Argentina $8,90 (recargo al interior $0,20)) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net 4

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

Bases de datos

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

ADO.NET
Conceptos fundamentales
SQL Server Management Studio

SQL Server 2005

Versiones - Instalación paso a paso

ISBN 978-987-1347-43-8
9 789871 347438 00004

Incluye CD-ROM #3

RedUSERS

COMUNIDAD DE TECNOLOGIA

¡SUMATE!

EL SITIO Nº1 DE TECNOLOGIA

Noticias al instante // Entrevistas y coberturas exclusivas // Análisis y opinión de los máximos referentes // Reviews de productos // Trucos para mejorar la productividad // Registrate, participa, y comparte tus opiniones

SUSCRIBITE

SIN CARGO A CUALQUIERA DE NUESTROS **NEWSLETTERS** Y RECIBÍ EN TU CORREO ELECTRÓNICO TODA LA INFORMACIÓN DEL UNIVERSO TECNOLÓGICO ACTUALIZADA **AL INSTANTE**

INGRESÁ A
redusers.com/suscribirse-al-newsletter
¡Y REGISTRATE YA!

Foros · Encuestas · Tutoriales · Agenda de eventos · Videos · ¡Y mucho más!

redusers.com

Seguinos en:
www.facebook.com/redusers
www.twitter.com/redusers
www.youtube.com/redusersvideos

Funciones

Hasta el momento, estuvimos viendo los operadores, las variables y las sentencias más importantes que tiene el lenguaje C#. A continuación, veremos las funciones, más conocidas como métodos, que nos permitirán estructurar mejor el código de las aplicaciones. De esta manera, podremos programar de una forma mucho más clara, evitando repetir código al invocar esta función en el lugar de la aplicación donde la necesitemos, con lo cual optimizaremos su funcionamiento.

Uso de métodos o funciones en C#

Los métodos o funciones son porciones de código que realizan determinadas acciones. Toman ciertos argumentos y devuelven un valor.
En C#, las funciones se deben declarar dentro de un objeto. Por lo general, llevan un nombre que las identifica para definir la tarea que cumplen.
Un ejemplo claro de función es **WriteLine()**, perteneciente a la clase **Console**, que permite escribir una línea con un valor, pregunta o resultado de operación, en la consola de aplicación.

```
{
   Console.WriteLine("Función que muestra este
   texto en la consola:");
}
```

Las funciones o métodos poseen un tipo de dato, un identificador y parámetros:

- **Tipo de dato**: Es un valor que la función devuelve al terminar.
- **Identificador**: Es el nombre por el cual llamaremos a la función en nuestra aplicación. Será utilizado para invocarla.
- **Parámetros**: Son las variables que recibirá el método para realizar las operaciones necesarias para las cuales fue creado. El listado de variables puede estar vacío o tener todos los elementos que queramos o precisemos; esto se entiende como opcional.

> Las funciones son porciones de código que realizan acciones. Toman ciertos argumentos y devuelven un valor.

Métodos estáticos

Existen métodos estáticos que son considerados como métodos de clase. Éstos deben tener un tipo de dato como prefijo de retorno, y llevar la palabra reservada Static. El método estático se llama por medio de un identificador de la clase, en vez del identificador del objeto, como sucede con las variables estáticas. Muchas clases de la librería BCL definen métodos estáticos, como el que vinimos utilizando hasta ahora en los ejemplos de código, el método **WriteLine**, perteneciente a la clase **Console**.

Retorno de valores

Como ya dijimos, los métodos pueden retornar valores o no. Si no devuelven ninguno, se especifica que el tipo de dato de retorno sea **void**.

Uso de parámetros

En la Figura 013 vemos de qué forma se declaran los parámetros en las funciones. Los parámetros son, básicamente, declaraciones de variables, separadas por una coma (en caso de que tengamos que utilizar más de uno).

ⓘ Función Main

Cada programa en C# debe tener una función Main(), en donde escribiremos el código principal para que nuestro programa se inicie.

Parámetros de salida

Es probable que en el desarrollo de un sistema, necesitemos que una función retorne más de un valor. En ese caso, podemos recurrir a los parámetros de salida. Existe un modificador llamado **out**, que nos permitirá realizar esta operación dentro de la función. Veamos un ejemplo para comprenderlo un poco mejor:

```
{
public Static void CalculoInteres(double
deposito, out double 30dias, out double
60dias)
  {
  //A 30 días 5% de interés
  30dias = (deposito * 1.05);
  //A 60 días 12% de interés
  60dias = (deposito * 1.12);
}

{
  //Declaro las variables que recibirán los
  valores de retorno de la función
  double A30, A60;
  CalculoInteres(5500, out A30, out A60);
  Console.WriteLine("$ 5500 a 30 días me dará: $
  " A30);
  Console.WriteLine("$ 5500 a 60 días me dará: $
  " A60);
}
}
```

Y el resultado obtenido en la consola de la aplicación será el siguiente:

```
$ 5500 a 30 días me dará: $ 5775
$ 5500 a 30 días me dará: $ 6160
```

Componentes de una función

```
1  static void main ()
2  {
3     Console.WriteLine("Viendo en acción nuestra funcion"}
4     int Resultado = SumarNumeros(32 + 35) ;
5     Console.WriteLine(Resultado):
6  }
7     static int SumarNumeros(int Valor1, int Valor2)
8     {
9       return Valor1+Valor2;
10    }
```

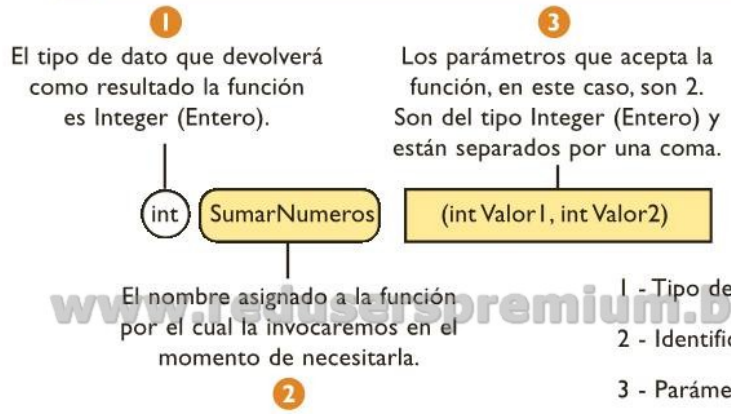


FIGURA 013 | Aquí podemos ver el código de un programa que contiene una función, y su composición.

Clases en C#

A continuación, veremos lo que se conoce como la estructura de datos más importante de este lenguaje: las clases.

Qué es una clase

Se conoce como **clase** a una estructura de datos utilizada para crear y definir nuestros propios tipos de datos, que nos ayudará a ampliar los propios del lenguaje.
En general, las clases se utilizan por medio de lo que llamamos **instancias**, que denominan **objetos**. Los objetos del mismo tipo pueden compartir una estructura común entre ellos, pero poseerán valores distintos en las variables que los componen (llamadas **miembros**).
Dentro de un programa, podemos crear objetos partiendo de una clase, y la única limitación que tenemos es la memoria que posea la computadora en la que se ejecute el software.

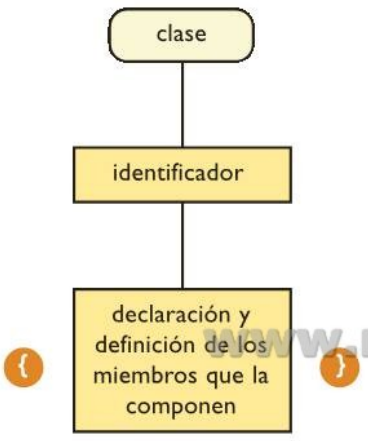
> Una clase es una estructura de datos utilizada para crear y definir nuestros propios tipos de datos.

