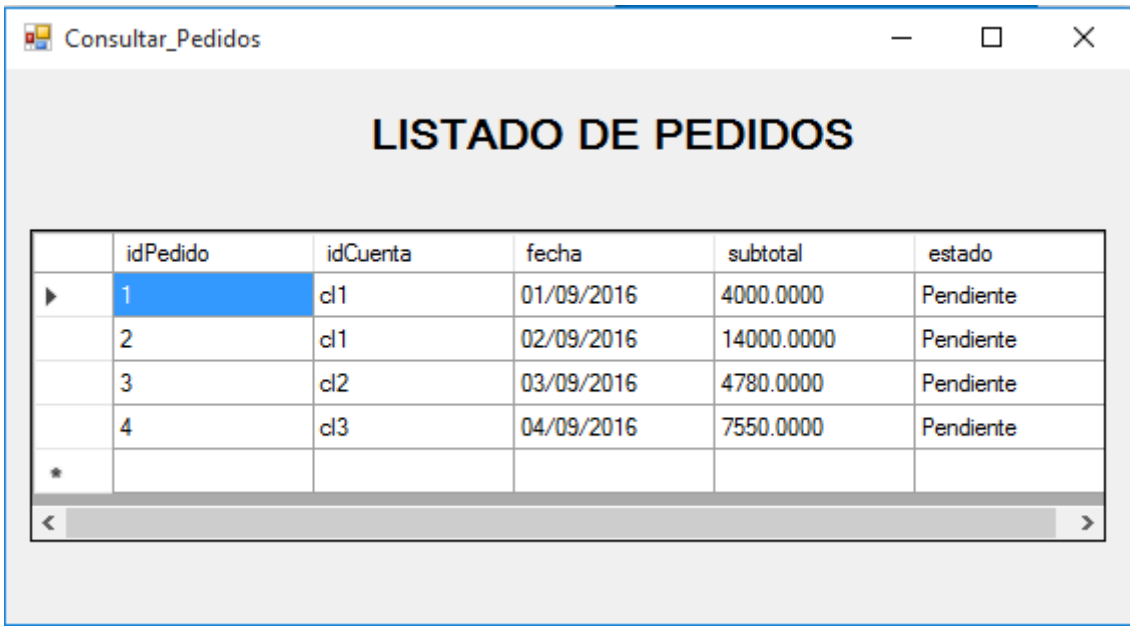
Paso 21: Agregar los elementos necesarios para la estructuración del menú principal. Una opción podría ser el contenido que muestra la Figura 112.

Figura 112. Opción de menú principal.



Paso 22: Agregar los elementos necesarios para realizar el “Listado de pedidos”. Tomar como base a la Figura 113.

Figura 113. Interfaz para el Listado de Pedidos.



Paso 23: Agregar el siguiente código correspondiente para que se muestre el listado general de los pedidos:

