UNIVERSIDAD PRIVADA-DE-TACNA

INGENIERIA DE SISTEMAS

TITULO:

Trabajo de Unidad - Mapeo Objeto Relacional

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)

Índice

1. PROBLEMATICA

1.1. Objetivos:

- Aplicar las Pruebas Unitarias y Edpoints en una aplicacion.

1.2. Equipos, materiales, programas y recursos utilizados:

- Computadora con sistema operativo Windows 10.
- Microsoft Visual Studio 2017
- Microsoft SQL Server 2016
- Net Framework 4.6.1
- Base de datos SchoolDB

1.3. Problematica:

- Los servicios bancarios que ofrecen los cajeros automáticos han aumentado de forma notable en los últimos años.cajeros informen de la comisión y los gastos de la operación antes de que se realice.

  La primera opción de la que dispone el cliente, y que recomiendan tanto las entidades como los organismos supervisores ante cualquier problema que se produzca al hacer uso del cajero, es la de utilizar de forma inmediata el número del centro de atención al usuario del banco. el que aparece en el cajero o el de la red.

2. MARCO TEORICO

2.1. Entity Framework

– Entity Framework (EF) es un mapeador relacional de objetos (O/RM) probado y probado para .NET con muchos años de desarrollo y estabilización de características.
Es la tecnología de acceso a datos recomendada por Microsoft para nuevas aplicaciones.

– Como O/RM, EF reduce la discrepancia de impedancia entre los mundos relacionales y orientados a objetos, permitiendo a los desarrolladores escribir aplicaciones que interactúan con datos almacenados en bases de datos relacionales utilizando objetos .NET de tipo fuerte que representan el dominio de la aplicación y eliminando la necesidad para una gran parte del código de "plumbing"de acceso a datos que normalmente necesitan escribir.

– EF implementa muchas características populares de O/RM:
- Mapeo de clases de entidad POCO que no dependen de ningún tipo de EF
- Seguimiento automático de cambios
- Resolución de identidad y Unidad de Trabajo.
- Carga ansiosa, perezosa y explícita.
- Traducción de consultas fuertemente tipadas utilizando LINQ
- Capacidades de mapeo enriquecidas, incluyendo soporte para:

  - Relaciones uno a uno, uno a muchos y muchos a muchos

  - Herencia (tabla por jerarquía, tabla por tipo y tabla por clase concreta)

  - Tipos complejos

  - Procedimientos almacenados

- Un diseñador visual para crear modelos de entidad.
- Una experiencia de Çódigo Primero"para crear modelos de entidad al escribir código.
- Los modelos pueden generarse a partir de bases de datos existentes y luego editarse manualmente, o pueden crearse desde cero y luego usarse para generar nuevas bases de datos.
- Integración con modelos de aplicaciones de .NET Framework, incluido ASP.NET, y mediante enlace de datos, con WPF y WinForms.
- Conectividad de base de datos basada en ADO.NET y numerosos proveedores disponibles para conectarse a SQL Server, Oracle, MySQL, SQLite, PostgreSQL, DB2, etc.

2.2. Pruebas Unitarias

– Una prueba unitaria se utiliza para comprobar que un método concreto del código de producción funciona correctamente, probar las regresiones o realizar pruebas relacionadas (buddy) o de humo. Una prueba por orden se utiliza para ejecutar otras pruebas en un orden especificado.

3. DESARROLLO

Parte 1: Consultando Datos

3.1. ANALISIS

3.2. DISEÑO

3.3. PRUEBAS

- Crear un proyecto de pruebas a nuestra solución.
- Dentro de la clase, agregamos el siguiente metodo ObtenerAlEstudianteConIDUno.

```
public void ObtenerAlEstudianteConIDUno()
{
    var ctx = new SchoolDBEntities();
    var student = ctx.Students.Find(1);
}
```

- Agregamos el siguiente metodo BuscarAlPrimerEstudianteConElNombreBill.

```
public void BuscarAlPrimerEstudianteConElNombreBill()
{
    using (var ctx = new SchoolDBEntities())
    {
        var student = (from s in ctx.Students
                       where s.StudentName == "Bill"
                       select s).FirstOrDefault<Student>();
    }
}
```

- Agregamos el siguiente metodo BuscarEstudiantesAgrupadosPorEstandar.
- Agregamos el siguiente metodo ObetenerListadoDeEstudiantesOrdenadosPorNombre.
- Agregamos el siguiente metodo BuscarTodosLostudiantesConElEstandarUno.