Declarar una clase

Para declarar clases en C# utilizaremos archivos con extensión .CS, que compondrán nuestro proyecto o solución. Dentro de ellos podremos definir todas las clases que nos parezcan convenientes, pero siempre es recomendable utilizar un archivo por cada una que definamos para nuestro programa. Esto nos facilitará la lectura y

Estructura de una clase

Estructura gráfica de una clase



Estructura en código de una clase

Archivo nombredelaclase.CS

```
clase <identificador>

{
    // Cuerpo de la clase donde
    // definimos los miembros
    // que la compondrán

    tipo miembro1
    tipo miembro2
    tipo miembro3
    tipo miembro4
}
```


FIGURA 014 | Una misma clase, graficada según cómo sería su código.

modificación o corrección de cualquiera de ellas, en caso de que lo necesitemos.
Para crear una clase, en primer lugar debemos escribir la palabra reservada **class** y, luego, el **identificador**, que será el nombre con el cual la clase será identificada en el momento de usarla. Es preciso respetar que ese nombre no coincida con ninguna palabra reservada propia del lenguaje C#. El nombre otorgado tiene que ser bien claro y fácil de recordar. Generalmente, los nombres de las clases se eligen sobre la base del uso que vayamos a darles dentro del programa.
A continuación, veremos una clase graficada para mejorar su comprensión y, luego, la misma clase en código fuente. Dentro del "cuerpo" de la clase declaramos y definimos los miembros que la compondrán: Datos y Métodos.

Modificadores de acceso

Las variables o miembros de una clase se declaran de la misma manera que las variables convencionales en C#, pero pueden poseer algunos modificadores particulares o especiales. Es posible utilizar lo que se conoce como **modificador de acceso**, que regulará la manera de interactuar con una variable o el método desde otros objetos.
Siempre podremos acceder a las variables de una clase desde los métodos declarados en ella. Para otros métodos de otras clases, éstos nos impondrán limitaciones. Dichas limitaciones son:

- **internal**: La variable es declarada como interna. Podemos acceder a su contenido desde métodos de clases que se encuentran dentro del mismo assembly o ejecutable de nuestro proyecto.
- **private**: La variable es declarada como privada. Sólo podemos acceder a su contenido desde métodos de la clase, pero no desde métodos de clases derivadas. Si cuando declaramos la variable no especificamos lo contrario, será del tipo private.
- **protected**: La variable es declarada como protegida. No podemos acceder a su contenido desde objetos externos, pero sí, desde métodos de la clase y clases derivadas de ella.
- **protected internal**: La variable es declarada como protegida interna. Podemos acceder a su contenido desde métodos de clases que se encuentren dentro del mismo assembly o ejecutable, desde métodos que estén ubicados dentro de la misma clase y desde clases derivadas de ella.
- **public**: La variable es declarada como pública, de modo que podemos acceder a su contenido desde cualquier objeto de cualquier tipo.

Veamos cómo declarar una clase sencilla con sus miembros en el siguiente ejemplo:

```
class Persona
{
   public string nombre;
   public string apellido;
   public string DNI;
   public int edad;

   public Persona() {}
}
```

De acuerdo con el fragmento de código anterior, hemos definido la clase **Persona**. Ésta constituye un tipo de dato nuevo, que estará disponible para ser utilizado en nuestra aplicación.
Ahora, vamos a agregar los valores correspondientes a los miembros de la clase:

```
static void Main(string[] args)
{
      Persona p = new Persona();
      p.nombre = "Julián";
      p.apellido = "Luna";
      p.DNI = "46.850.185";
      p.edad = 2;
```

```
        Console.WriteLine("Descripción de persona :
         {0},{1}", p.nombre, p.apellido);
        Console.WriteLine("Otros datos : {0},{1}",
         p.DNI, p.edad + " años");
        Console.Read();
    }
```

Instanciación

Para poder usar un tipo de dato referenciado, debemos crearlo, mediante un proceso llamado instanciación. Ésta invoca el método que tiene el mismo nombre que la clase, denominado **constructor** de la clase.
Para definir una clase derivada de Persona, utilizaremos el operador **new**. Luego, las variables internas de la clase, llamadas **Nombre**, **Apellido**, **DNI** y **edad**, son modificadas desde fuera de la clase. Esto es posible ya que todas ellas están definidas como públicas en la clase original Persona.
Veamos la forma de hacerlo:

```
    Persona p = new Persona();

    p.nombre = "Julián";
    p.apellido = "Luna";
    p.DNI = "46.850.185";
    p.edad = 2;
```

Por último, las variables internas de la clase son leídas y mostradas en la consola en dos renglones diferentes: en el primero, el nombre y el apellido; en el segundo, su documento y edad actual:

```
    Console.WriteLine("Descripción de persona :
    {0},{1}", p.nombre, p.apellido);
    Console.WriteLine("Otros datos : {0},{1}",
    p.DNI, p.edad + " años");
    Console.Read();
```

> Las variables o miembros de una clase se declaran de la misma manera que las variables convencionales en C#.

Veamos cómo instanciar objetos, que posean la misma variable con valores distintos:

```
static void Main(string[] args)
{
   Persona pers1 = new Persona();
   Persona pers2 = new Persona();

   pers1.nombre = "Nicolás";
   pers2.nombre = "Julián";

   Console.WriteLine("El valor de la variable
   nombre del objeto pers1 es:
   {0}", pers1.nombre);
   Console.WriteLine("El valor de la variable
   nombre del objeto pers2 es:
   {0}", pers2.nombre);
   Console.Read();
}
```

En el ejemplo anterior, hemos utilizado la clase Persona, de la cual instanciamos las clases **pers1** y **pers2**. Ambos objetos son independientes entre sí. A **pers1** le asignamos el valor "Nicolás", y a **pers2**, "Julián".

Constantes de una clase

Cuando declaramos variables temporales del tipo constante, éstas tienen un valor asignado que no se puede modificar. Podemos declarar de la misma forma una variable del tipo constante dentro del cuerpo de una clase, que actuará de la misma manera que si fuese una variable temporal.

Las constantes declaradas dentro de clases pueden poseer los mismos modificadores de acceso que las variables (public, private, protected, etc.).

```
public class Persona
{
  public const string ciudadano = "Argentino";
  //…
}
```

A nuestra clase Persona le agregamos una constante llamada **ciudadano**, cuyo valor es "Argentino". Esta constante tendrá este valor previamente asignado, que no puede ser modificado, por lo cual será el mismo para todas las instancias que creemos sobre la base de la clase Persona. Notemos también que cuando declaramos la constante, también la inicializamos, es decir, le asignamos su valor. Esto es obligatorio cuando utilizamos este tipo de dato en las clases.
En las constantes dentro de una clase encontramos también una característica adicional: pueden ser accedidas o consultadas sin necesidad de instanciarlas, de modo que no necesitaremos crear un objeto de la clase Persona.

```
static void Main(string[] args)
{
  Persona pers1 = new Persona();
  Persona pers2 = new Persona();

  pers1.nombre = " Nicolás";
  pers2.nombre = " Julián";
  string origen = Persona.ciudadano;

  Console.WriteLine("El valor de la variable
  nombre del objeto pers1 es:
  {0}", pers1.nombre);
  Console.WriteLine("El valor de la variable
  nombre del objeto pers2 es:
  {0}", pers2.nombre);
  Console.WriteLine("El país de origen para los
  objetos pers1 y pers2 es:
  {0}", origen);
  Console.Read();
}
```

Asignación de valores

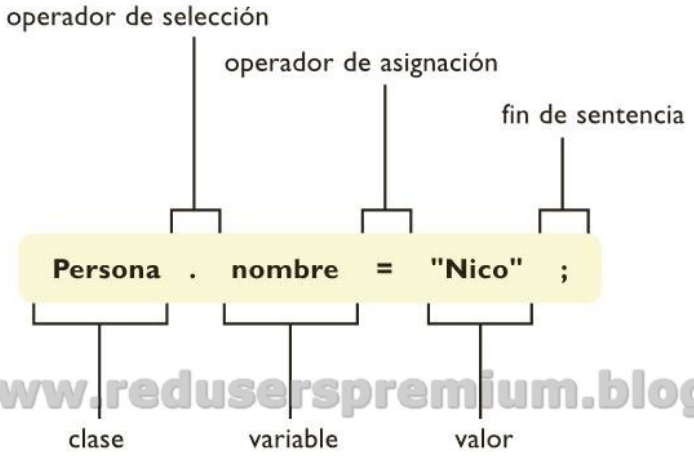


FIGURA 015 | La asignación de valores a variables de una clase se realiza prácticamente de la misma forma en que se le asigna valor a cualquier variable declarada fuera de ella.

