

UNIVERSIDAD PRIVADA-DE-TACNA



INGENIERIA DE SISTEMAS

TITULO:

Trabajo Final de Unidad

CURSO:

BASE DE DATOS II

DOCENTE(ING):

Patrick Cuadros Quiroga

Integrantes:

Orlando Antonio Acosta Ortiz	(2015052775)
Orestes Ramirez Ticona	(2015053236)
Nilson Laura Atencio	(2015053846)
Roberto Zegarra Reyes	(2010036175)
Richard Cruz Escalante	(2013047247)
Wilfredo Vilca Chambilla	(2006028540)

Índice

1. PROBLEMÁTICA	1
2. MARCO TEÓRICO	2
2.1. Entity Framework	2
2.2. API REST	2
2.3. Consultas LINQ	2
2.4. Pruebas Unitarias	3
3. DESARROLLO	4
3.1. Desarrollo del Sistema	4
3.2. Analisis	9
3.3. Diseño	15
3.4. Pruebas	21
4. ANALISIS E INTERPRETACION DE RESULTADOS	25
5. CONCLUSIONES	27

1. PROBLEMÁTICA

El siguiente trabajo se desarrolla con el Sistema de Cajero Automatico, el cual busca que el cualquier cajero del Banco Pichincha pueda realizar acciones de forma mas automatizada y con toda seguridad:

- La Gestion de Cajero, Retiros, Depositos, Consultas
- Gestión de las usuarios que cuenten con una cuenta en el banco PICHINCHA.
- La venta de distintos tipos de planes.
- La realización de reportes sobre las transacciones realizadas del día, del mes, y sobre los clientes correspondientes a cada local.
- El sistema permitirá realizar la autenticación de los usuarios.

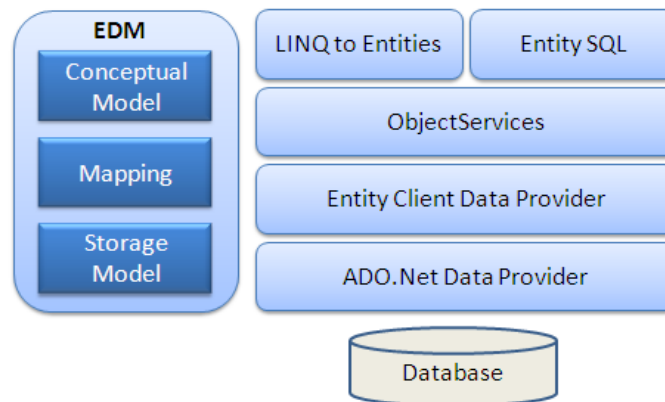
La motivación principal de este proyecto es que los cajeros del Banco Pichincha tengan una automatizacion mas eficaz, lo que le permitirá realizar sus tareas sin inconvenientes.

2. MARCO TEORICO

2.1. Entity Framework

Entity Framework es un marco de ORM de código abierto para aplicaciones .NET admitidas por Microsoft. Permite a los desarrolladores trabajar con datos utilizando objetos de clases específicas del dominio sin centrarse en las tablas y columnas de la base de datos subyacente donde se almacenan estos datos. Con el Entity Framework, los desarrolladores pueden trabajar en un nivel más alto de abstracción cuando tratan con datos, y pueden crear y mantener aplicaciones orientadas a datos con menos código en comparación con las aplicaciones tradicionales.[2]

Su arquitectura:



2.2. API REST

Ha pasado más de una década desde que Roy Fielding, un científico informático estadounidense y uno de los autores principales de la especificación HTTP, introdujo Representational State Transfer (REST) como un estilo de arquitectura. A lo largo de los años, REST ha ganado impulso gracias a su popularidad para la creación de servicios web.

Al no tener estado, y al crear identificadores únicos para permitir el almacenamiento en caché, la creación de capas y la capacidad de lectura, las API REST hacen uso de los verbos HTTP existentes (GET, POST, PUT y DELETE) para crear, actualizar y eliminar nuevos recursos. El término REST se usa con frecuencia para describir cualquier URL que devuelva JSON en lugar de HTML.

2.3. Consultas LINQ

Permite consultar cualquier objeto enumerable en ADO.NET mediante el uso del modelo de programación de Language-Integrated Query (LINQ) [1] para recuperar datos de la base de datos subyacente. El proveedor de la base de datos traducirá estas consultas LINQ al lenguaje de consulta específico de la base de datos (por ejemplo, SQL para una base de datos relacional).

Hay 3 tecnologías ADO.NET LINQ distintas:

- LINQ to DataSet: proporciona una capacidad de consulta mas rica y optimizada sobre DataSet.

- LINQ to SQL: permite consultar directamente los esquemas de las base de datos de SQL Server.
- LINQ to Entities: permite consultar Entity Data Model.

2.4. Pruebas Unitarias

Las pruebas unitarias son la mejor forma de probar el código de una aplicación a lo largo de su desarrollo. Estas pruebas permiten asegurar que los métodos devuelven resultados correctos, teniendo en cuenta los argumentos que se le pasan y, además, se pueden utilizar para hacer Test Driven Development. Se trata de una técnica en la que se escriben antes que las clases y los métodos.^[4]

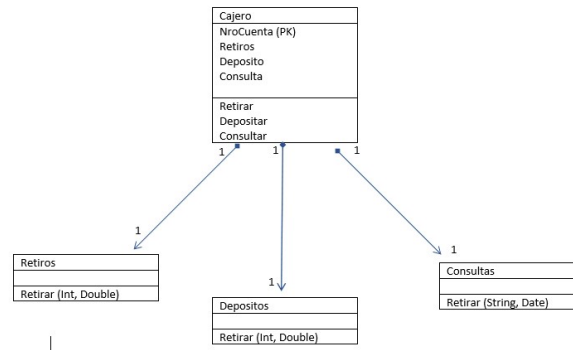
No obstante, tras realizar la integración con otros módulos deberá revisarse de nuevo la interfaz.