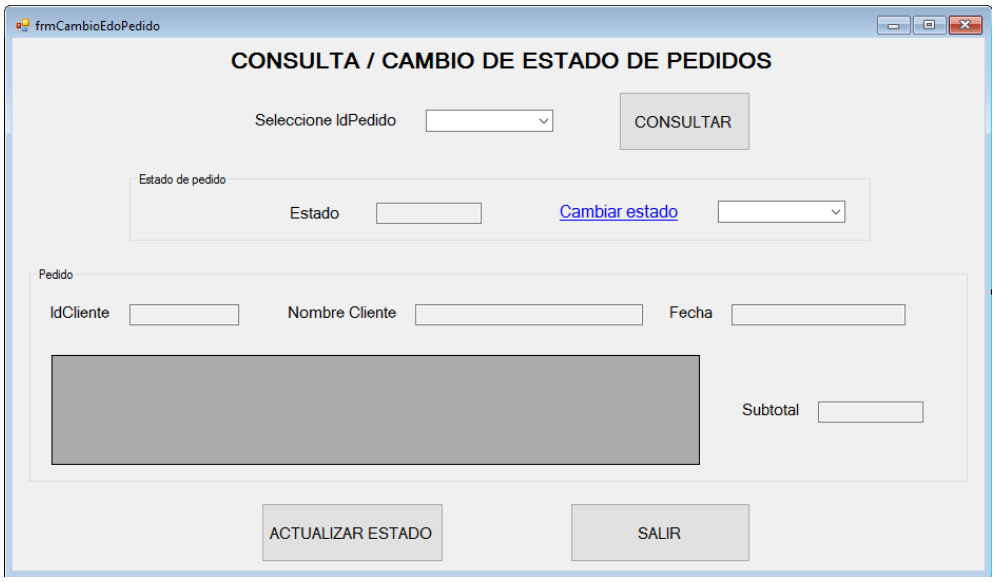
```
Imports CapaNegocios
Public Class Consultar_Pedidos
    Private Sub Consultar_Pedidos_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        Call cargarPedidos()
    End Sub
    Sub cargarPedidos()
        Dim capaNeg As New PedidoCN
        dgvPedidos.DataSource = capaNeg.LISTADOPEDIDOS.Tables("Pedidos")
    End Sub
End Class
```

Paso 24: Agregar el siguiente código para referenciar el **Listado de pedidos** en el menú principal:

```
Private Sub ListadoDePedidosToolStripMenuItem_Click(sender As Object, e As
EventArgs) Handles ListadoDePedidosToolStripMenuItem.Click
        Consultar_Pedidos.Show()
    End Sub
```

Paso 25: Agregar un nuevo Windows Forms a la **Capa de Presentación,** asignar el nombre frmCambioEdoPedido y agregar los elementos necesarios para la consulta y cambio de estado en pedido como lo muestra la Figura 114.

Figura 114. Interfaz para consultar y modificar estado de pedido.



Paso 26: Agregar el siguiente código correspondiente para que se realice la actualización del estado del pedido:

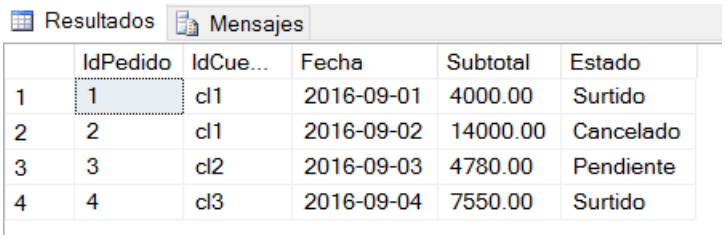
```
Imports CapaEntidad
Imports CapaNegocios
Public Class frmCambioEdoPedido
   Dim capaNegocios As New PedidoCN


   Private Sub frmCambioEdoPedido_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
      Call cargarIdPedidos()
      Call cargarEdos()

   End Sub

   Sub cargarIdPedidos()

      cboIdPedido.DataSource =
CapaNegocios.CONSULTARIDPEDIDO.Tables("Pedidos")
      cboIdPedido.DisplayMember = "IdPedido"
      cboIdPedido.ValueMember = "IdPedido"
   End Sub
```

```
    Sub cargarEdos()
        cboCambioEdo.Items.Add("Surtido")
        cboCambioEdo.Items.Add("Cancelado")
    End Sub

    Sub cargarDetalle()
        dgvDetalle.DataSource =
capaNegocios.CONSULTARDETALLEPEDIDO(Val(cboIdPedido.Text)).Tables("Detalle
Pedido")
    End Sub
    Sub cargarPedido()
        Dim t As DataTable
        t = capaNegocios.CONSULTARPEDIDO(Val(cboIdPedido.Text)).Tables("Pedidos")
        Dim r As DataRow = t.Rows(0)

        txtIdCliente.Text = CStr(r("IdCuenta"))
        txtNombreCli.Text = CStr(r("Nombre"))
        txtFecha.Text = CStr(r("Fecha"))
        txtSubtotal.Text = CStr(r("Subtotal"))
        txtEstado.Text = CStr(r("Estado"))
    End Sub

    Private Sub btnConsultar_Click(sender As Object, e As EventArgs) Handles
btnConsultar.Click
        Call cargarDetalle()
        Call cargarPedido()
    End Sub

    Private Sub lilCambiarEdo_LinkClicked(sender As Object, e As
LinkLabelLinkClickedEventArgs) Handles lilCambiarEdo.LinkClicked
        cboCambioEdo.Visible = True
    End Sub

    Private Sub btnActualizarEdo_Click(sender As Object, e As EventArgs) Handles
btnActualizarEdo.Click
        capaNegocios.ACTUALIZAPEDIDO(Val(cboIdPedido.Text), cboCambioEdo.Text)

        'Acciones a ejecutar cuando actualice el estado del pedido
        btnActualizarEdo.Enabled = False
        cboCambioEdo.Visible = False
        txtIdCliente.Text = ""
        txtNombreCli.Text = ""
        txtFecha.Text = ""
        txtSubtotal.Text = ""
        txtEstado.Text = ""
```

```
    End Sub

    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles btnSalir.Click
        Me.Dispose()
    End Sub

End Class
```

