

UNIVERSIDAD PRIVADA-DE-TACNA



INGENIERIA DE SISTEMAS

TITULO:

INFORME DE LABORATORIO N° 02

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. INFORMACIÓN GENERAL	1
1.1. Objetivos:	1
1.2. Equipos, materiales, programas y recursos utilizados:	1
2. MARCO TEORICO	2
2.1. Entity Framework	2
2.2. Pruebas Unitarias	3
3. PROCEDIMIENTO	4
4. ANALISIS E INTERPRETACION DE RESULTADOS	8
5. CONCLUSIONES	11
6. REFERENCIAS	12

1. INFORMACIÓN GENERAL

1.1. Objetivos:

- Realizar y aplicacion de Pruebas Unitarias en un proyecto C Sharp

1.2. Equipos, materiales, programas y recursos utilizados:

- Computadora
- Sistema operativo Windows 10
- Microsoft Visual Studio 2017
- Microsoft SQL Server 2016, 2017
- Net Framework 4.6.1
- Base de datos SchoolDB

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 "Código 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.
- Características:
- Para que una prueba unitaria tenga la calidad suficiente se deben cumplir los siguientes requisitos:
- Automatizable
No debería requerirse una intervención manual. Esto es especialmente útil para integración continua.
- Completas
Deben cubrir la mayor cantidad de código.
- Repetibles o Reutilizables
No se deben crear pruebas que sólo puedan ser ejecutadas una sola vez. También es útil para integración continua.
- Independientes
La ejecución de una prueba no debe afectar a la ejecución de otra.
- Profesionales
Las pruebas deben ser consideradas igual que el código, con la misma profesionalidad, documentación, etc. Aunque estos requisitos no tienen que ser cumplidos al pie de la letra, se recomienda seguirlos o de lo contrario las pruebas pierden parte de su función.

3. PROCEDIMIENTO

Parte 1: Consultando Datos

- 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.

```
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);
            }
        }
    }
}
```

- Agregamos el siguiente metodo ObetenerListadoDeEstudiantesOrdenadosPorNombre.

```

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

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
                                };
    }
}

```

- 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.

```

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

```

- Agregar el método ActualizarElPrimerEstudianteSatisfactoriamente, tomar como referencia el siguiente código.
- Agregar el método EliminarElPrimerEstudianteSatisfactoriamente, tomar como referencia el siguiente código.

```

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

```

```

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

```

- Agregar el método: AgregarTresEstudiantesSatisfactoriamente, tomar como referencia el siguiente código:

```

Oreferencias
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();
    }
}

```

- Finalmente agregar el método de pruebas: EliminarTresEstudiantesSatisfactoriamente, tomar como referencia el siguiente código:


```
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" };
    };

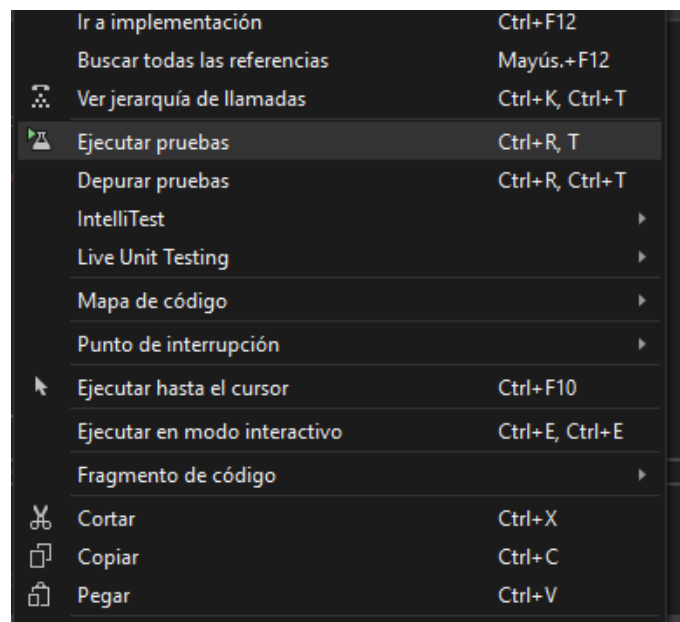
    using (var context = new SchoolDBEntities())
    {
        context.Students.RemoveRange(studentsToRemove);
        context.SaveChanges();
    }
}
```

4. ANALISIS E INTERPRETACION DE RESULTADOS

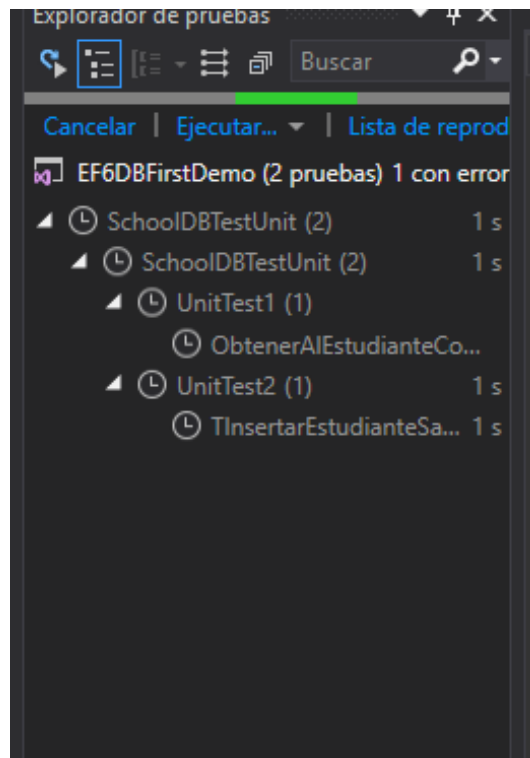
-Al compilar el proyecto de prueba, las pruebas aparecen en el Explorador de pruebas. Si el Explorador de pruebas no está visible, elija Prueba en el menú de Visual Studio, elija Ventanas y, después, Explorador de pruebas.