3. DESARROLLO

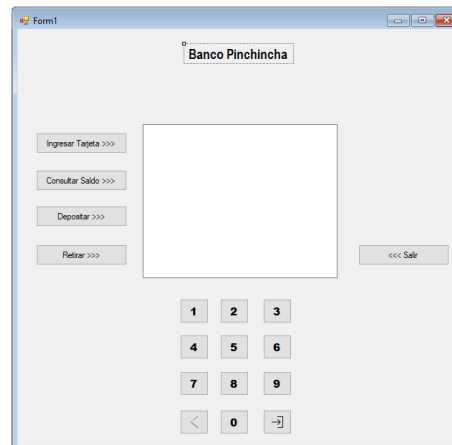
3.1. Desarrollo del Sistema

1. Escribiendo consultas con el operador PIVOT

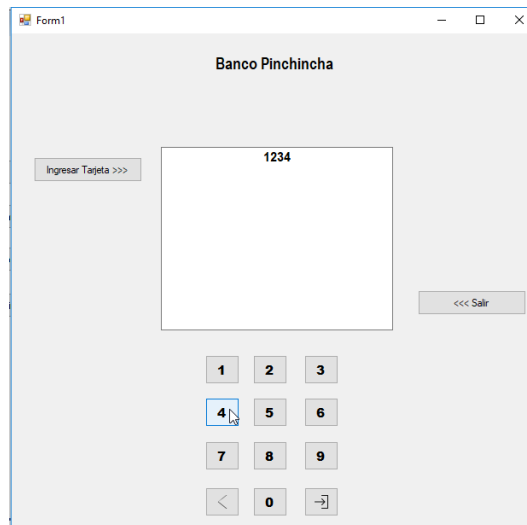
a) Paso 1: Diagrama de Clases de solución Cajero



b) Paso2: Crear un Windows form para desorrallar nuestra aplicación de escritorio:



c) Paso 3: Codificar los botones según lo planeado para nuestra aplicación, por ejemplo para esta aplicación hemos decidido virtualizar un cajero automático.



d) Paso 4: Muestra de algo de código de los botones:

```

1 referencia
private void Button1_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "1";
}

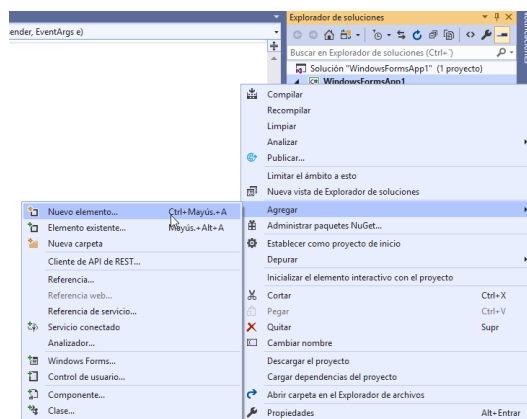
1 referencia
private void BtnMiro2_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "2";
}

1 referencia
private void BtnMiro3_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "3";
}

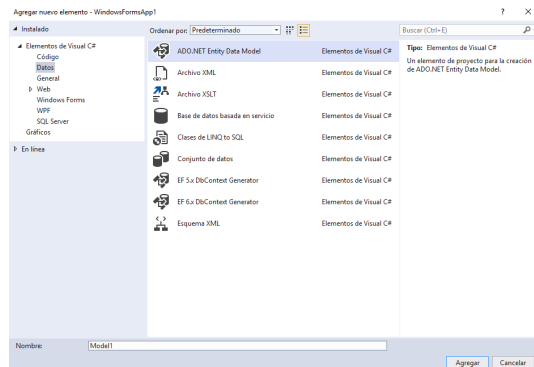
1 referencia
private void BtnMiro4_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "4";
}

```

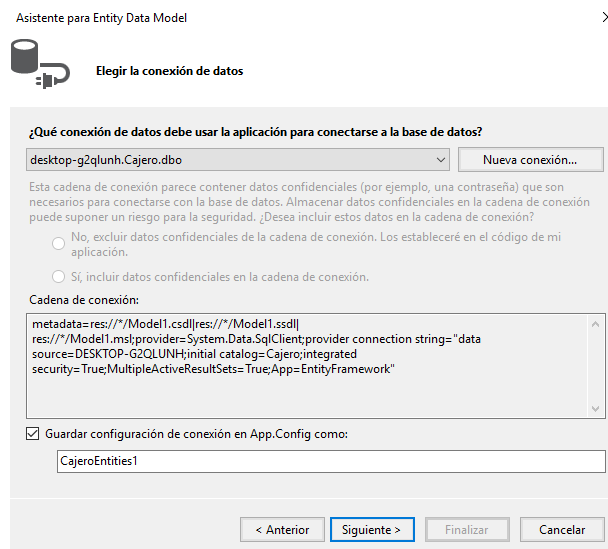
e) Paso 5: aplicar un ORM a nuestra aplicación: aplicaremos el ORM Entity framework(define que es y para que sirve entity framework)



f) Paso 6: Agregaremos el modelo ADO.NET entity data model

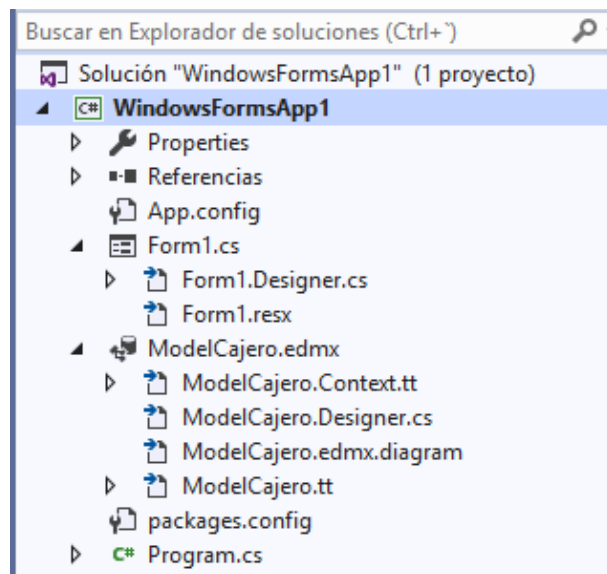


g) Paso 7: ELuego configuraremos con la base de datos que se encuentra en Nuestro servidor SQL



2. Escribiendo consultas con el operador UNPIVOT

- a) Paso 1: Crear y consultar la vista Sale.PivotCustGroups.
 - Escribir la siguiente consulta para crear una vista llamada Sales.PivotCustGroups.

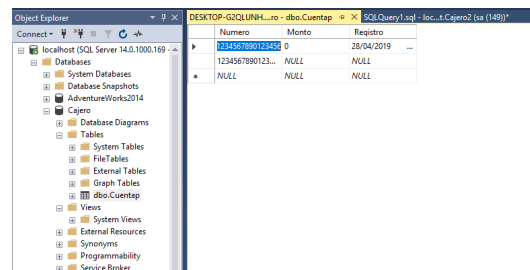


- Después escribir la siguiente consulta.