Paso 27: Ejecutar la aplicación y evaluar la funcionalidad. En la Figura 115 se muestran los datos en SQL Server:

Figura 115. Datos previos a la actualización del estado.

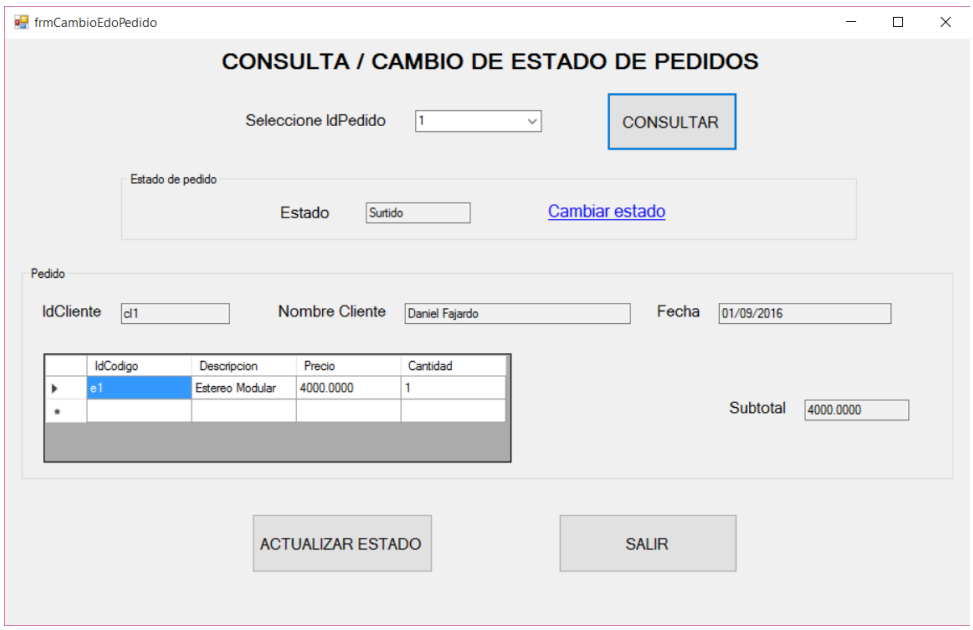


Resultados | Mensajes

	IdPedido	IdCue...	Fecha	Subtotal	Estado
1	1	cl1	2016-09-01	4000.00	Surtido
2	2	cl1	2016-09-02	14000.00	Cancelado
3	3	cl2	2016-09-03	4780.00	Pendiente
4	4	cl3	2016-09-04	7550.00	Surtido