En el ejemplo, luego de haber agregado a la Clase Persona la constante ciudadano, modificamos el código del programa, creando una variable **origen**, la cual obtendrá el valor de la constante **ciudadano**, directamente de la clase **Persona**.

Variables estáticas de una clase

Las variables estáticas, al igual que las constantes, pueden ser accedidas sin necesidad de instanciar un objeto, pero con la ventaja de que podemos modificar su valor como si fuese cualquier otra variable. Este tipo de variables, tal como sucede con las constantes o las variables convencionales, pueden tener los mismos modificadores de acceso (public, private, protected, etc.). Ahora agregaremos a la clase Persona una variable **Static**:

```
public class Persona
{
  public static int añoactual = 2006;
  public const string ciudadano = "Argentino";
  //…
}
```

Luego, cambiamos su valor desde nuestro programa, antes de mostrarla en pantalla:

```
class Program
{
  static void Main(string[] args)
  {
      Persona pers1 = new Persona();
      Persona pers2 = new Persona();

      Persona.anioActual = 2007;
      pers1.nombre = "Nicolás";
      pers2.nombre = "Julián";

      string origen = Persona.ciudadano;
      int muestroAnio= Persona.anioActual;

      Console.WriteLine("El valor de la variable
      nombre del objeto pers1 es:
      {0}", pers1.nombre);
      Console.WriteLine("El valor de la variable
      nombre del objeto pers2 es:
      {0}", pers2.nombre);
      Console.WriteLine("El valor de la variable
      estática añoActual es:
      {0}", muestroAnio);
      Console.WriteLine("El país de origen
      para los objetos pers1 y pers2 es:
      {0}", origen);
      Console.Read();
  }
}
```

Las constantes declaradas dentro de clases pueden poseer los mismos modificadores de acceso que las variables (public, private, protected, etc.).

FIGURA 016 | Podemos observar en la aplicación de consola la clase Persona funcionando, con todo lo referente a clases que aplicamos en ella.

PRACTICA02

Desarrollo de nuestra primera aplicación en C#

Estamos preparados para construir nuestro primer programa. Esta aplicación será para Windows, por lo que repasaremos las características de un desarrollo de este tipo.

Componentes de desarrollo de una aplicación para Windows

En C#, como en Visual Basic .NET, podemos crear tanto aplicaciones de consola como aplicaciones para Windows, entre otras.
El entorno de desarrollo IDE de ambos lenguajes es similar. Encontraremos un cuadro de herramientas con los controles que nos permitirán componer una aplicación con toda la funcionalidad que el entorno Windows requiere; el explorador de soluciones, la ventana de propiedades y el panel de resultado, entre otros elementos.

ⓘ Simulación

Con esta aplicación, simularemos el funcionamiento de un cajero automático. En ella ingresaremos nuestra clave, que nos permitirá consultar el saldo de nuestra cuenta bancaria, y extraer y depositar dinero. Este programa difiere mucho de una aplicación real de cajero automático, ya que por el momento no utilizaremos bases de datos, sino valores almacenados en las variables del programa.

Para dar comienzo a nuestro proyecto, en el entorno de desarrollo de C#, accedemos al menú Archivo/Nuevo Proyecto. La ventana que veremos será similar a la de Visual Basic .NET; entre las plantillas que se nos ofrecen, seleccionamos Aplicación para Windows. Como nombre, utilizaremos CajeroAutomatico. C# cargará todos los componentes necesarios para iniciar el proyecto. En el centro de la pantalla aparece un formulario o Form, llamado Form1. En la ventana de propiedades de Form1, cambiamos el valor Text por Banco Flores S.A. Si esta ventana no está accesible, presionamos la tecla <F4>, y en ella modificamos la propiedad **Text**. A continuación, agregamos los controles necesarios antes de comenzar a ingresar el código.
Los controles que utilizaremos son:

- 1 panel: Objeto contenedor que incluirá parte de los controles por agregar.
- 3 textbox: Cajas de texto donde ingresaremos datos.
- 4 buttons: Botones que nos permitirán confirmar la realización de las operaciones.
- 6 labels: Nos informarán en la pantalla los resultados de las operaciones que vamos a realizar.

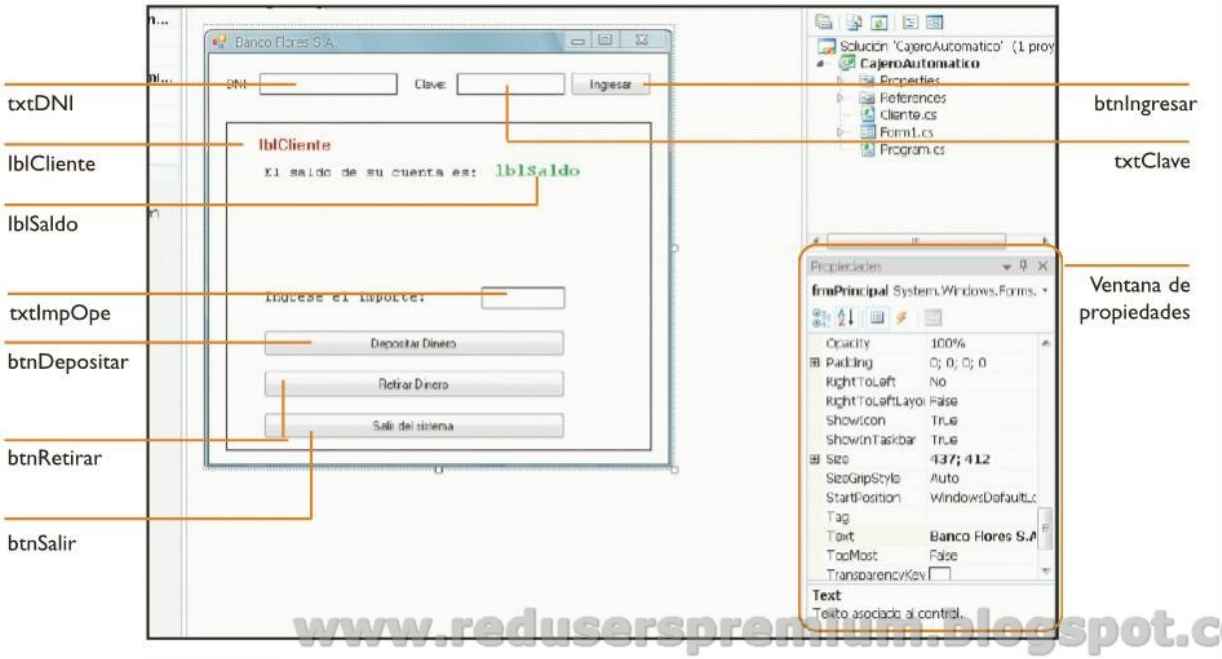
Para agregar dichos controles, basta con arrastrarlos desde el cuadro de herramientas hasta el Form1.