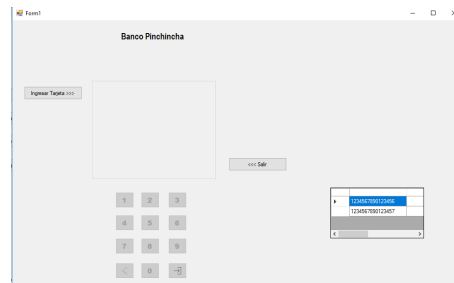
```
1 referencia
public void llamarDatos()
{
    using (CajeroEntities db = new CajeroEntities())
    {
        var list = db.Cuentas;
        foreach (var oCuenta in list)
        {
            dgvDatos.ColumnCount = 4;
            dgvDatos.Rows.Add(oCuenta.Numero);
        }
    }
}
```

b) Paso 2: Escriba una instrucción SELECT para recuperar una fila para cada país y grupo de cliente.

- En el panel de consulta escribir la siguiente consulta.



c) Paso 3: Eliminar las vistas creadas.



3. Escribiendo consultas con las clausulas GROUPING SETS, CUBE, and ROLLUP.

- a) Paso 1: Escriba una instruccion SELECT que use LA SUBCLAUSULA GROUPING SETS para devolver el número de Clientes para diferentes conjuntos de agrupación.
- Escribir la siguiente consulta y ejecutar.

```

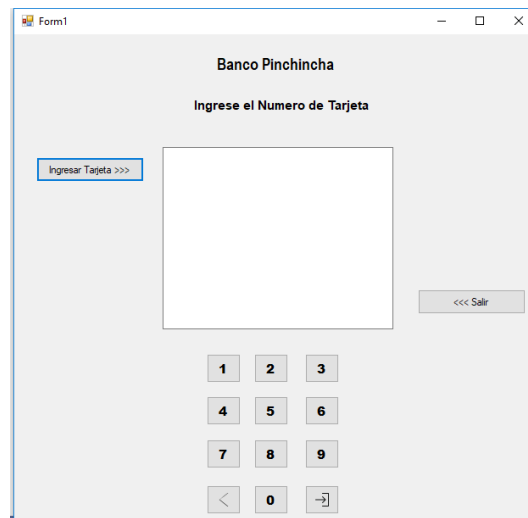
1 referencia
public void Logear()
{
    using (CajeroEntities db = new CajeroEntities())
    {
        NroCuenta = TxtPantalla.Text;

        for (int i = 0; i < DgvDatos.RowCount; i++)
        {
            string dato;
            dato = Convert.ToString(DgvDatos.Rows[i].Cells[0].Value);

            if (NroCuenta == dato)
            {
                lblGuia.Text = "Bienvenido";
                MostrarBotones();
                break;
            }
            else
            {
                lblGuia.Text = "Usted No esta Registrado";
                OcultarBotones();
            }
        }
    }
}

```

- b) Paso 2: Escriba una instruccion SELECT que use la subclausula CUBE para recuperar Grouping sets basados en valores de ventas anuales, mensuales y diarios.
- Escribir la siguiente consulta.



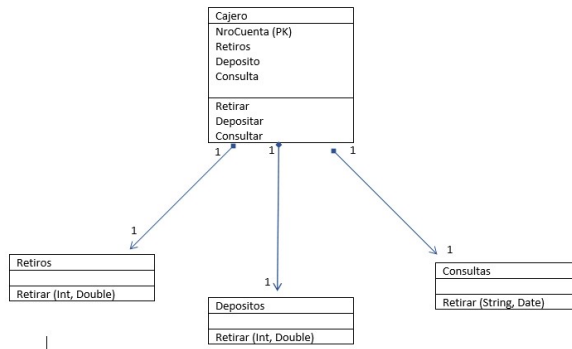
- c) Paso 3: Escriba la misma instruccion SELECT usando la subclausula ROLLUP.
- Escribir la siguiente consulta.

- d) Paso 4: Analizar el valor total de ventas por año y mes.
- Escribir la siguiente consulta y ejecutar.

3.2. Analisis

1. Escribiendo consultas con el operador PIVOT

- a) Paso 1: Escribir una sentencia SELECT para recuperar el numero de clientes para un grupo especifico de clientes.
- Abrir el SQL Server Management Studio y conectar a la basa de datos (local) usando Windows.
- Usar la base de datos TSQL
- Ejecutar el siguiente codigo para crear una vista



- En el panel de consulta, escribir la siguiente consulta.

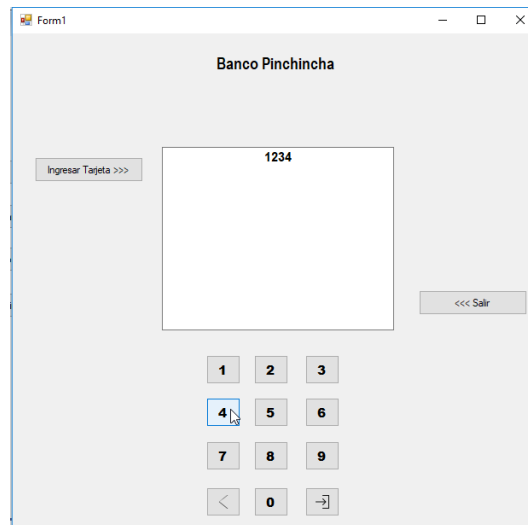
```

1//-----
2public void Consultar()
3{
4    using (CajeroEntities db = new CajeroEntities())
5    {
6        for (int i = 0; i < dgDatos.RowCount; i++)
7        {
8            string datos;
9            datos = Convert.ToString(dgDatos.Rows[i].Cells[0].Value);
10           if (NroCuentas.Count == 0)
11           {
12               TxtPantalla.Text = "No se encontro la cuenta";
13           }
14           else
15           {
16               TxtPantalla.Text = "Saldo de la cuenta: " + Convert.ToString(dgDatos.Rows[i].Cells[1].Value) + " Soles.";
17           }
18       }
19   }
20 }
  
```

- Luego modificamos el codigo, aplicando el operador PIVOT.

b) Paso 2: Especifique el elemento de agrupacion para el operador PIVOT.

- Escribir la siguiente consulta para poder modificar la vista creada anteriormente, añadiendo 2 columnas adicionales.



