I.E.S ABASTOS

Test de Sistemas Gestores de Bases de Datos

Autor: **Jaime Aguilá Sánchez**

Ciclo Desarrollo de Aplicaciones Multiplataforma

Memoria del Proyecto de DAM

IES Abastos. Curso 2015/16. Grupo 7U. 27 de Mayo de 2016

Tutor individual: Alfredo Oltra

Non nova, sed nove.

Contenido

Tabla de ilustraciones

1. Introducción

1.1. Definición de SGBD SQL, NoSQL y ORM

1.1.1. Bases de datos

La primera vez que se escuchó el término base de datos fue en un congreso celebrado en California en **1963**. Una base de datos es un cúmulo de información que se encuentra reunida o estructurada.

Desde el punto de vista informático, una base de datos es un sistema formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos y un conjunto de programas que manipulen ese conjunto de datos origen de la base de datos.

1.1.2. SQL

En la época de los **80** se creó un lenguaje de consultas de acceso a bases de datos que permite realizar consultas para recuperar información de interés de una base de datos y realizar cambios de manera sencilla, es el **SQL**; aparte de examinar grandes cantidades de información y deja detallar varios tipos de operaciones frente a la misma información.

Durante este tiempo **SQL** comenzó a ser el modelo de la industria; las bases de datos relacionales con su sistema de tablas (compuesta por filas y columnas) pudieron competir con las bases jerárquicas y de red, como consecuencia de que su nivel de programación era sencillo y su nivel de programación era relativamente bajo.

En la década de los **90** la investigación en bases de datos giró en torno a las bases de datos orientadas a objetos. Las cuales han tenido bastante éxito a la hora de gestionar datos complejos en los campos donde las bases de datos relacionales no han podido desarrollarse de forma eficiente.

Sistema Gestor de Base de Datos. Un Sistema Gestor de Base de Datos (SGBD, en inglés DBMS: DataBase Management System) es un sistema de software que permite la definición de bases de datos; así como la elección de las estructuras de datos necesarios para el almacenamiento y búsqueda de los datos, ya sea de forma interactiva o a través de un lenguaje de programación. Un SGBD relacional es un modelo de datos que facilita a los usuarios describir los datos que serán almacenados en la base de datos junto con un grupo de operaciones para manejar los datos.

Los **SGBD** relacionales son una herramienta efectiva que permite a varios usuarios acceder a los datos al mismo tiempo. Brindan facilidades eficientes y un grupo de funciones con el objetivo de garantizar la confidencialidad, la calidad, la seguridad y la integridad de los datos que contienen, así como un acceso fácil y eficiente a los mismos.

Típicamente las bases de datos relacionales modernas han mostrado poca eficiencia en determinadas aplicaciones que usan los datos de forma intensiva, incluyendo el indexado de un gran número de documentos, la presentación de páginas en sitios que tienen gran tráfico, y en sitios de streaming audiovisual. Las implementaciones típicas de RDBMS se han

afinado o bien para una cantidad pequeña pero frecuente de lecturas y escrituras o para un gran conjunto de transacciones que tiene pocos accesos de escritura.

1.1.3. NoSQL

A finales de los 90, aparece el termino **NoSQL** (a veces llamado "no sólo SQL") es una amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico del sistema de gestión de bases de datos relacionales (RDBMS) en aspectos importantes, el más destacado es que no usan SQL como el principal lenguaje de consultas. Los datos almacenados no requieren estructuras fijas como tablas, normalmente no soportan operaciones JOIN, ni garantizan completamente ACID (atomicidad, consistencia, aislamiento y durabilidad), y habitualmente escalan bien horizontalmente. Los sistemas **NoSQL** se denominan a veces "no sólo SQL" para subrayar el hecho de que también pueden soportar lenguajes de consulta de tipo SQL.