A continuación, le asignaremos el nombre que identificará a cada uno de los controles dentro del programa. Para hacerlo, seleccionamos el control y, en la ventana de propiedades, cambiamos su nombre. Veamos en la Figura 017 cómo llamaremos a cada uno de ellos.
Al control Panel1, le cambiamos la propiedad **Visible** a **false**. Ésta ocultará el control junto con todos los componentes que contenga. Cuando validamos el DNI y la Contraseña, si ambos datos son correctos, la propiedad cambiará a **true**.
Una vez que distribuimos los controles en la pantalla, empezaremos a agregar el código correspondiente en cada uno, para que la aplicación comience a tomar vida.

En C#, como en Visual Basic .NET, podemos crear aplicaciones tanto de consola como para Windows, entre otras.

En primer lugar, tenemos dos cajas de texto y un botón. En la primera caja ingresamos nuestro DNI, y en la segunda, la contraseña. Lluego presionamos el botón **Ingresar** para dar inicio al sistema.
Para este caso, el sistema verificará que el DNI sea igual a "123456" y que la contraseña

Guía Visual 001 | Cajero automático



Así debe verse nuestro Form1 luego de agregarle todos los controles necesarios.
La propiedad Name identifica a cada uno de los controles asignados en nuestro programa.

PARA AGREGAR CONTROLES A NUESTRA APLICACIÓN, BASTA CON ARRASTRARLOS DESDE EL CUADRO DE HERRAMIENTAS HASTA EL FORM DONDE ESTAMOS TRABAJANDO.

sea "7890". Cuando aprendamos el manejo con bases de datos, podremos modificar este programa, para que ambos datos puedan personalizarse. A continuación, en la ventana Explorador de Soluciones, agregamos un módulo de clase. Para hacerlo, sobre el nombre de la solución, realizamos clic derecho y, del menú, escogemos **Agregar/Módulo**. Elegimos la plantilla módulo, y como nombre le ponemos Clientes. El sistema generará un archivo llamado **clientes.cs** en nuestro proyecto, dentro del cual vamos a declarar una clase llamada **Clientes** con algunas propiedades que luego utilizaremos para mostrar en nuestro programa:

```
namespace WindowsApplication1
{
   class Cliente
   {
      public string apellido = "Luna";
      public string nombres ="Nicolás Julián";
      public string DNI ="123456";
      public string clave = "7890";
      public const string ca = "14300708/9605";
      public static double saldo = 2800;
   }
}
```

Lo particular de esta clase será la variable **ca** del tipo **constante**, que contendrá nuestro número de caja de ahorros, que no puede ser modificado, por lo menos, desde el cajero automático. También veamos que al momento de declarar la clase **Cliente**, las variables que la componen ya tienen su valor asignado.
Luego, en el Form de la aplicación creamos dos variables, que serán útiles para el correcto funcionamiento del programa: **resultado** será del tipo **Double**, y permitirá manejar el resultado de las operaciones que haremos; **importe** será del mismo tipo, y contendrá el valor monetario que retiraremos o depositaremos a través del programa.
Luego de crear las variables, instanciamos la clase Cliente como **CliActual**.


```
public partial class frmPrincipal : Form
{
   Double resultado = 0;
   Double importe = 0;
   Cliente CliActual = new Cliente();
```

Passwordchar

Para que una caja de texto muestre algún carácter comodín como contraseña, en la ventana de propiedades dispondremos de una denominada PasswordChar. En ella ingresamos el carácter que hará de comodín para que no se vea por pantalla la contraseña ingresada.

La validación

Hasta aquí, hemos creado las variables y clases necesarias para ingresar el código que hará funcionar nuestro cajero automático. A continuación, para poder operar en el cajero, debemos colocar un número de DNI y la clave, datos que serán validados según los ingresados en las variables de la clase Clientes.

El código que realizará la validación irá en el botón de ingreso, llamado **btnIngresar**. Al hacer clic en él, validará los datos registrados en la caja de texto **txtDNI** y la caja de texto **txtClave**, contra nuestra clase **Cliente**. Si coinciden, nos habilitará para ingresar en el cajero automático y operar. Para incluir el código en el botón, hacemos doble clic sobre él; se abrirá la ventana de código, donde podremos escribir el código para el evento clic.

Éstas son las líneas que tendremos que ingresar:

```
private void button1_Click(object sender,
EventArgs e)
{
  // Realiza una comparación si el txtDNI
  es igual a CliActual.DNI y txtClave es
  igual a CliActual.clave.
  if ((txtDNI.Text == CliActual.DNI) &
  (txtClave.Text == CliActual.clave))
  {
    panel1.Visible = true;
    string CajaAhorros = Cliente.ca;
    lblCliente.Text = CliActual.apellido
    + ", " + CliActual.nombres + ", CA: " +
    CajaAhorros;
    lblSaldo.Text = "$ " +
    Convert.ToString(Cliente.saldo);
```

Flujo de validación

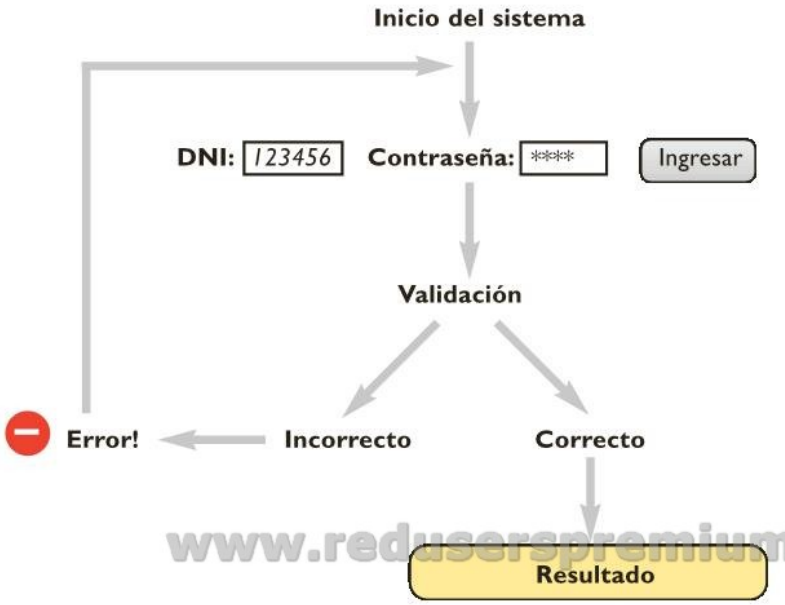


FIGURA 017 | En este caso, utilizamos la sentencia if-else.

```
        txtImpOpe.Focus();
      }
      // Si es incorrecto alguno o los dos
      valores, avisa con un mensaje de error
      y limpia los campos para reintentar.
      else
      {
        MessageBox.Show("Error en ingreso de
        clave o DNI");
        txtDNI.Text = "";
        txtClave.Text = "";
        txtDNI.Focus();
      }
    }
```

A continuación, crearemos una función llamada **RealizoOperacion**, que, a través de un parámetro denominado **tOpe** (del tipo **Integer)** que le pasaremos, utilizará la sentencia **Switch** comprobando el valor recibido en el parámetro, para realizar una transacción, ya sea extracción (1), depósito (2) o salir del sistema (3):

```
    private void RealizoOperacion(int tOpe)
    {
      switch (tOpe)
      {
        case 1:
          {
          if (txtImpOpe.Text != "")
            {
            importe =
            Convert.ToDouble(txtImpOpe.Text);
            resultado = (Cliente.saldo) +
            (importe);
            Cliente.saldo = resultado;
            lblSaldo.Text = "$ " +
            Convert.ToString(Cliente.saldo);
            txtImpOpe.Text = "";
            }
            return;
          }
        case 2:
          {
          if (txtImpOpe.Text != "")
            {
            importe =
            Convert.ToDouble(txtImpOpe.Text);
            resultado = (Cliente.saldo) -
            (importe);
            if (resultado >= 0)
              {
              Cliente.saldo = resultado;
              lblSaldo.Text = "$ " +
              Convert.ToString(Cliente.saldo);
              txtImpOpe.Text = "";
              }
            else
              {
              MessageBox.Show("No hay saldo
              suficiente para realizar esta
              operación.");
              }
            }
          else
            {
            MessageBox.Show("Error en el
            ingreso del importe. Verifíquelo.");
            txtImpOpe.Focus();
```

En Internet

En el sitio web, ***desarrollador. redusers.com***, podremos encontrar el código fuente completo de los ejemplos prácticos que se vayan desarrollando a lo largo de la obra, listos para descargar. Ésta es una herramienta realmente útil para comprender los conceptos y probar las posibles variantes.