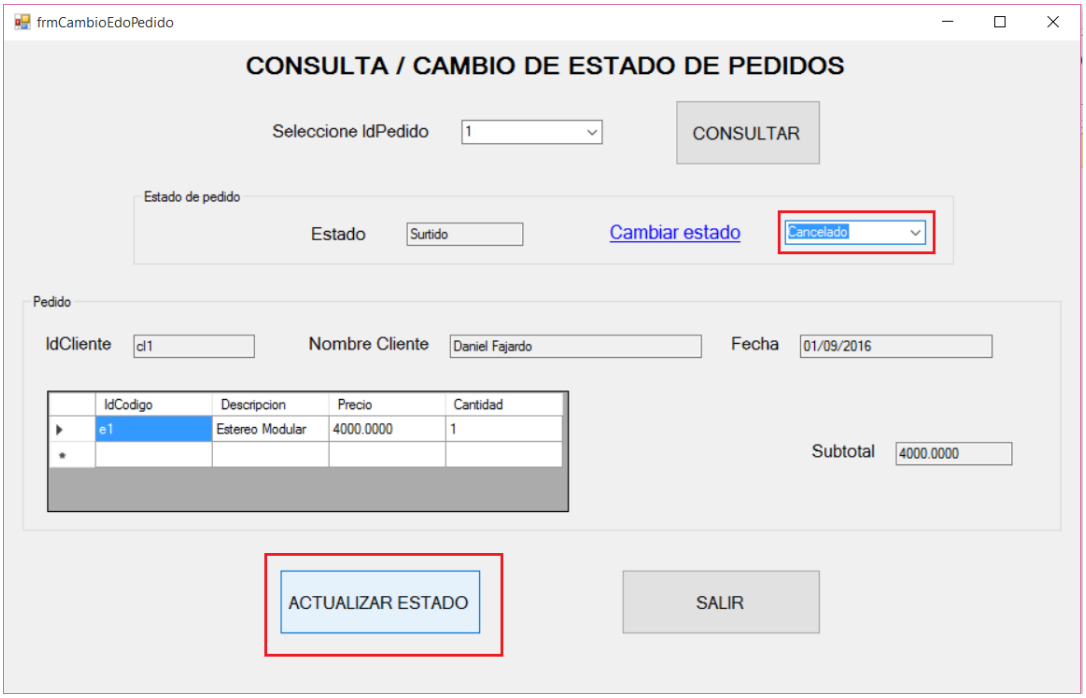
Seleccionar el IdPedido para que realice las consultas correspondientes como se muestra en la Figura 116.

Figura 116. Interfaz de consulta de pedidos.



Seleccionar la opción “Cambiar estado” y seleccionar el nuevo estado a modificar como se muestra en la Figura 117.

Figura 117. Interfaz de modificación de estado de pedido.



Una vez dado clic en “Actualizar Estado”, se observa la notificación del cambio de estado como se presenta en la Figura 118. Realizar nuevamente una consulta para comprobar el cambio, como lo muestra la Figura 119.

Figura 118. Notificación de la actualización del estado de pedido.