Los sistemas de bases de datos NoSQL crecieron con las principales compañías de Internet, como Google, Amazon, Twitter y Facebook. Estas tenían que enfrentarse a desafíos con el tratamiento de datos que las tradicionales RDBMS no solucionaban. Con el crecimiento de la web en tiempo real existía una necesidad de proporcionar información procesada a partir de grandes volúmenes de datos que tenían unas estructuras horizontales más o menos similares. Estas compañías se dieron cuenta de que el rendimiento y sus propiedades de tiempo real eran más importantes que la coherencia, en la que las bases de datos relacionales tradicionales dedicaban una gran cantidad de tiempo de proceso.

En ese sentido, a menudo, las bases de datos NoSQL están altamente optimizadas para las operaciones recuperar y agregar, y normalmente no ofrecen mucho más que la funcionalidad de almacenar los registros (p.ej. almacenamiento clave-valor). La pérdida de flexibilidad en tiempo de ejecución, comparado con los sistemas SQL clásicos, se ve compensada por ganancias significativas en escalabilidad y rendimiento cuando se trata con ciertos modelos de datos.

SQL	NoSQL	ORM
Ventajas		
Control de redundancia	Gran escalabilidad horizontal	Rapidez en el desarrollo.
Estandarización.	Pueden manejar enormes cantidades de datos.	Abstracción de la base de datos.
Integridad	No generan cuellos de botella.	Reutilización.
Seguridad	Escalamiento sencillo.	Seguridad.
Rapidez de desarrollo	Diferentes DBs NoSQL para diferentes proyectos.	Mantenimiento del código.
Mantenimiento: Cambios en la estructura de datos sin cambiar los programas que lo usan.	Se ejecutan en clusters de máquinas baratas.	Lenguaje propio para realizar las consultas.
Desventajas		
Tamaño	No están lo suficientemente maduros para algunas empresas.	Tiempo utilizado en el aprendizaje.
Susceptibilidad de fallas	Limitaciones de Inteligencia de Negocios.	Aplicaciones algo mas lentas.
Compatibilidad en la recuperación a fallas	La falta de experiencia.	
Puedo llegar a trabajar en forma "lenta" debido a la cantidad de verificaciones que debe hacer.	Problemas de compatibilidad.	

1 Ventajas y Desventajas. SQL, NoSQL y ORM

1.1.4. ORM

Object-Relational Mapping, o lo que es lo mismo, mapeo de objeto-relacional, es un modelo de programación que consiste en la transformación de las tablas de una base de datos, en una serie de entidades que simplifiquen las tareas básicas de acceso a los datos para el programador.

La ventaja principal de estos sistemas es que reducen la cantidad de código necesario para lograr lo que se conoce como una persistencia de objetos.

1.2. Definición del proyecto

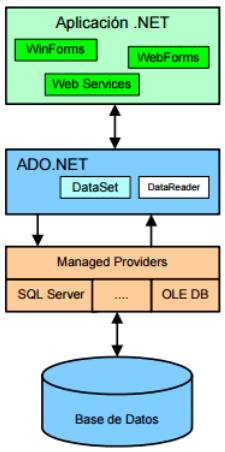
Este proyecto surge como una propuesta de la profesora Cristina Ausina, con un enfoque genérico como una aplicación para poder probar distintos Sistemas Gestores de Bases de Datos (SGBD) centrándonos sobre todo en poder compararlos entre sí, SGBD SQL, NoSQL, y ORMs.

Viendo la dificultad de poder comparar sistemas que tienen finalidades distintas y no comparten la forma de uso, en SQL los datos comparten un esquema con una definición de datos, y tienen un lenguaje de consulta estándar, el **SQL,** mientras que los sistemas **NoSQL**, no tienen un lenguaje de consulta estándar, además no comparten una estructura fija en los datos, con lo cual no tiene mucho sentido comparar estos dos sistemas entre sí.

Con respecto a los **ORM**, siempre van a ser más lentos, ya que son una capa sobre un SGBD, su utilidad es encapsular funcionalidad y acabar obteniendo el esquema en objetos dentro del lenguaje, con ello tampoco tiene mucho sentido compararlos entre sí.