LA FUNCIÓN MESSAGEBOX NOS PERMITE ENVIAR INFORMACIÓN CON MENSAJES A TRAVÉS DE PANTALLAS. UNA VEZ DECLARADA LA FUNCIÓN, SÓLO NOS QUEDA LLAMARLA DESDE EL EVENTO CLICK DE CADA BOTÓN..

```
            }
          }
          return;
        case 3:
          {
          DialogResult result;
          MessageBoxButtons buttons =
          MessageBoxButtons.YesNo;
          string mensaje = "¿Desea salir de
          las operaciones?";
          string tituloMB = CliActual.apellido +
          " " + CliActual.nombres;
          result = MessageBox.Show(this,
          mensaje, tituloMB, buttons);
          if (result == DialogResult.Yes)
            {
              {
              Cliente.saldo = 0;
              resultado = 0;
              importe = 0;
              txtImpOpe.Text = "";
              lblSaldo.Text = "";
              panel1.Visible = false;
              txtDNI.Text = "";
              txtClave.Text = "";
              txtDNI.Focus();
              MessageBox.Show("Gracias por
              utilizar nuestros servicios!");
              }
            }
          }
```

```
            return;
          }
        }
```

Dentro de esta función, realizamos operaciones básicas de suma y resta entre variables; luego, mostramos los resultados en la caja de texto o etiqueta correspondiente en la pantalla de nuestro programa.
Utilizamos, además, funciones que nos sirven para convertir valores dentro del lenguaje, como **Convert.ToString y Convert.ToDouble**. Estas funciones permiten convertir el valor de una caja de texto, que por defecto es de tipo **string**, a un valor **Double** y viceversa. El uso de estas funciones es **Convert.ToString(VariableNumerica) y Convert.ToDouble(cajadetexto.text)**.
Otra función utilizada es **MessageBox**, que nos permite informar con mensajes a través de la pantalla. Su sintaxis es:

MessageBox.Show("Mensaje para el usuario.", "Título de la ventana de usuario").

Una vez declarada la función con su correspondiente código, sólo nos queda llamarla desde el evento **Click** de cada botón, pasándole el parámetro correspondiente para que realice la operación.

SIEMPRE ES BUENO MANTENER COMENTADO EL CÓDIGO DE NUESTRO PROGRAMA, PARA FACILITAR EL DISEÑO DE LA DOCUMENTACIÓN, PERO NO CONVIENE COMENTAR CADA LÍNEA ESCRITA, YA QUE DIFICULTARÍA SU LECTURA.

```
private void button2_Click(object sender,
EventArgs e)
{
  // Parámetro valor 1: Depositar Dinero
  RealizoOperacion(1);
}

private void button3_Click(object sender,
EventArgs e)
{
  // Parámetro valor 2: Retirar Dinero
  RealizoOperacion(2);
}

private void button4_Click(object
sender, EventArgs e)
{
  // Parámetro valor 3: Salir del sistema.
  RealizoOperacion(3);
}
```

Con estas líneas de código, nuestro programa ya está funcionando. Sólo nos queda probarlo para verificar que lo haga correctamente. A continuación, nos introduciremos en el mundo de las bases de datos.

FIGURA 018 | Ya podemos comenzar a utilizar nuestra primera aplicación Windows hecha en C#.

Bases de datos

Almacenar y administrar información

3

Contenidos

En la actualidad, resulta prácticamente imprescindible comprender qué es una base de datos y cómo trabajar con ella al momento de programar un sistema. En este capítulo, desarrollaremos esta temática a fondo.

Temas tratados

- Introducción a las bases de datos
- Instalación y configuración de SQL Server 2005
- Fundamentos sobre bases de datos
- Pasos por seguir para llevar adelante la normalización
- El lenguaje SQL desde cero
- Cómo gestionar bases de datos

Bases de datos

Las bases de datos nos permitirán organizar la información y, además, consultarla, actualizarla y administrarla desde un programa.

Introducción a las bases de datos

Empezaremos por entender qué es exactamente una base de datos y cómo gestionar los valores que en ella se almacenan, a través de la plataforma .NET.

- Estructura de la información
- El lenguaje SQL
- Transact-SQL

Instalación y configuración de SQL Server 2005

Conoceremos cada una de las herramientas y componentes que posee, y cuáles son los pasos que se deben seguir para llevar adelante una correcta instalación de todo el entorno.

- Las ediciones de SQL
- El proceso de instalación
- Los componentes

Fundamentos sobre bases de datos

Nos introduciremos en los aspectos esenciales que hacen a la creación, diseño e implementación de bases de datos que sean robustas y confiables para almacenar información.

- Sistema relacional
- Diagrama entidad-relación
- Normalización

El lenguaje SQL desde cero

Conoceremos en detalle la sintaxis de este poderoso lenguaje, y cómo emplearlo para gestionar la información almacenada en una base de datos.

- Realizar consultas
- Modificar y actualizar datos
- Agrupar y aplicar funciones
- Unir tablas

Ejercicios prácticos

Mediante la implementación de un sistema completo desarrollado con C# y Visual Basic, aprenderemos a emplear los principales comandos de SQL Server.

- Primeros pasos
- Desarrollo del código
- Puesta en marcha del sistema

Bases de datos

Vamos a introducirnos en uno de los temas más interesantes del mundo de la programación: la administración de bases de datos.

Empecemos por entender qué es exactamente una base de datos. Se trata de un conjunto de información organizada que está almacenada en algún soporte no volátil, y que puede ser consultada, actualizada y administrada mediante un grupo de programas. Éstos reciben el nombre de Sistema de Gestión de Bases de Datos (SGDB o DBRMS por sus siglas en inglés) y ofrecen al usuario las herramientas necesarias para realizar operaciones sobre la información almacenada, tales como agregar, modificar, borrar, consultar, cruzar; administrar la seguridad y realizar operaciones de backup, entre otras opciones.
Existen distintos tipos de bases de datos según la organización que se les dé: jerárquicas, de red, relacionales, orientadas a objetos, etc. Actualmente, las más utilizadas para el desarrollo de aplicaciones de negocios son las relacionales, entre las que se encuentran Microsoft Access, Microsoft SQL Server y MySQL.

> Una base de datos es un conjunto de información organizada que está almacenada en algún soporte no volátil.