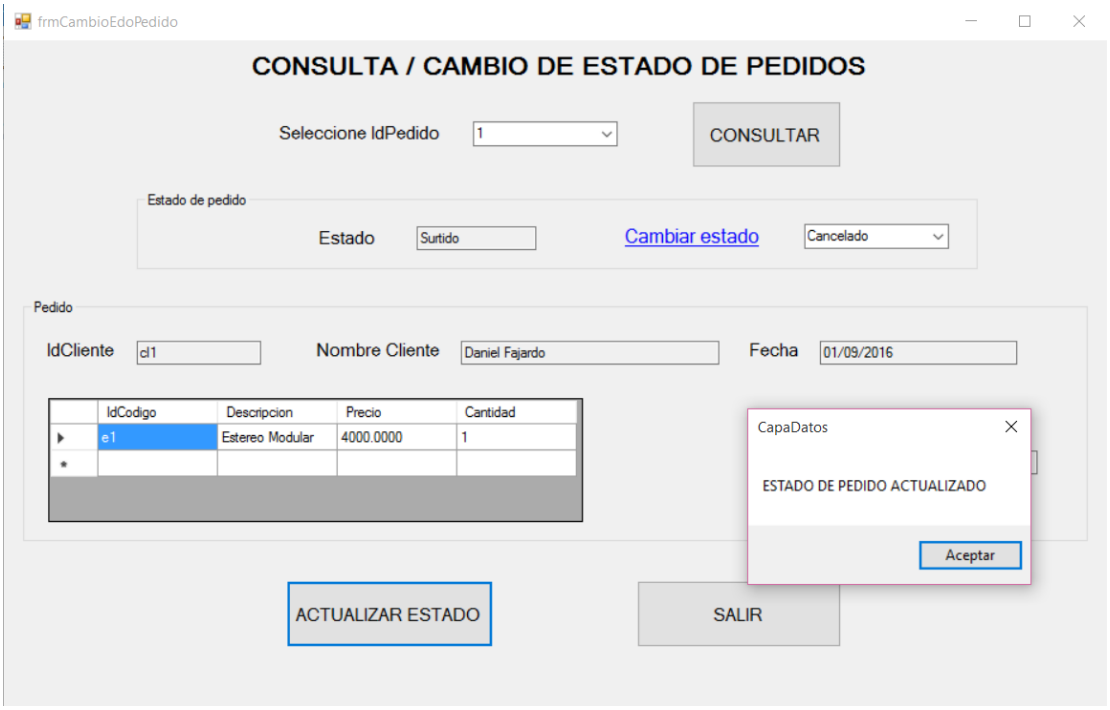
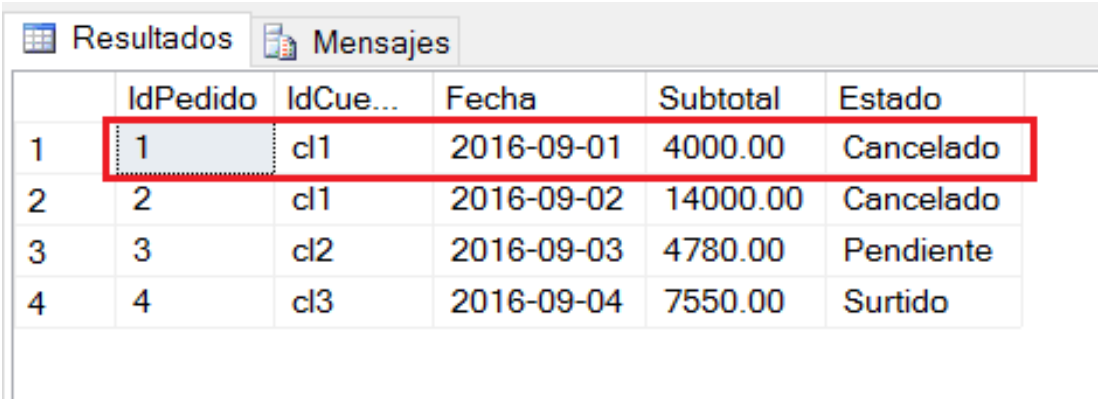


Figura 119. Comprobación de cambio de estado de pedido.



CONCLUSIONES

El presente proyecto de investigación fue propuesto a partir de la necesidad por obtener información acerca de la nueva Versión de Visual Studio 2015 y de cómo aplicar nuevas tecnologías de desarrollo de sistemas de información y sus accesos a datos, por lo que conllevó tiempo, dedicación y esfuerzo primeramente por conocer cuáles eran las diferencias de antiguas versiones en comparación con ésta y posteriormente aplicar conocimientos aprendidos.