- Escribir la siguiente consulta.

```

1 referencia
private void Button1_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "1";
}

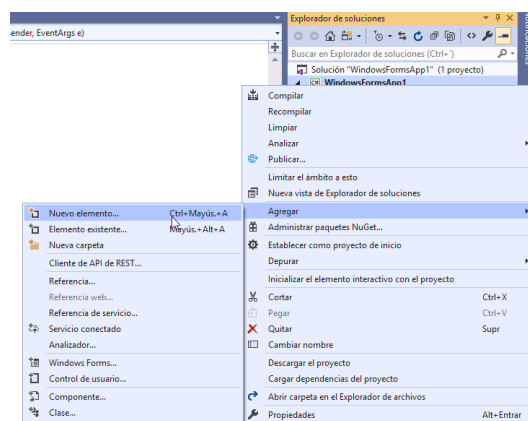
2 referencia
private void BtnMro2_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "2";
}

3 referencia
private void BtnMro3_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "3";
}

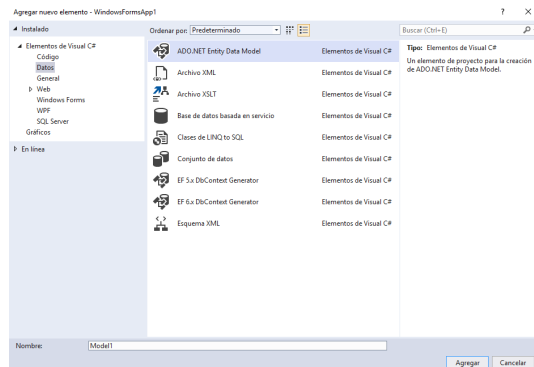
4 referencia
private void BtnMro4_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "4";
}

```

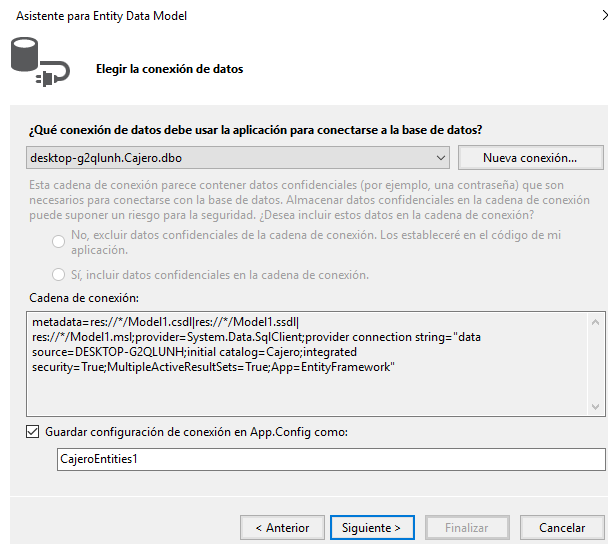
- Modificar la consulta para incluir columnas adicionales desde la vista.



- c) Paso 3: Use una expresion de tabla común (CTE) para especificar el elemnto de agrupacion para el operador PIVOT.
- Escribir la siguiente consulta y ejecutar.

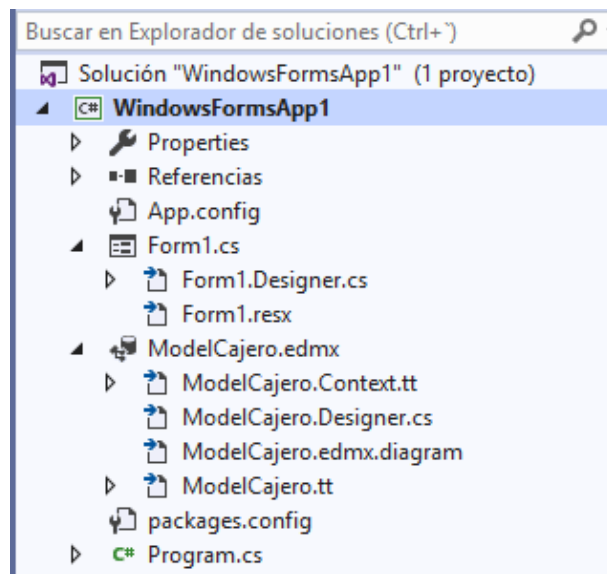


- d) Paso 4: Escribe una instrucción SELECT para recuperar el monto total de ventas para cada cliente y categoría de producto.
- Escribir la siguiente consulta.



2. Escribiendo consultas con el operador UNPIVOT

- a) Paso 1: Crear y consultar la vista Sale.PivotCustGroups.
- Escribir la siguiente consulta para crear una vista llamada Sales.PivotCustGroups.

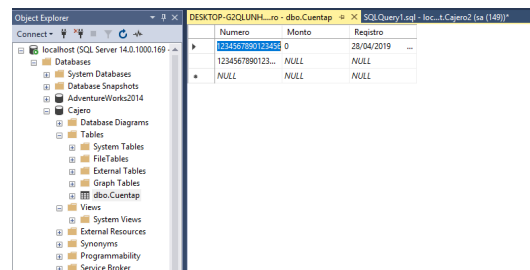


- Después escribir la siguiente consulta.

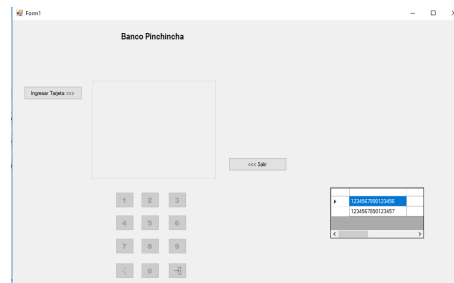
```
1 referencia
public void llamarDatos()
{
    using (CajeroEntities db = new CajeroEntities())
    {
        var list = db.Cuentas;
        foreach (var oCuenta in list)
        {
            dgvDatos.ColumnCount = 4;
            dgvDatos.Rows.Add(oCuenta.Numero);
        }
    }
}
```

b) Paso 2: Escriba una instrucción SELECT para recuperar una fila para cada país y grupo de cliente.

- En el panel de consulta escribir la siguiente consulta.



c) Paso 3: Eliminar las vistas creadas.