Estructura de la información

En las bases de datos relacionales, la información se almacena en tablas, que están compuestas por filas y columnas, lo que, comúnmente,

Composición

FIGURA 001 | Ésta es la composición de una base de datos y su contenido.

en informática se conoce como registros y campos. A su vez, las tablas pueden estar relacionadas con otras, de ahí que sean relacionales. Estas relaciones se establecen según la lógica del negocio, es decir, de las necesidades con respecto a los datos que se están almacenando. Por ejemplo, tenemos una tabla llamada Facturas, donde se guarda la facturación de un negocio; y otra llamada Clientes, donde están almacenados los datos de las personas o empresas que realizaron compras. Ambas tablas estarán relacionadas, de modo que cada vez que el cliente Juan Pérez compre algún producto, al emitirle la factura correspondiente, no tenga que pedirle otra vez sus datos. Esto tiene que ver con el concepto de normalización, que veremos más adelante.
Como dijimos, una tabla está compuesta por registros y campos. Entonces, si tomamos la tabla Clientes, un registro correspondería a todos los datos de un cliente, mientras que un campo sería un dato particular de ese registro, por ejemplo, la dirección.
En la Figura 002 podemos ver un ejemplo con datos almacenados.

El lenguaje de consulta SQL

Una base de datos no sería muy útil si permitiera sólo guardar datos y no, consultarlos. Para esto existe un lenguaje de consulta llamado **SQL**, sigla de *Structured Query Language* o lenguaje estructurado de consultas. Está basado en un estándar llamado **ANSI SQL**, que fue evolucionando con los años. Conozcamos cómo funciona.

Transact-SQL

Microsoft se basa en el estándar ANSI-SQL, y lo amplió para poder otorgar al usuario mayor control sobre SQL Server. Esta variante se denominó Transact-SQL o T-SQL.
T-SQL nos permite no sólo realizar consultas sobre los datos: también podemos modificarlos, actualizarlos o eliminarlos. Además, da la posibilidad de efectuar operaciones sobre tablas, relaciones e índices; crear bases de datos; administrar permisos a los usuarios, y realizar procesamientos de cadenas de datos, operaciones matemáticas, y más.
Todas estas opciones representaron una mejora importante para el lenguaje SQL, y permitieron crear en los lenguajes de programación un SQL embebido, que potencia mucho más cada uno de ellos, ya que ayuda al desarrollador a diseñar un sistema con toda la eficiencia necesaria en la consulta y operación con bases de datos.
Un ejemplo de una consulta sencilla con T-SQL, que devuelve todos los registros que contiene la tabla Clientes, puede ser el siguiente:

```
select * from clientes
```

En la página siguiente, veremos con más claridad algunos comandos muy utilizados de Transact-SQL, su función y un breve ejemplo (Tabla 1). Muchos de los otros comandos no detallados en la tabla los trataremos más

Clientes : Tabla

	Nro_Cliente	Nombre	CUIT
▶	1	Juan Pérez	20-25860589-8
	2	GMS S.R.L.	30-25854475-0
	3	Mónica Gutierrez	27-28557832-5
	4	F-digital S.A.	20-24516107-8
	5	Del Campo Natural	27-24752516-6
*	(Autonumérico)		

FIGURA 002 | Composición de una tabla.

adelante, a medida que comencemos las prácticas con SQL.

Claves primarias

Una clave primaria nos permite identificar de manera unívoca un registro dentro de una tabla. Esto es necesario para poder establecer una relación con otra tabla. Por ejemplo, en una tabla Clientes, para hacer referencia a uno específico, debemos tener definida una clave primaria única para cada uno, con lo cual nos aseguramos de acceder al registro necesario. Por lo general, esto se hace por medio de uno o más campos que contienen valores únicos para identificar a un cliente desde el resto de las tablas que componen la base de datos, y que requieren hacer referencia a él.

Claves foráneas

Para hacer referencia a un registro en particular de la tabla Clientes desde la tabla Facturas, necesitamos incluir en ella la clave primaria del cliente. Entonces, se llama clave foránea al campo o campos que contiene la clave primaria de otra tabla, con el fin de identificar un registro externo. En el caso de la tabla Facturas, necesitamos indicar a qué cliente le fue emitida, por lo que debemos incluir como clave foránea el número de cliente.

Una clave primaria nos permite identificar de manera unívoca un registro dentro de una tabla. Esto es necesario para poder establecer una relación con otra tabla.

Índices

Cuando realizamos búsquedas dentro de una tabla, el motor de base de datos recorre los registros de manera secuencial hasta encontrar el dato solicitado. Esta acción no es óptima, en especial, cuando los volúmenes de datos son muy grandes, porque se pueden producir demoras importantes en devolver los resultados. Una forma de solucionar este problema es por medio de los índices. El índice

Tabla 1 | Transact-SQL

Comando	Función que cumple	Ejemplo
SELECT	Permite seleccionar registros de una tabla.	SELECT * FROM Clientes;
INSERT	Permite agregar registros a una tabla.	INSERT INTO Clientes VALUES ('Cathedral Soft', '30-25051996-4')
UPDATE	Actualiza registros en una tabla.	UPDATE Clientes SET Nombre ='CATHEDRAL SOFTWARE' WHERE Nombre ='CATHEDRAL SOFT';
DELETE	Elimina registros de una tabla.	DELETE FROM Clientes WHERE Nombre ='CATHEDRAL SOFT';
DECLARE	Permite crear variables.	DECLARE NomCli CURSOR FOR SELECT * FROM Clientes;
SET	Permite asignarle un valor a una variable.	SET TIME ZONE 'Argentina/Buenos Aires';
CREATE TABLE	Permite crear una tabla y sus campos.	CREATE TABLE Facturas_Clientes (NroFactura LONG, IDCliente LONG DEFAULT 0, Fecha DATE DEFAULT 'DateTime');

de una tabla se puede comparar con el de un libro. Si en un texto no contáramos con un índice de capítulos, para encontrar uno en particular deberíamos recorrer hoja por hoja hasta dar con el que precisamos. Esto nos demandaría mucho más tiempo que si utilizáramos el índice, dado que en él buscaríamos el capítulo en cuestión para averiguar en qué página se encuentra y, con este dato, iríamos directamente a esa página, con el consiguiente ahorro de tiempo que esto implica. En una tabla los índices funcionan exactamente de la misma manera: por ejemplo, si dentro de la tabla Clientes tenemos que buscar por nombre de manera frecuente, lo más conveniente será crear un índice en este campo. El motor de base de datos, internamente, crea una lista ordenada de los datos que contiene esa columna y mantiene una referencia acerca de a qué registro pertenece. Cuando se solicita el registro cuyo nombre de cliente es Amalia Fernández, el motor efectúa la búsqueda sobre el índice y obtiene la posición del registro en la tabla; recién entonces accede a ese registro y devuelve los datos solicitados por el usuario.