Parte 2: Guardando Datos

- Agregar al proyecto de pruebas una clase de pruebas TestUnit02, agregar el método InsertarEstudianteSatisfactoriamente, tomar como referencia el siguiente código.
- Agregar el método ActualizarElPrimerEstudianteSatisfactoriamente, tomar como referencia el siguiente código.

```
public void BuscarEstudiantesAgrupadosPorEstandar()
{
    using (var ctx = new SchoolDBEntities())
    {
        var students = ctx.Students.GroupBy(s => s.StandardId);
        foreach (var groupItem in students)
        {
            Console.WriteLine(groupItem.Key);
            foreach (var stud in groupItem)
                Console.WriteLine(stud.StudentId);
        }
    }

}
```

```
public void ObetenerListadoDeEstudiantesOrdenadosPorNombre()
{
    using (var ctx = new SchoolDBEntities())
    {
        var students = from s in ctx.Students
                       orderby s.StudentName ascending
                       select s;
    }
}
```

- Agregar el método EliminarElPrimerEstudianteSatisfactoriamente, tomar como referencia el siguiente código.
- Agregar el método: AgregarTresEstudiantesSatisfactoriamente, tomar como referencia el siguiente código:
- Finalmente agregar el método de pruebas: EliminarTresEstudiantesSatisfactoriamente, tomar como referencia el siguiente código:

```
public void BuscarTodosLostudiantesConElEstandarUno()
{
    using (var ctx = new SchoolDBEntities())
    {
    var anonymousObjResult = from s in ctx.Students
    where s.StandardId == 1
    select new {
    Id = s.StudentId,
    Name = s.StudentName
        };
    }
}
```

```
0 referencias
public void TInsertarEstudianteSatisfactoriamente()
{
    using (var context = new SchoolDBEntities())
    {
        var std = new Student()
        {
            FirstName = "Bill",
            LastName = "Gates"
        };
        context.Students.Add(std);
        context.SaveChanges();
    }
```

```
0 referencias
public void ActualizarElPrimerEstudianteSatisfactoriamente()
{
    using (var context = new SchoolDBEntities())
    {
        var std = context.Students.First<Student>();
        std.FirstName = "Steve";
        context.SaveChanges();
    }
}
0 referencias
```

```
public void EliminarElPrimerEstudianteSatisfactoriamente()
{
    using (var context = new SchoolDBEntities())
    {
        var std = context.Students.First<Student>();
        context.Students.Remove(std);
        context.SaveChanges();
    }
}
```

```
0 referencias
public void AgregarTresEstudiantesSatisfactoriamente()
{
    IList<Student> newStudents = new List<Student>();
    {
        new Student() { StudentName = "Steve" };
        new Student() { StudentName = "Bill" };
        new Student() { StudentName = "James" };
    };
    using (var context = new SchoolDBEntities())
    {
        context.Students.AddRange(newStudents);
        context.SaveChanges();
    }
}
```

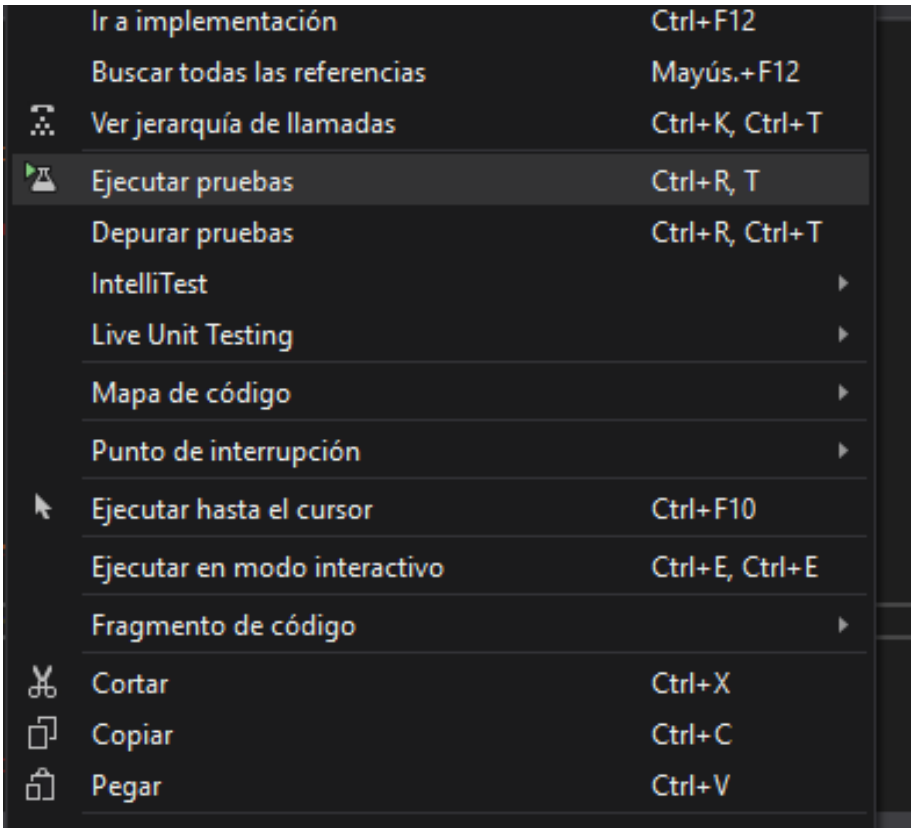
```
public void EliminarTresEstudiantesSatisfactoriamente()
{
    IList<Student> studentsToRemove = new List<Student>();
    {
        new Student() { StudentID = 1, StudentName = "Steve" };
        new Student() { StudentID = 2, StudentName = "Bill" };
        new Student() { StudentID = 3, StudentName = "James" };
    };

using (var context = new SchoolDBEntities())
    {
     context.Students.RemoveRange(studentsToRemove);
    context.SaveChanges();
    }
```