3. Escribiendo consultas con las clausulas GROUPING SETS, CUBE, and ROLLUP.

- a) Paso 1: Escriba una instruccion SELECT que use LA SUBCLAUSULA GROUPING SETS para devolver el número de Clientes para diferentes conjuntos de agrupación.
- Escribir la siguiente consulta y ejecutar.

```

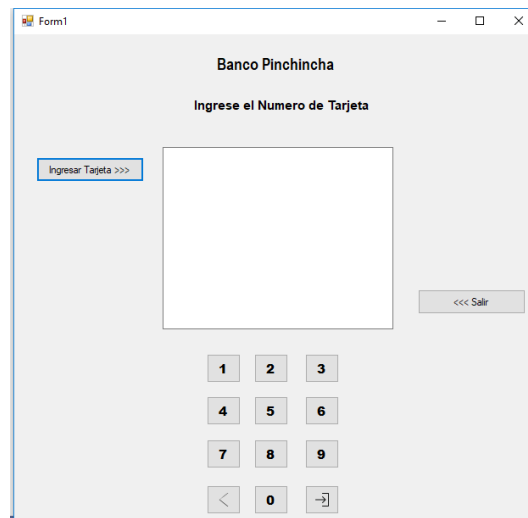
1 referencia
public void Logear()
{
    using (CajeroEntities db = new CajeroEntities())
    {
        NroCuenta = TxtPantalla.Text;

        for (int i = 0; i < DgvDatos.RowCount; i++)
        {
            string dato;
            dato = Convert.ToString(DgvDatos.Rows[i].Cells[0].Value);

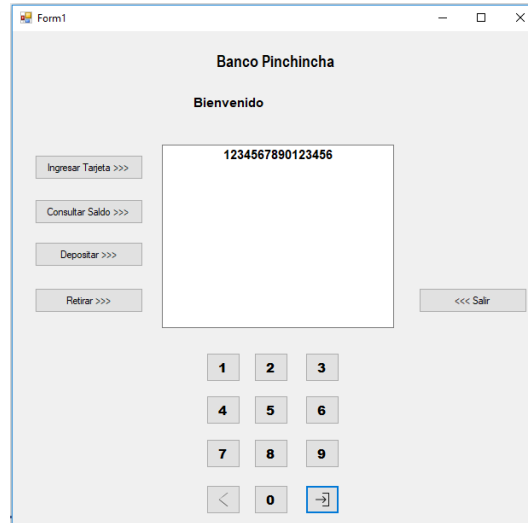
            if (NroCuenta == dato)
            {
                lblGuia.Text = "Bienvenido";
                MostrarBotones();
                break;
            }
            else
            {
                lblGuia.Text = "Usted No esta Registrado";
                OcultarBotones();
            }
        }
    }
}

```

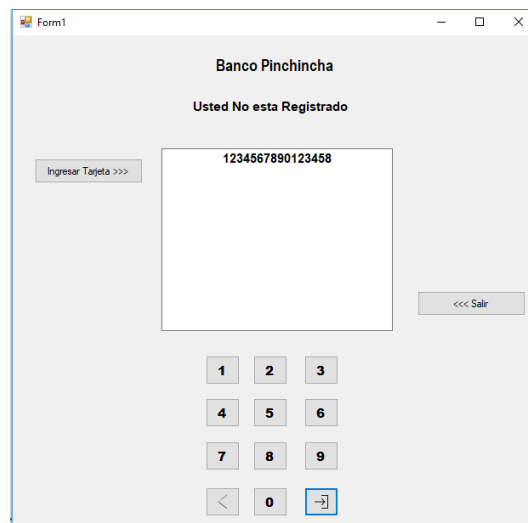
- b) Paso 2: Escriba una instruccion SELECT que use la subclausula CUBE para recuperar Grouping sets basados en valores de ventas anuales, mensuales y diarios.
- Escribir la siguiente consulta.



- c) Paso 3: Escriba la misma instruccion SELECT usando la subclausula ROLLUP.
- Escribir la siguiente consulta.



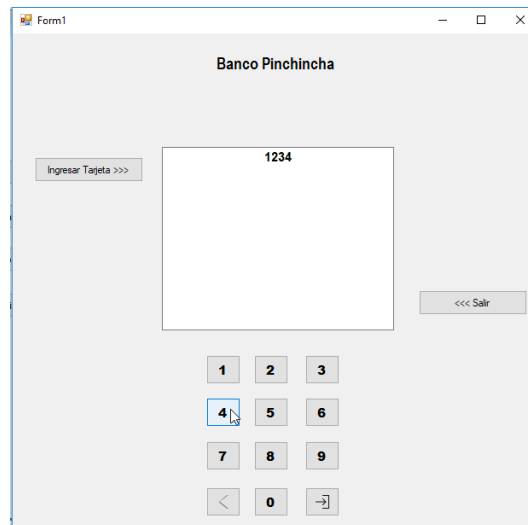
- d) Paso 4: Analizar el valor total de ventas por año y mes.
- Escribir la siguiente consulta y ejecutar.



3.3. Diseño

1. Escribiendo consultas con el operador PIVOT

- a) Paso 1: Escribir una sentencia SELECT para recuperar el numero de clientes para un grupo especifico de clientes.
- Abrir el SQL Server Management Studio y conectar a la basa de datos (local) usando Windows.
- Usar la base de datos TSQL
- Ejecutar el siguiente codigo para crear una vista



- Escribir la siguiente consulta.

```

1 referencia
private void Button1_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "1";
}

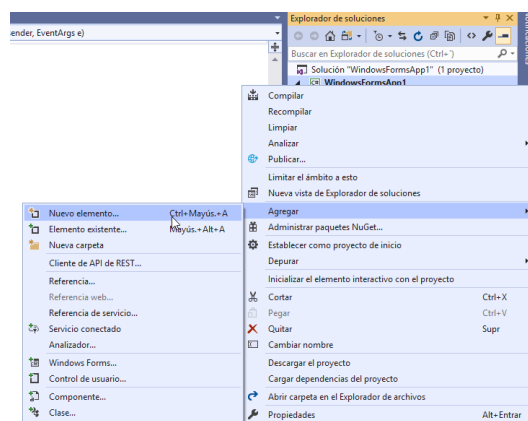
2 referencia
private void BtnMro2_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "2";
}

3 referencia
private void BtnMro3_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "3";
}

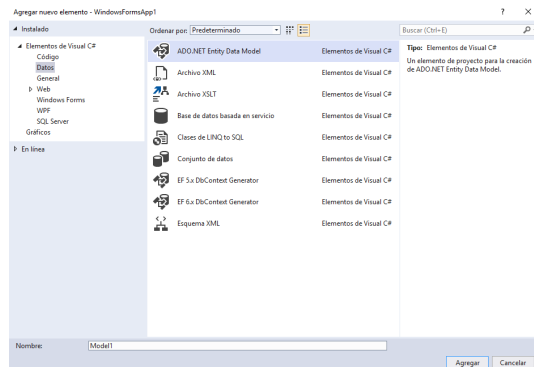
4 referencia
private void BtnMro4_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "4";
}