Claves primarias y foráneas

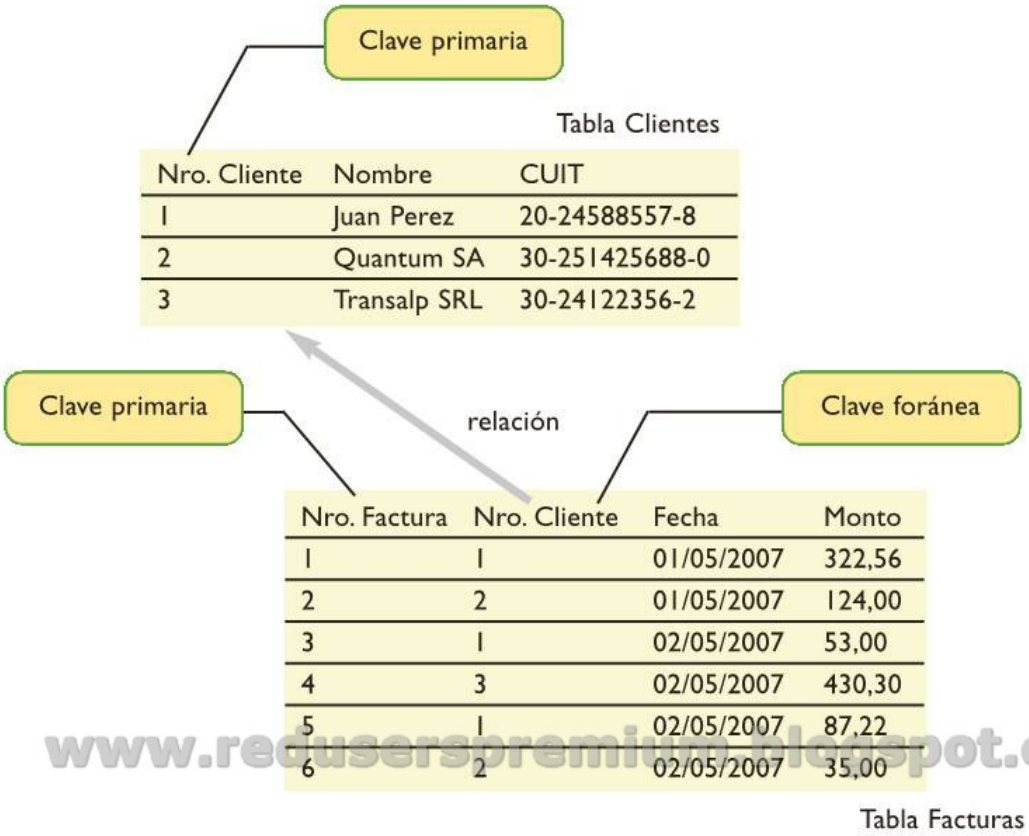


FIGURA 003 | Las claves primarias y foráneas son unas de las características principales de las bases de datos relacionales.

SQL Server 2005

Conoceremos en detalle una de las herramientas más poderosas del entorno .NET.

SQL Server es un sistema para administración de bases de datos que posee una arquitectura cliente/servidor. Utiliza el lenguaje de consulta Transact-SQL para recibir comandos desde los clientes que se conectan a él, y ofrece una gran variedad de herramientas y servicios para desarrollar y administrar bases de datos de distintos tamaños y complejidad.

Para comprender bien de qué se trata un sistema cliente/servidor, podemos decir que el servidor ofrece uno o varios servicios, que el cliente utiliza. En un ejemplo de la vida real, un servidor sería un surtidor de gasolina, que brinda al automovilista, que actuaría como cliente, el servicio de expendio de combustible. Un mismo surtidor puede servir a **n** cantidad de vehículos. De la misma manera, SQL Server ofrece sus servicios a los clientes que se conectan, para que éstos puedan realizar las operaciones que precisen.

Ediciones de SQL Server 2005

SQL Server 2005 está disponible en varios tipos de ediciones, cada una de las cuales presenta distintas características y está destinada a cubrir diferentes tipos de necesidades: desde grandes bases de datos empresariales distribuidas en una red corporativa a escala mundial, hasta una pequeña base de datos monousuario alojada en la misma computadora donde se ejecuta la aplicación que la usa. Las ediciones de SQL Server 2005 disponibles y sus características más destacables son las siguientes:

- **SQL Server 2005 Enterprise Edition**: Es la más completa y, a la vez, la más costosa. Está pensada para ser utilizada en aplicaciones de misión crítica, grandes almacenes de datos (*data warehouse*), análisis complejos y minería de datos. Soporta hasta 64 procesadores, y no está limitada en cuanto a la cantidad de memoria soportada ni al tamaño que puede llegar a tener cada base de datos.
- **SQL Server 2005 Standard Edition**: Está pensada para cubrir las necesidades de la mayoría de los negocios de pequeño y mediano tamaño. Su costo es menor que el de la edición Enterprise y no posee algunas de las características más avanzadas de ésta, como Parallel Indexing, Online Restore y Fast Recovery, entre otras. A pesar de esto, es una excelente opción para cubrir la mayoría de los requerimientos de un negocio. Soporta hasta cuatro procesadores, y no está limitada en cuanto a la cantidad de memoria que utiliza ni al tamaño máximo que puede llegar a tener una base de datos.

SQL Server es un sistema para administración de bases de datos que posee una arquitectura cliente/servidor. Utiliza el lenguaje de consulta Transact-SQL para recibir comandos desde los clientes que se conectan a él.

- **SQL Server 2005 Workgroup Edition**: Es ideal para negocios pequeños o soluciones departamentales, para ser utilizada por servidores web o aplicaciones de datos. Se puede actualizar fácilmente tanto a la edición Standard como a la Enterprise. Soporta hasta dos procesadores y 3 GB de RAM, aunque no tiene límite en lo que respecta al tamaño que puede llegar a tener una base de datos.
- **SQL Server 2005 Express Edition**: Reemplaza a MSDE (*Microsoft Data Engine*), es gratuita y de distribución libre, liviana e ideal para ser utilizada en soluciones departamentales, prototipos y aplicaciones Web de baja complejidad. Soporta un procesador, 2 GB de RAM, y el tamaño máximo que puede llegar a tener una base de datos es de 4 GB.
- **SQL Server 2005 Compact Edition**: Pensada para usar en dispositivos móviles, se integra con SQL Server 2005 y Visual Studio 2005, y permite desarrollar aplicaciones para Pocket PCs, smart phones y cualquier dispositivo que emplee Windows Mobile como sistema operativo.
- **SQL Server 2005 Developer Edition**: Posee exactamente las mismas características que la edición Enterprise; la única diferencia concreta radica en que la licencia de uso la destina sólo a ambientes de desarrollo, y no está permitido usarla en ambientes de producción.

Componentes de SQL Server 2005

SQL Server 2005, además del motor de base de datos, posee un conjunto de herramientas adicionales que permiten potenciar la productividad del usuario y realizar reportes, análisis de datos, notificaciones de eventos especiales, exportación a distintos formatos y muchas tareas más.

- SQL Server Analysis Services (SSAS): Es una herramienta enfocada al análisis de datos. Se utiliza para el desarrollo de datamarts y data warehouses, permite aplicar técnicas como el data mining, y trabaja con cubos, dimensiones y valores.
- SQL Server Reporting Services (SSRS): Se utiliza para generar reportes que se pueden consultar con un navegador Web como Internet Explorer. También permite convertirlos en múltiples formatos y tiene un esquema de seguridad centralizado.
- SQL Server Integration Services (SSIS): Brinda la posibilidad de conectarse a una gran variedad de orígenes de datos, y de realizar extracciones, ejecutar código SQL y automatizar tareas en las que se manipulan flujos de datos.
- SQL Server Notification Services (SSNS): Es un servicio de notificaciones en el cual un suscriptor puede recibir un aviso cuando un determinado valor cumple con cierta condición programada de antemano. Soporta múltiples destinos para los mensajes, por ejemplo, un teléfono celular, una casilla de mail, etc.

FIGURA 004 | A continuación, repasaremos todo el proceso de instalación de SQL Server. Podremos utilizar para esto la versión Express incluida en esta obra.

Proceso de instalación

Antes de instalar SQL Server 2005, debemos verificar que el equipo en el cual lo haremos cumpla con los requisitos mínimos necesarios de hardware y software para llevar a cabo con éxito este proceso.

Al iniciar la instalación de SQL Server 2005 Express Edition, el instalador nos mostrará el Contrato de licencia de usuario final; luego, realizará la comprobación de los requisitos mínimos del sistema para que la aplicación se instale y ejecute correctamente. Al finalizar, nos mostrará en pantalla cada una de las verificaciones realizadas. En la lista se presentan tres columnas: Acción, Estado y Mensaje. La columna Acción describe la comprobación efectuada, Estado indica si la verificación resultó satisfactoria o no, y Mensaje informa en detalle el problema encontrado, en caso de que haya alguno. Para continuar, presionamos el botón Siguiente.

FIGURA 006 | Comprobación de requisitos del sistema para instalar SQL Server 2005 Express Edition.