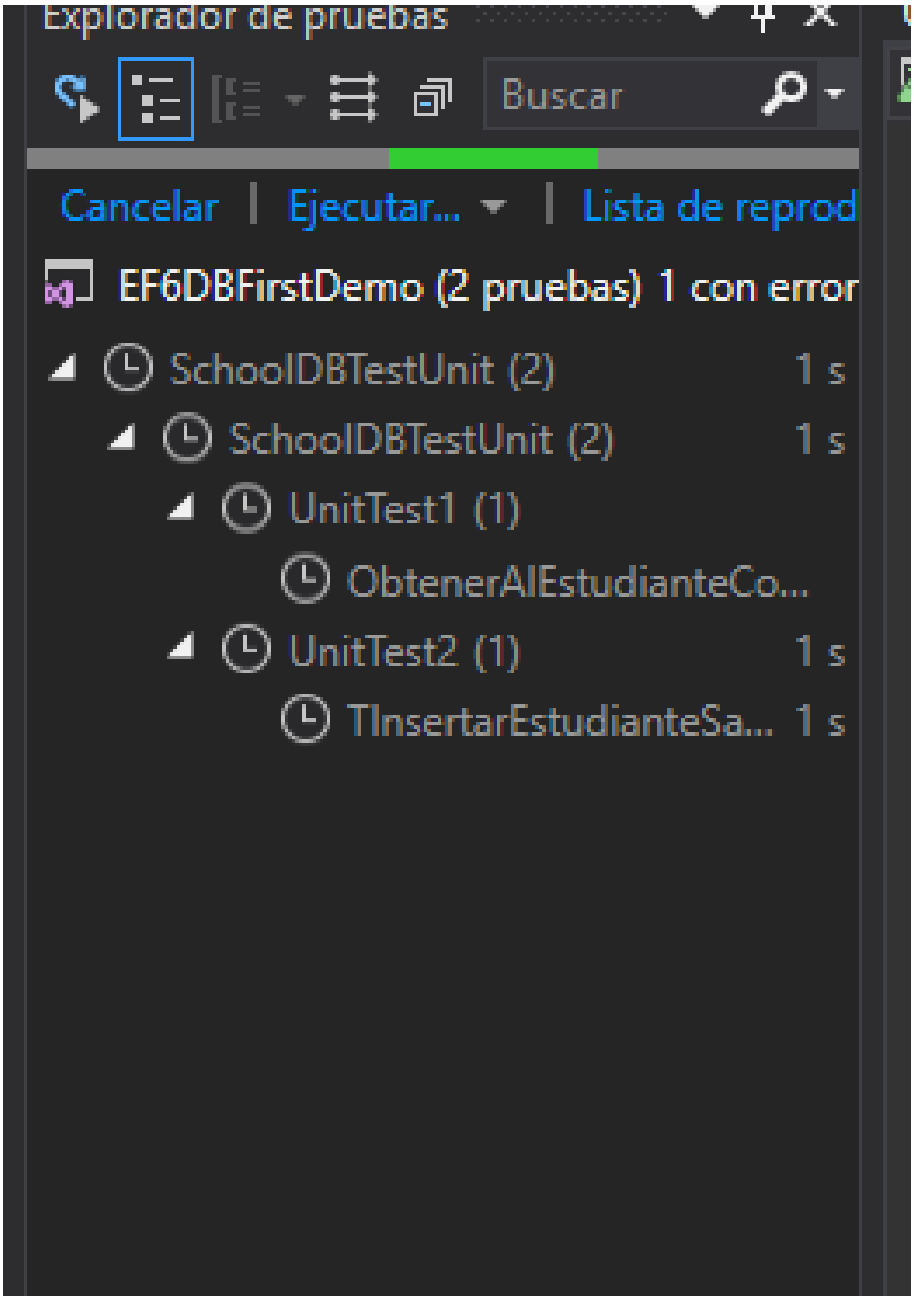
4. ANALISIS E INTERPRETACION DE RESULTADOS

-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.


Explorador de pruebas
Buscar
Cancelar | Ejecutar... | Lista de reprod
EF6DBFirstDemo (2 pruebas) 1 con error
SchoolDBTestUnit (2) 1 s
SchoolDBTestUnit (2) 1 s
UnitTest1 (1)
ObtenerAlEstudianteCo...
UnitTest2 (1) 1 s
TInsertarEstudianteSa... 1 s

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.

6. REFERENCIAS