Lo que si tenía sentido era centrarnos en los SGBD de tipo SQL, y no solo para compararlos entre ellos (SQL Server, MySQL, PostgreSQL), sino para poder comparar distintos conectores y distintas consultas de un mismo SGBD.

Dentro de .Net existen distintas tecnologías para acceder a las bases de datos **MDAC** (Microsoft Data Access Components), **OLE DB** (Object Linking and Embedding for Databases), **DAO** (ActiveX Data Objects), **RDS** (Remote Data Services), **ADO** (ActiveX Data Objects), vamos a centrarnos en **ADO .NET** que es el más reciente, y en la actualidad es el principal de .NET.



2 Esquema ADO.NET

ADO.NET es un conjunto de componentes del software que pueden ser usados por los programadores para acceder a datos y a servicios de datos. Es una parte de la biblioteca de clases base que están incluidas en el Microsoft .NET Framework. Es comúnmente usado por los programadores para acceder y para modificar los datos almacenados en un Sistema Gestor de Bases de Datos Relacionales, aunque también puede ser usado para acceder a datos en fuentes no relacionales.

El modelo **ADO.NET** está dividido en dos grupos:

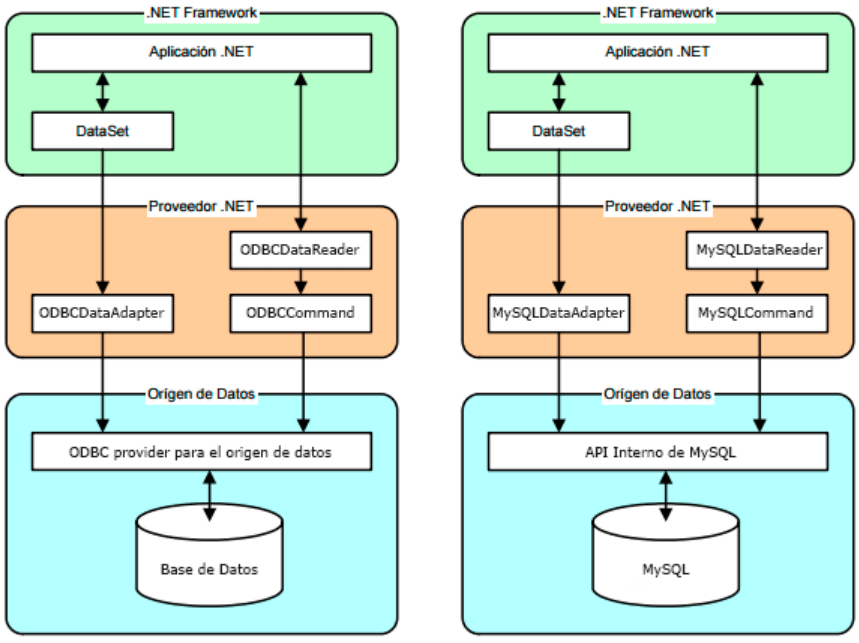
- Los proveedores de datos o Managed Data Providers
- Los contenedores de datos, que aunque están vinculados a los orígenes de datos, son independientes de ellos.

Los **Managed Provider** o .NET Managed Data Provider es el proveedor de datos de la plataforma .NET que permite conectar a un origen de datos con una aplicación para recuperar y modificar información. También este proveedor sirve de puente entre el origen de datos y el objeto más importante de ADO.NET, el DataSet.

ADO.NET, provee por defecto proveedores para OleDB, ODBC, y SQL Server y permite el uso de proveedores de terceros que implementen el estándar ADO.NET.

Se va a desarrollar la aplicación para que admita la conexión por **ODBC** para permitir cualquier Base de Datos genérica que este instalada en el sistema, y el conector nativo para **MySQL** ya que en la actualidad es una de las bases de datos más utilizadas, sobre todo a nivel de aplicaciones Web.