Participar en este proyecto me deja grandes conocimientos sobre cómo determinar qué tecnologías se puede aplicar para la realización de un sistema de información, de acuerdo a las necesidades planteadas en un previo análisis y diseño tanto de la aplicación como de la base de datos. De igual forma es determinante saber cómo y de dónde adquirir información y sobretodo saber aplicarla ante una necesidad.

En este proyecto se dejan las bases de cómo aplicar nuevas herramientas de desarrollo con ejemplificaciones de los mismos, mencionando una breve explicación del cómo y el porqué de la utilización del código fuente.

Respecto a mi experiencia en la participación en éste proyecto de investigación es recomendable y necesario que los docentes motiven a los alumnos a participar en estos proyectos ya que se adquieren habilidades, destrezas, conocimientos y herramientas que nos ayudan a resolver problemas y con esto ser mejores profesionales.

Le agradezco a mi asesora, la M.C. Raquel Ochoa Ornelas por darme la oportunidad de formar parte de este proyecto, adquirir nuevos conocimientos, confiar en mis capacidades, alentarme y apoyarme en todo momento.

REFERENCIAS BIBLIOGRÁFICAS

Alonso, F., Martinez, L. y Segovia, F. J. (2005). *Introducción a la ingeniería de software. Modelos de desarrollo de programas.* España: Delta.

Arjona Torres, M. (1999). *Dirección estratégica. Un enfoque práctico. Principios y aplicaciones de la Gestión y el Rendimiento.* Madrid España: Díaz de Santos.

Barranco de Areba, J. (2001). *Metodología del análisis estructurado de sistemas*. Madrid: Ortega.

Berengel Gómez, J. L. (2016). *Desarrollo de aplicaciones web en entorno servidor UF1844.* Madrid: Nobel.

Booch, G., Rumbaugh J. y Jacobson I. (2006). *El Lenguaje Unificado de Modelado.* Madrid: Pearson Educación S.A.

Campderrich Falgueras, B. (2003). *Ingeniería de software*. Barcelona: Editorial UOC.

Caracteristicas de SQL Server Management Studio (s.f). Recuperado el 13 de diciembre del 2016, de https://msdn.microsoft.com/es-es/library/ms174219.aspx

Cebollero, M., Coles, M., Natarajan, J., Bruchez, R. y Shaw, S. (2015). *Pro T-SQL Programmer's Guide*. NewYork: Apress.

Cómo: Crear consultas de TableAdapter (s.f). Recuperado el 6 de diciembre del 2016, de https://msdn.microsoft.com/es-es/library/kda44dwy.aspx

Connection String Syntax (s.f). Recuperado el 6 de diciembre del 2016, de https://msdn.microsoft.com/en-us/library/ms254500(v=vs.110).aspx

Coral, C., Moraga, M. A. y Piattini, M. (2010). *Calidad del producto y proceso software*. Madrid: RA-MA.

DataContext (Clase) (s.f). Recuperado el 8 de diciembre del 2016, de https://msdn.microsoft.com/es-es/library/system.data.linq.datacontext(v=vs.100).aspx

De Pablos Herederos, C., López Hermoso Agius, J. J., Romo Romero, S. M. y Medina Salgado, S. (2011). *Organización y transformación de los sistemas de información en la empresa*. España: Esic.

Elmasri, R., Navathe, S. (2002). *Fundamentos de sistemas de bases de datos.* Madrid: Pearson Educación S.A.

Espacio de nombres System.Configuration (s.f). Recuperado el 2 de diciembre del 2016, de https://msdn.microsoft.com/es-es/library/system.configuration(v=vs.110).aspx

Fernandez Alarcón, V. (2006). *Desarrollo de sistemas de información. Una metodología basada en el modelado.* Barcelona: Ediciones UPC.

Fontela, C. (2011). *UML: Modelado de software para profesionales*. México: Alfaomega.

Gabillaud, J. (2013). *SQL Server 2012. SQL, Transact SQL. Diseño y creación de una base de datos*. Barcelona: Ediciones ENI.