-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.



TEST 1

¿Cuál es la sentencia SQL generada del lado del gestor de base de datos?
La sentencia por cada metodo generado fue:

- ObtenerAlEstudianteConIDUno

```

Audit Login      -- network protocol: LPC set quote...
RPC:Completed    exec sp_executesql N'SELECT TOP (2)...

exec sp_executesql N'SELECT TOP (2)
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Extent1].[Height] AS [Height],
[Extent1].[Weight] AS [Weight],
[Extent1].[RowVersion] AS [RowVersion],
[Extent1].[GradeId] AS [GradeId]
FROM [dbo].[Students] AS [Extent1]
WHERE [Extent1].[StudentID] = @p0,N'@p0 int',@p0=1

```

- BuscarAlPrimerEstudianteConElNombreBill

```

SQL:BatchCompleted  SELECT TOP (1) [Extent1].[Stu...

SELECT TOP (1)
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Extent1].[Height] AS [Height],
[Extent1].[Weight] AS [Weight],
[Extent1].[RowVersion] AS [RowVersion],
[Extent1].[GradeId] AS [GradeId]
FROM [dbo].[Students] AS [Extent1]
WHERE N'Bill' = [Extent1].[StudentName]

```

- BuscarEstudiantesAgrupadosPorGrade

```
SQL:BatchCompleted SELECT [Project2].[GradeId] A... .Net SqlClie... sharonf... Mic
<
SELECT
[Project2].[GradeId] AS [GradeId],
[Project2].[C1] AS [C1],
[Project2].[StudentID] AS [StudentID],
[Project2].[StudentName] AS [StudentName],
[Project2].[Height] AS [Height],
[Project2].[Weight] AS [Weight],
[Project2].[RowVersion] AS [RowVersion],
[Project2].[GradeId1] AS [GradeId1]
FROM ( SELECT
[Distinct1].[GradeId] AS [GradeId],
[Extent2].[StudentID] AS [StudentID],
[Extent2].[StudentName] AS [StudentName],
[Extent2].[Height] AS [Height],
[Extent2].[Weight] AS [Weight],
[Extent2].[RowVersion] AS [RowVersion],
[Extent2].[GradeId] AS [GradeId1],
CASE WHEN ([Extent2].[StudentID] IS NULL) THEN CAST(NULL AS int) ELSE 1 END AS [C1]
FROM (SELECT DISTINCT
[Extent1].[GradeId] AS [GradeId]
FROM [dbo].[Students] AS [Extent1] ) AS [Distinct1]
LEFT OUTER JOIN [dbo].[Students] AS [Extent2] ON [Distinct1].[GradeId] = [Extent2].[GradeId]
) AS [Project2]
ORDER BY [Project2].[GradeId] ASC, [Project2].[C1] ASC
```

- ObtenerListadoDeEstudiantesOrdenadosPorNombre
Ascendente

```
SQL:BatchCompleted SELECT [Extent1].[StudentID] AS [StudentID],
<
SELECT
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Extent1].[Height] AS [Height],
[Extent1].[Weight] AS [Weight],
[Extent1].[RowVersion] AS [RowVersion],
[Extent1].[GradeId] AS [GradeId]
FROM [dbo].[Students] AS [Extent1]
ORDER BY [Extent1].[StudentName] ASC
```

Descendente

```
SQL:BatchCompleted SELECT [Extent1].[StudentID] AS [StudentID],
<
SELECT
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName],
[Extent1].[Height] AS [Height],
[Extent1].[Weight] AS [Weight],
[Extent1].[RowVersion] AS [RowVersion],
[Extent1].[GradeId] AS [GradeId]
FROM [dbo].[Students] AS [Extent1]
ORDER BY [Extent1].[StudentName] DESC
```

- BuscarTodosLostudiantesConElGradoUno

```
SQL:BatchCompleted SELECT [Extent1].[StudentID] AS [StudentID],
<
SELECT
[Extent1].[StudentID] AS [StudentID],
[Extent1].[StudentName] AS [StudentName]
FROM [dbo].[Students] AS [Extent1]
WHERE 1 = [Extent1].[GradeId]
```

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

Barrientos, Pablo Andrés (04 de 2014). Enfoque para pruebas de unidad basado en la generación aleatoria de objetos. p. 101. Consultado el 28 de abril de 2014. “The art of sotware Testing” - Glenford J. Myers “Tema 4: Corrección y Robustez en Java” - <http://dis.um.es/bmoros/privado/apuntes/Curso08-09/POO4-Java-0809.pdf>