Aparte, dejaremos la aplicación preparada para implementar en el futuro el proveedor SQL server o cualquier otro proveedor que implemente el estándar ADO.NET.



3 Conectores ADO.NET del proyecto

1.3. Casuísticas típicas

1.3.1. Tipo conector

En muchas ocasiones nos hemos encontrado con aplicaciones, tanto de escritorio como aplicaciones Web, en las que la conexión con la base de datos se realiza mediante conexión **ODBC**, siendo el SGBD MySQL, Oracle o incluso SQL Server, teniendo estos SGBD **conectores nativos para ADO.NET**, y siempre nos ha surgido la duda, realmente merecerá la pena el realizar los cambios necesarios para usar un conector nativo en vez de usarlo de tipo **ODBC**, dependiendo de cómo este desarrollada la aplicación, podemos tener el cambio realizado con una mínima inversión.

Pero siempre nos queda la duda de si este cambio habrá merecido realmente la pena, o simplemente es un cambio anecdótico.

1.3.2. Apertura y cierre de conexiones

En otro caso, nos encontramos ante una aplicación Web legada, nos ponen en antecedentes comentándonos que es una aplicación desarrollada en PHP, que en un principio usaba MySQL pero que con el tiempo tuvieron que migrarla a Oracle, ya que MySQL no era capaz de soportar el tráfico y que incluso determinadas partes más pesadas, tuvieron que migrarlas de PHP en Apache, a Java en Tomcat ya que continuaban teniendo problemas.

Revisando el código, se descubre que los desarrolladores anteriores en cada método que necesitaban lanzar una consulta a la base de datos, previamente abrían la conexión, y al acabar el método cerraban dicha conexión, si una página necesitaba obtener los datos de varias tablas, y los métodos iban llamando a otros métodos, se iban realizando aperturas de conexiones en cascada a la base de datos, llegando en algunos casos a que la misma petición de un cliente tuviera entre 20 y 30 conexiones abiertas, además de existir secciones de la web donde no se liberaban todas las conexiones quedando muchas de ellas en memoria.

Aunque no hace falta una aplicación para ver que no reutilizar recursos, y no liberarlos de forma correcta, es una mala práctica, sí que queremos poder comprobar el costo de abrir y cerrar la conexión con la base de datos en diferentes puntos, a nivel global, a nivel de hilo o a nivel de consulta.

1.3.3. Campos de ordenación y filtros

En otros casos, desarrollamos la aplicación en base a unas especificaciones previas, todo parece funcionar correctamente, la llevamos a producción, el cliente comienza a usarla, y con el tiempo ocurren estas dos situaciones:

- En cuanto la base de datos comienza a crecer, determinadas secciones se hacen cada vez más lentas.

- En un apartado en particular, los clientes quieren poder filtrar u ordenar por un campo que en principio no estaba previsto, y al permitir dicha acción, el rendimiento cae en picado.

En estos casos, puede ser algo muy sencillo de solucionar, generando los índices necesarios, para no perder rendimiento, o puede ser algo más difícil de encontrar, teniendo que cambiar el orden o la forma de unir determinadas consultas.

En cualquiera de los casos, puede ser muy útil contar con una aplicación que nos permita lanzan una batería de consultas sobre una base de datos, guardarnos los resultados y poder repetir el proceso conforme vayamos realizando cambios, para poder en base a estas medidas tomadas, saber si los cambios son beneficiosos.

1.3.4. Diferentes sentencias SQL con mismos resultados

En otros casos, nos podemos encontrar con consultas SQL, que vienen formadas por la unión de muchas tablas, en estos casos, el orden de unión de dichas tablas y la forma en las que las unimos, puede ser un factor clave para que el sistema funcione bien o se vuelva tremendamente lento en cuanto crecen los datos.