```

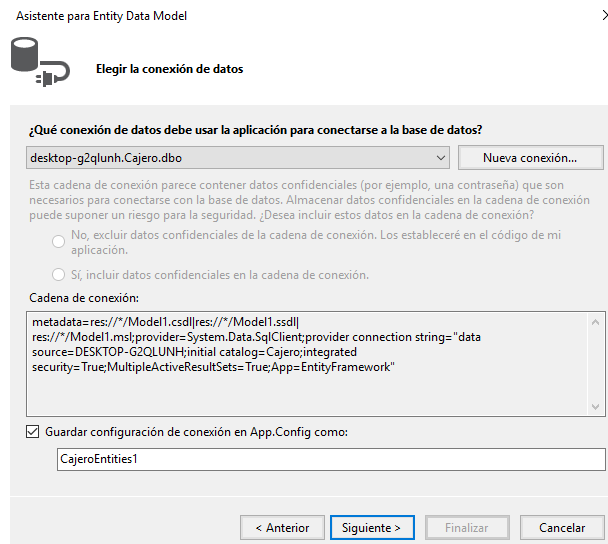
- Modificar la consulta para incluir columnas adicionales desde la vista.



- c) Paso 3: Use una expresion de tabla común (CTE) para especificar el elemnto de agrupacion para el operador PIVOT.
- Escribir la siguiente consulta y ejecutar.

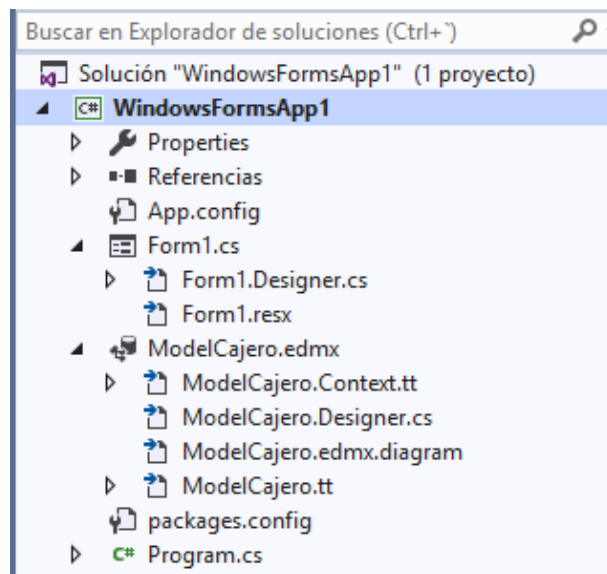


- d) Paso 4: Escribe una instrucción SELECT para recuperar el monto total de ventas para cada cliente y categoría de producto.
- Escribir la siguiente consulta.



2. Escribiendo consultas con el operador UNPIVOT

- a) Paso 1: Crear y consultar la vista Sale.PivotCustGroups.
- Escribir la siguiente consulta para crear una vista llamada Sales.PivotCustGroups.

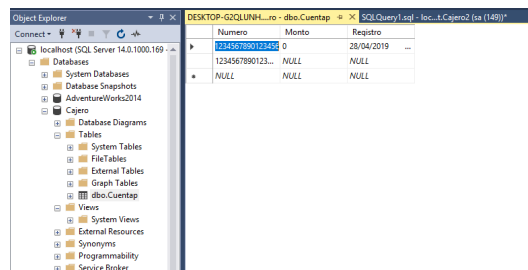


- Después escribir la siguiente consulta.

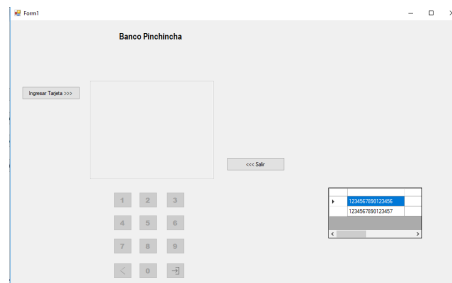
```
1 referencia
public void llamarDatos()
{
    using (CajeroEntities db = new CajeroEntities())
    {
        var list = db.Cuentas;
        foreach (var oCuenta in list)
        {
            dgvDatos.ColumnCount = 4;
            dgvDatos.Rows.Add(oCuenta.Numero);
        }
    }
}
```

b) Paso 2: Escriba una instrucción SELECT para recuperar una fila para cada país y grupo de cliente.

- En el panel de consulta escribir la siguiente consulta.



c) Paso 3: Eliminar las vistas creadas.



3. Escribiendo consultas con las clausulas GROUPING SETS, CUBE, and ROLLUP.

- a) Paso 1: Escriba una instruccion SELECT que use LA SUBCLAUSULA GROUPING SETS para devolver el número de Clientes para diferentes conjuntos de agrupación.
- Escribir la siguiente consulta y ejecutar.

```

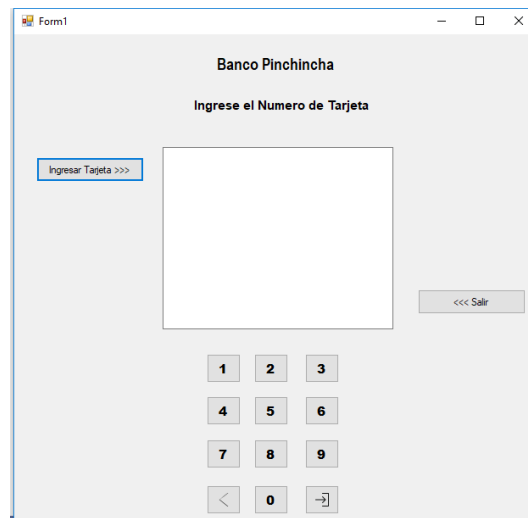
1 referencia
public void Logear()
{
    using (CajeroEntities db = new CajeroEntities())
    {
        NroCuenta = TxtPantalla.Text;

        for (int i = 0; i < DgvDatos.RowCount; i++)
        {
            string dato;
            dato = Convert.ToString(DgvDatos.Rows[i].Cells[0].Value);

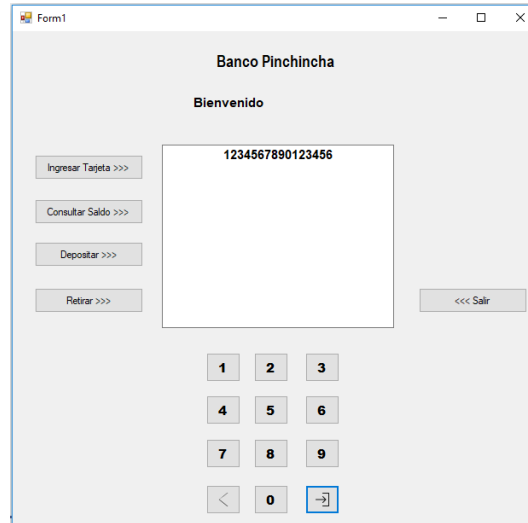
            if (NroCuenta == dato)
            {
                lblGuia.Text = "Bienvenido";
                MostrarBotones();
                break;
            }
            else
            {
                lblGuia.Text = "Usted No esta Registrado";
                OcultarBotones();
            }
        }
    }
}