Gabillaud, J. (2015). *SQL Server 2014. SQL, Transact SQL. Diseño y creación de una base de datos*. Barcelona: Ediciones ENI.

Información general sobre TableAdapter (s.f). Recuperado el 2 de diciembre del 2016, de https://msdn.microsoft.com/es-es/library/bz9tthwx.aspx

Jones, A., Stephens, R. K., Plew, R. R., Garrett, R. F. y Kriegel, A. (2005). *SQL Functions Programmer's Reference*. Indianápolis: Wyley Publishing, Inc.

Laudon Kenneth C. y Laudon, J. P. (2004). *Sistemas de información gerencial*. México: Pearson Educación.

LINQ to SQL Tools in Visual Studio (s.f). Recuperado el 8 de diciembre del 2016, de https://msdn.microsoft.com/en-us/library/bb384429.aspx

López Román, L. (2011). *Programación estructurada y orientada a objetos. Un enfoque algorítmico*. México: Alfaomega.

Método Table (Of TEntity). InsertOnSubmit (REntity) (s.f). Recuperado el 11 de diciembre del 2016, de https://msdn.microsoft.com/es-es/library/bb763516(v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1

Microsoft DataAdapter (s.f). Recuperado el 6 de diciembre del 2016, de https://msdn.microsoft.com/es-es/library/system.data.common.dataadapter.aspx

Microsoft DataRowView (s.f). Recuperado el 7 de diciembre del 2016, de https://msdn.microsoft.com/es-es/library/system.data.datarowview(v=vs.110).aspx

Microsoft DataSet (s.f). Recuperado el 1 de diciembre del 2016, de https://msdn.microsoft.com/es-mx/library/system.data.dataset(v=vs.110).aspx

Microsoft DataTable (s.f). Recuperado el 3 de diciembre del 2016, de https://msdn.microsoft.com/es-es/library/system.data.datatable.aspx

Microsoft DataView (s.f). Recuperado el 7 de diciembre del 2016, de https://msdn.microsoft.com/es-es/library/system.data.dataview(v=vs.110).aspx

Microsoft SqlCommand (s.f). Recuperado el 1 de diciembre del 2016, de https://msdn.microsoft.com/es-es/library/system.data.sqlclient.sqlcommand(v=vs.110).aspx

Microsoft SqlDataAdapter (s.f). Recuperado el 1 de diciembre del 2016, de https://msdn.microsoft.com/es-es/library/system.data.sqlclient.sqldataadapter(v=vs.110).aspx

Novedades de Visual Studio 2015 (s.f). Recuperado el 1 de diciembre del 2016, de https://msdn.microsoft.com/es-mx/library/bb386063.aspx

Operaciones básicas de consulta (Visual Basic) (s.f). Recuperado el 1 de diciembre del 2016, de https://msdn.microsoft.com/es-mx/library/bb384504.aspx

Pérez López, C. (2007). *Microsoft SQL Server 2005. Administración y análisis de bases de datos*. México: Alfaomega Grupo Editor S.A de C.V.

Pialorsi, P., Russo M. (2008). *Programación LinQ.* Madrid: Grupo Anaya S.A.

Pressman, R. S. (2005). *Ingeniería del software. Un enfoque práctico*. México: McGraw-Hill.

Propiedad SqlCommand.CommandType (s.f). Recuperado el 1 de diciembre del 2016, de https://msdn.microsoft.com/es-es/library/system.data.sqlclient.sqlcommand.commandtype(v=vs.110).aspx

Torres R., Manuel Á. (2012). *Programación orientada a objetos con Visual Basic 2012*. Lima-Perú: Macro.

Viso, E. y Pelaez, C. (2007). *Introducción a las ciencias de la computación con JAVA*. México: Las prensas de ciencias.

Walkthrough: Creating LINQ to SQL Classes (O/R Designer). (s.f). Recuperado el 1 de diciembre del 2016, de https://msdn.microsoft.com/en-us/library/bb384428.aspx