2. Objetivos y planificación

2.1. Objetivos

El objetivo principal es la creación de una aplicación que nos permita configurar una serie de test, que permitan ejecutarse contra distintos conectores de Bases de Datos y recoja los resultados en ficheros XML, que nos permitan abrir estos en formato PDF con los resultados convertidos a gráficos que nos permitan procesar la información de una forma más visual.

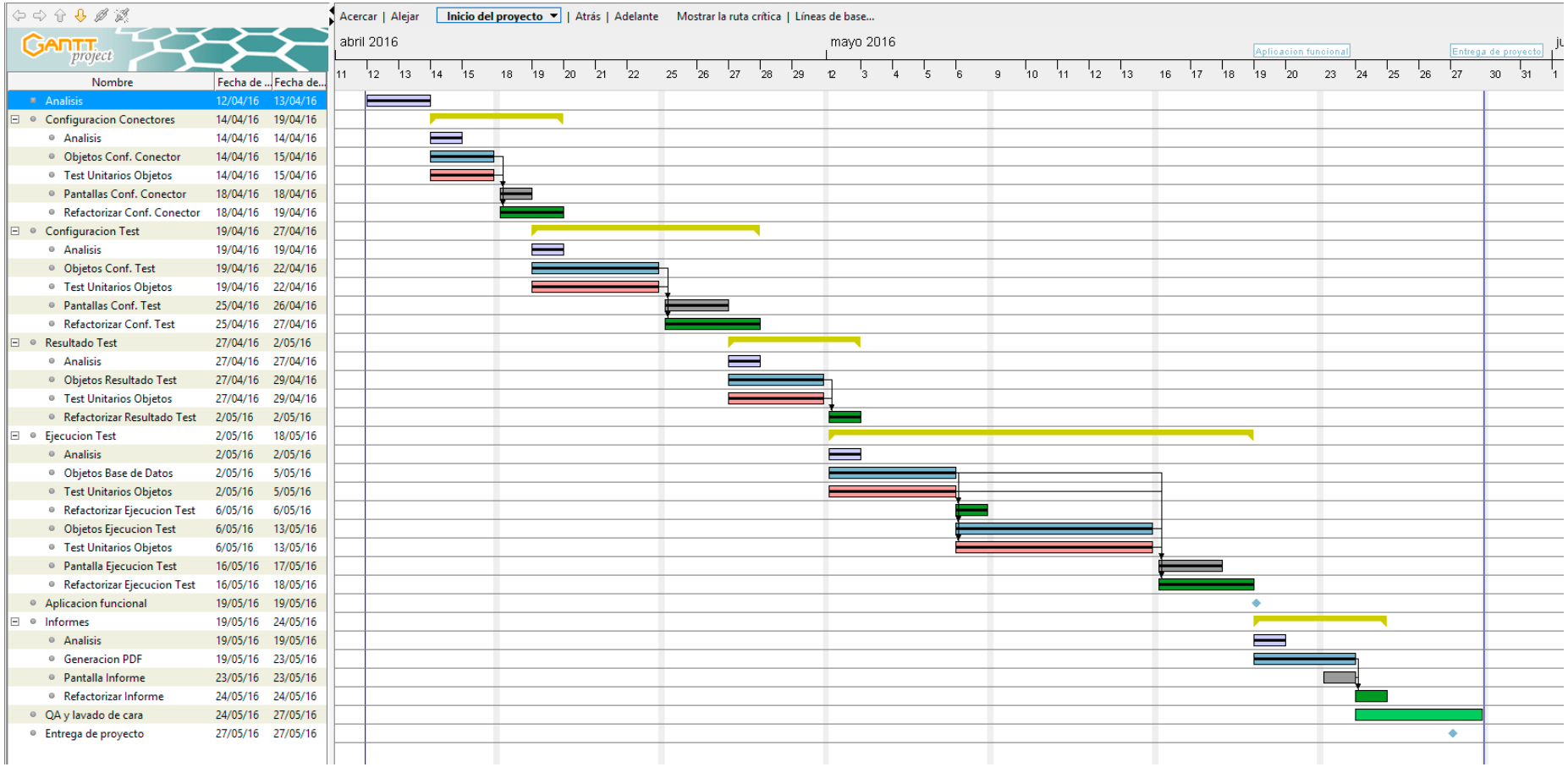
Específicamente la aplicación deberá tener las siguientes secciones:

- Configurar los conectores a Base de Datos que podremos usar.
- Configurar los Test, con cada una de sus secciones.
- Permitir ejecutar un Test sobre una lista de conectores
- Guardar los resultados en XML
- Permitir abrir estos resultados y convertirlos en PDF con un formato más visual con gráficas.
- Permitir comparar varios resultados.

A nivel personal, con el proyecto se busca:

- Obtener el título de Técnico en Desarrollo de Aplicaciones Multiplataforma.
- Utilizar patrones de diseño y respetar los principios de Clean Code en la medida de lo posible, para mejorar mi uso de ellos.
- Aprender a usar los Test Unitarios dentro de un proyecto de .NET
- Obtener una aplicación que nos permita mayor flexibilidad a la hora de tomar decisiones sobre el resultado de determinadas consultas.

2.2. Planificación



4 Diagrama de Gantt

3. Diseño

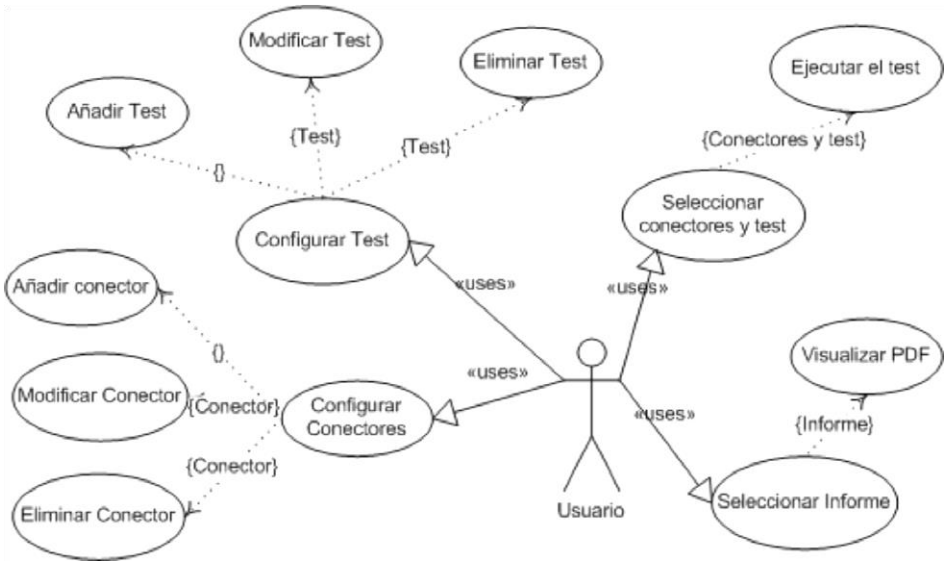
3.1. Requisitos mínimos

Para poder utilizar la aplicación tendremos unos requisitos mínimos, que serán los siguientes:

- Windows XP o superior, con .NET 4.0 instalado.
- Una base de datos instalada con un conector ODBC para Windows, o MySQL instalado con el conector ODBC o el Nativo de .NET

3.2. Diagrama de casos de uso

En la aplicación no se hace distinción de roles ni hay ninguna autentificación con lo cual solo tendremos un actor, que será el usuario en sí.



5 Diagrama de casos de uso

El actor, podrá realizar cuatro acciones principales:

Configurar conectores, a partir de esta acción podrá acceder a otras tres acciones secundarias:

- Añadir conector
- Modificar conector
- Eliminar conector

Configurar test, a partir de esta acción podrá acceder a otras tres acciones secundarias:

- Añadir test
- Modificar test
- Eliminar test

Ejecutar test, para ello deberá antes seleccionar una lista de conectores y un test.