El próximo paso nos permite seleccionar los componentes de SQL Server que se instalarán. Al momento de seleccionarlos, hay que tener en cuenta cuáles vamos a usar y cuáles no, ya que cada uno consume recursos del equipo que pueden ser requeridos por otras

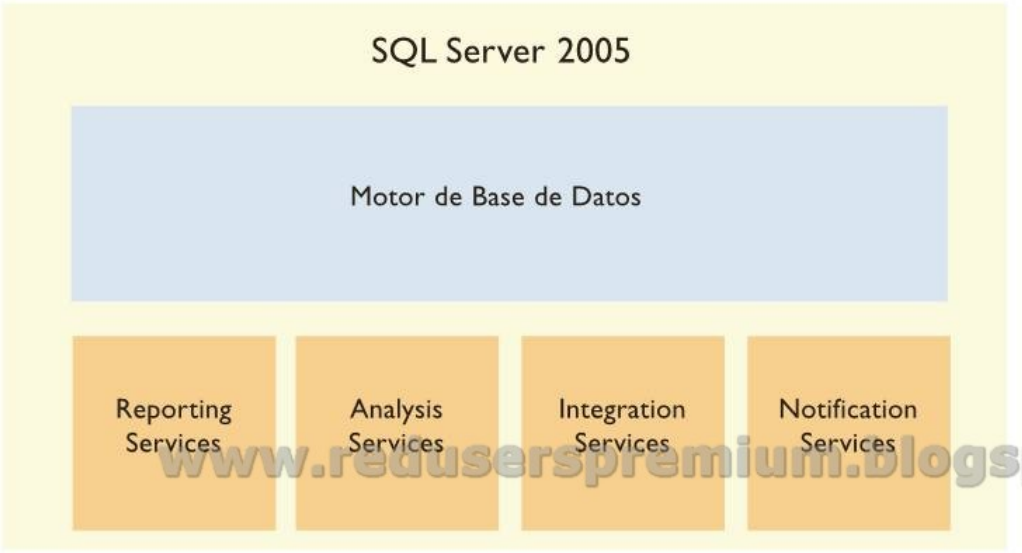


FIGURA 006 | Además de servidor de bases de datos, SQL Server ofrece servicios adicionales que amplían las posibilidades al momento de crear soluciones de datos.

aplicaciones. Si disponemos de una computadora con mucha capacidad y memoria, nos conviene instalar todos los componentes, para que queden disponibles en el futuro.
A continuación, elegimos la instancia que vamos a instalar. Una instancia de SQL Server contiene todos los componentes del servidor. Podemos crear más de una instancia en una misma máquina, y entonces será como tener instalados varios servidores SQL en un equipo. Las ediciones Enterprise y Developer soportan hasta 50 instancias, mientras que el resto admite hasta 16.
En el cuadro de diálogo podemos seleccionar si vamos a instalar una instancia predeterminada, que no requiere la asignación de un nombre, o tenemos la opción de especificar puntualmente cómo se llamará la nueva instancia que se está instalando.

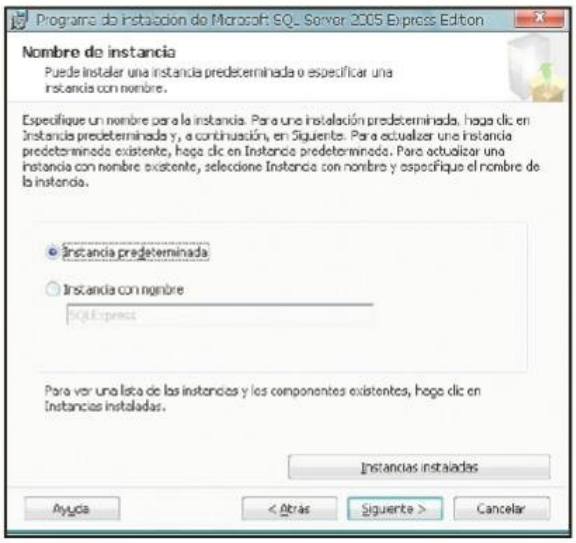


FIGURA 007 | Las instancias permiten tener varios servidores SQL en una misma computadora.

Utilizando el botón Instancias instaladas, es posible averiguar cuáles ya están presentes. Para continuar, presionamos Siguiente.
El próximo paso es especificar qué cuenta de servicio vamos a utilizar, proporcionando nombre de usuario, dominio y contraseña. Esta cuenta puede utilizarse para todos los servicios, o se puede asignar una diferente por cada uno.
Luego, veremos un sumario con los componentes seleccionados que se van a instalar, y el programa nos informará que está listo para comenzar el proceso, que se activará cuando lo confirmemos haciendo clic en **Instalar**.
Una vez concluida la instalación, veremos un resumen con los componentes instalados. Podemos consultar el Registro de resumen para saber si se produjo algún error durante esa etapa. Ahora SQL Server 2005 Express Edition ya está instalado y listo para usar. En caso de que se requiera reiniciar el equipo, se mostrará un mensaje; es muy recomendable hacerlo de inmediato para evitar que se produzca algún error y que todos los componentes instalados queden correctamente registrados en Windows.

Requerimientos mínimos

Antes de instalar SQL Server 2005, se debe verificar que el equipo cumpla con los requisitos mínimos necesarios de hardware y de software para llevar a cabo el proceso con éxito. Los requerimientos son los siguientes:

- Enterprise/Developer Edition: Microrocesador Pentium III de 600 MHz o superior (también soporta procesadores de 64 bits), 512 MB de memoria RAM y 350 MB de espacio libre en el disco.
- Workgroup/Standard Edition: Microprocesador Pentium III de 600 MHz o superior, 512 MB de memoria RAM y 350 MB de espacio libre en el disco.
- Express Edition: Microprocesador Pentium III de 600 MHz o superior, 192 MB de memoria RAM, 350 MB de espacio en disco.

USERS

CURSOS INTENSIVOS

CURSOS.REDUSERS.COM

SALIDA LABORAL

Los temas más importantes del universo de la tecnología desarrollados con la mayor profundidad y con un despliegue visual de alto impacto: Explicaciones teóricas, procedimientos paso a paso, videotutoriales, infografías y muchos recursos mas.

Brinda las habilidades necesarias para planificar, instalar y administrar redes de computadoras de forma profesional. Basada principalmente en tecnologías Cisco, es una obra actual, que busca cubrir la necesidad creciente de formar profesionales.

- 25 Fascículos
- 600 Páginas
- 3 CDs / 1 Libro

- 25 Fascículos
- 600 Páginas
- 4 CDs

Curso para dominar las principales herramientas del paquete Adobe CS3 y conocer los mejores secretos para diseñar de manera profesional. Ideal para quienes se desempeñan en diseño, publicidad, productos gráficos o sitios web.

Obra teórica y práctica que brinda las habilidades necesarias para convertirse en un profesional en composición, animación y VFX (efectos especiales).

- 25 Fascículos
- 600 Páginas
- 2 CDs / 1 DVD / 1 Libro

- 26 Fascículos
- 600 Páginas
- 2 DVDs / 2 Libros

Obra ideal para ingresar en el apasionante universo del diseño web y utilizar Internet para una profesión rentable. Elaborada por los máximos referentes en el área, con infografías y explicaciones muy didácticas.

Llegamos a todo el mundo con OCA * y DHL **

usershop@redusers.com +54 (011) 4110-8700

usershop.redusers.com.ar

** Válido en todo el mundo excepto Argentina. * Sólo válido para la República Argentina

Argentina $8,90 (recargo al interior $0,20)) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net 4

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

Bases de datos

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

ADO.NET
Conceptos fundamentales
SQL Server Management Studio

SQL Server 2005

Versiones - Instalación paso a paso

ISBN 978-987-1347-43-8
9 789871 347438 00004

Incluye CD-ROM #3