```

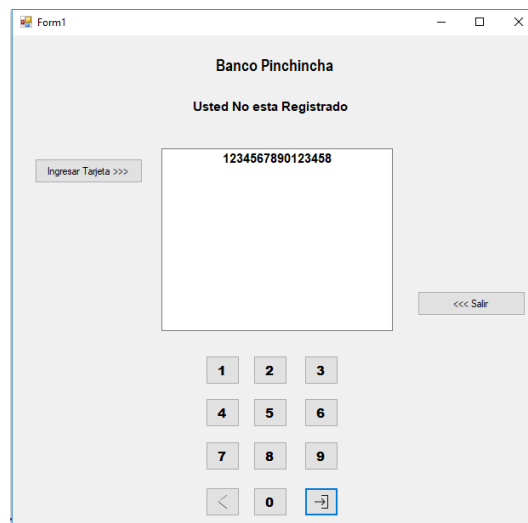
- b) Paso 2: Escriba una instruccion SELECT que use la subclausula CUBE para recuperar Grouping sets basados en valores de ventas anuales, mensuales y diarios.
- Escribir la siguiente consulta.



- c) Paso 3: Escriba la misma instruccion SELECT usando la subclausula ROLLUP.
- Escribir la siguiente consulta.



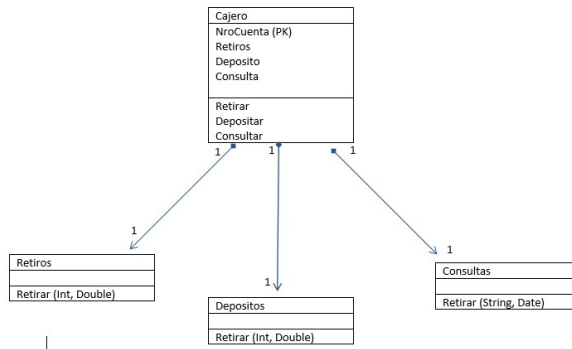
- d) Paso 4: Analizar el valor total de ventas por año y mes.
- Escribir la siguiente consulta y ejecutar.



3.4. Pruebas

1. Escribiendo consultas con el operador PIVOT

- a) Paso 1: Escribir una sentencia SELECT para recuperar el numero de clientes para un grupo especifico de clientes.
- Abrir el SQL Server Management Studio y conectar a la basa de datos (local) usando Windows.
- Usar la base de datos TSQL
- Ejecutar el siguiente codigo para crear una vista



- En el panel de consulta, escribir la siguiente consulta.

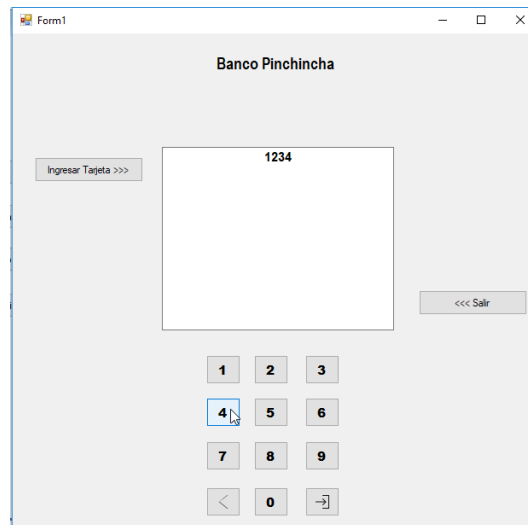
```

1//-----
2public void Consultar()
3{
4    using (CajeroEntities db = new CajeroEntities())
5    {
6        for (int i = 0; i < dgDatos.RowCount; i++)
7        {
8            string datos;
9            datos = Convert.ToString(dgDatos.Rows[i].Cells[0].Value);
10           if (NroCuentas.Count == datos)
11           {
12               TxtPantalla.Text = "Total Time : " + Convert.ToString(dgDatos.Rows[i].Cells[1].Value) + " Segs.";
13           }
14       }
15   }
16 }
  
```

- Luego modificamos el codigo, aplicando el operador PIVOT.

b) Paso 2: Especifique el elemento de agrupacion para el operador PIVOT.

- Escribir la siguiente consulta para poder modificar la vista creada anteriormente, añadiendo 2 columnas adicionales.



- Escribir la siguiente consulta.

```

1 referencia
private void Button1_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "1";
}

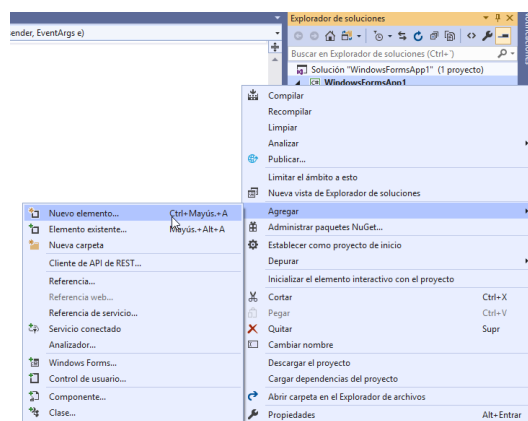
2 referencia
private void BtnMro2_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "2";
}

3 referencia
private void BtnMro3_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "3";
}

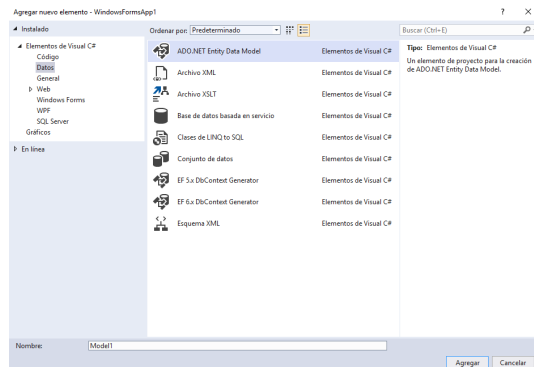
4 referencia
private void BtnMro4_Click(object sender, EventArgs e)
{
    TxtPantalla.Text = TxtPantalla.Text + "4";
}

```

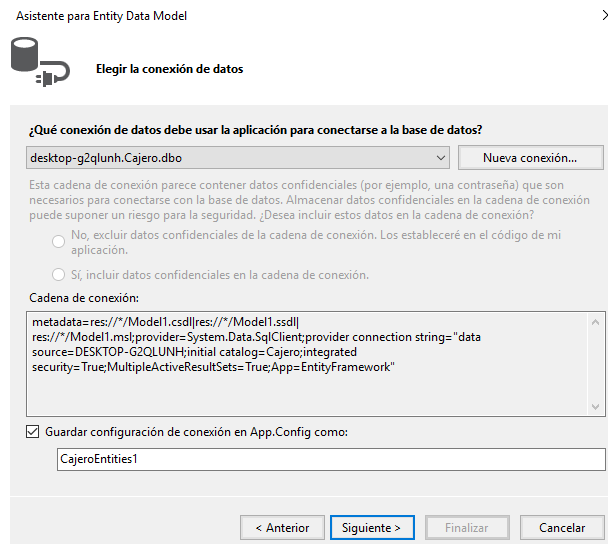
- Modificar la consulta para incluir columnas adicionales desde la vista.



- c) Paso 3: Use una expresion de tabla común (CTE) para especificar el elemnto de agrupacion para el operador PIVOT.
- Escribir la siguiente consulta y ejecutar.



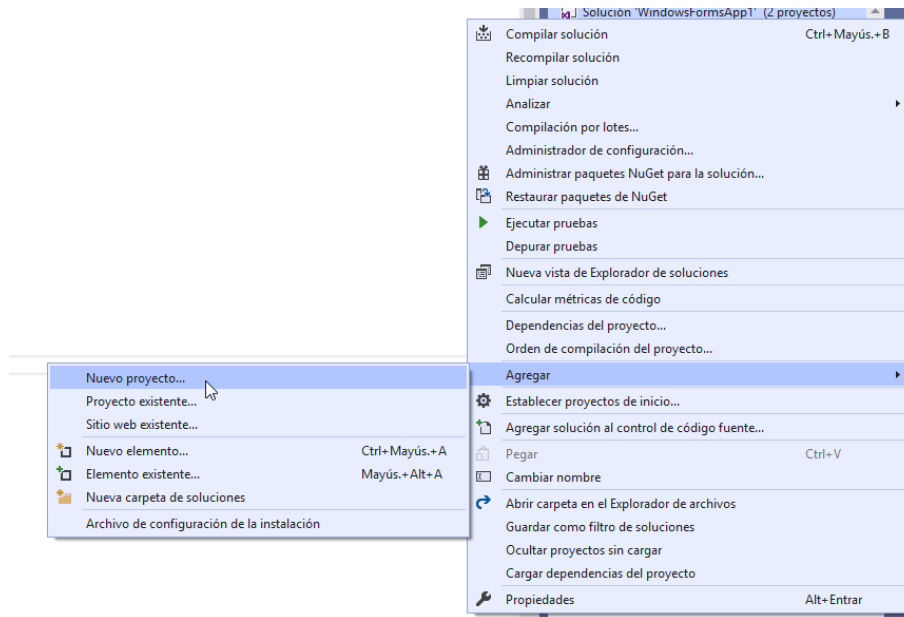
- d) Paso 4: Escribe una instrucción SELECT para recuperar el monto total de ventas para cada cliente y categoría de producto.
- Escribir la siguiente consulta.



4. ANALISIS E INTERPRETACION DE RESULTADOS

-Al compilar el proyecto de prueba, las pruebas aparecen en el Explorador de pruebas. Si el Explorador de pruebas no está visible, elija Prueba en el menú de Visual Studio, elija Ventanas y, después, Explorador de pruebas. -Se puede elegir Ejecutar todas para ejecutar todas las pruebas o bien Ejecutar para elegir un subconjunto de pruebas que se desea ejecutar. Después de ejecutar un conjunto de pruebas, aparecerá un resumen de la serie de pruebas en la parte inferior de la ventana Explorador de pruebas. Seleccione una prueba para ver los detalles de esa prueba en el panel inferior.










- Al compilar el proyecto de prueba.



- Explorador de pruebas.

Agregar un nuevo proyecto

Plantillas de proyecto recientes

-  Proyecto de prueba unitaria (.NET Framework) C#
-  Proyecto de prueba NUnit (.NET Core) C#
-  Aplicación de Windows Forms (.NET Framework) C#
-  Biblioteca de controles de Windows Forms (.NET Framework) C#
-  Aplicación de consola (.NET Core) C#
-  Aplicación web ASP.NET Core C#
-  Aplicación web ASP.NET (.NET Framework) C#
-  Biblioteca de clases (Windows universal) C#
-  Proyecto de pruebas xUnit (.NET Core) C#

X Lenguaje Plataforma Tipo de proyecto

 Proyecto de prueba de MSTest (.NET Core)
Proyecto que contiene pruebas unitarias de MSTest que se pueden ejecutar en .NET Core en Windows, Linux y MacOS.
C# Linux macOS Windows Prueba

 Proyecto de prueba unitaria (.NET Framework)
Proyecto que contiene pruebas unitarias.
C# Windows Prueba

 Proyecto de pruebas xUnit (.NET Core)
Proyecto que contiene pruebas xUnit que se pueden ejecutar en .NET Core en Windows, Linux y MacOS.
C# Windows Linux macOS Prueba

 Prueba de controladores web para Edge (.NET Core)
Proyecto que contiene pruebas unitarias que pueden automatizar las pruebas de IU de los sitios web en el explorador Edge (con Microsoft WebDriver).
C# Windows Web

 Prueba de controladores web para Edge (.NET Framework)
Proyecto que contiene pruebas unitarias que pueden automatizar las pruebas de IU de los sitios web en el explorador Edge (con Microsoft WebDriver).
C# Windows Web

 [Documentos de muestra controlado \(.NET Framework\)](#)

Siguiente

5. CONCLUSIONES

Las pruebas unitarias nos pueden ayudar a testear los detalles de nuestro programa son el primer paso en la realización de pruebas de un software, siendo el paso más importante en remover errores y defectos. Existen muchas herramientas que ayudan al equipo de testing de un proyecto. Elegir las más adecuadas no es una tarea sencilla. A través de las herramientas de testing uno puede automatizar y optimizar las pruebas de los programas, evitando pérdidas de tiempo significativas. Con la estandarización y el uso de buenas prácticas (ej.: documentación precisa), las pruebas unitarias pueden ser muy efectivas, resultando en la reducción de esfuerzo y el incremento de la calidad de los productos.