Seleccionar informe, y visualizar el PDF de dicho informe.

3.3. Diagrama de clases

La aplicación se dividirá en una serie de partes funcionales que serán las encargadas de que el sistema funcione.

- Conectores
- Tests
- Ejecucion de Test
- Resultados Test

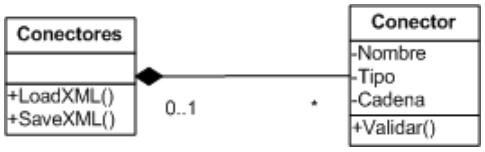
3.3.1. Diagrama de clases Conectores y Conector

La clase **Conector** tendrá una función:

- **Validar** que comprobara que los datos de sus atributos son correctos, el nombre no está vacío, el Tipo es ODBC o MySQL.

La clase **Conectores** tendrá dos funciones:

- **LoadXML**: Leerá el fichero XML donde están guardados los datos de los conectores y cargara con estos datos la lista de conectores.
- **SaveXML**: Se encargara de guardar en el fichero XML, el contenido de la lista de conectores.



6 Diagrama de clases - Conectores / Conector

3.3.2. Diagrama de clases Tests, Test, Sección, Bloque y Sentencia

Vamos a describirlos de dentro hacia afuera, empezando en la clase más simple, hasta llegar a la más compleja, justo en el orden inverso a como aparecen en el título de la sección y en la ilustración inferior.

La clase **Sentencia,** es una clase que lo único que hace es guardar la cadena de texto de la sentencia SQL.

La clase **Bloque,** tiene una colección de sentencias, y dos funciones:

- **SentenciasToString**: Recorrerá la lista de sentencias y retornara una cadena de texto con todas ellas.
- **Validar:** Comprobara que los datos introducidos son correctos, y tiene un nombre asignado.

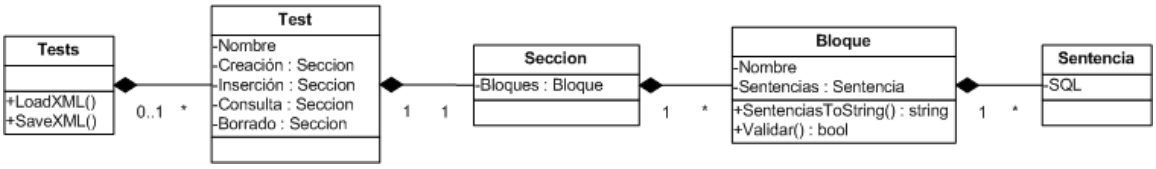
La clase **Sección**, será una colección de bloques.

La clase **Test,** tendrá asignado un **nombre** descriptivo, y cuatro secciones llamadas Creación, Inserción, Consulta y Borrado, que serán de tipo **Sección.**

Solo habrá una de cada tipo, aunque estas podrán contener varios bloques.

La clase **Tests** tendrá dos funciones:

- **LoadXML**: Leerá el fichero XML donde están guardados los datos de los test y cargara con estos datos la lista de tests
- **SaveXML**: Se encargara de guardar en el fichero XML, el contenido de la lista de tests.



7 Diagrama de clases - Tests / Test / Sección / Bloque / Sentencia

3.3.3. Diagrama de clases ResultadosTest, ResultadoSeccion y ResultadoBloque

Vamos a describirlos de dentro hacia afuera, empezando en la clase más simple, hasta llegar a la más compleja, justo en el orden inverso a como aparecen en el título de la sección y en la ilustración inferior.

La clase **ResultadoBloque,** es la clase que guardara el resultado de la ejecución de un bloque, tendrá el nombre del bloque en cuestión, el número de sentencias que se ejecutaron y el tiempo empleado en ello, en milisegundos.

La clase **ResultadoSeccion**, será una colección de ResultadosBloque.

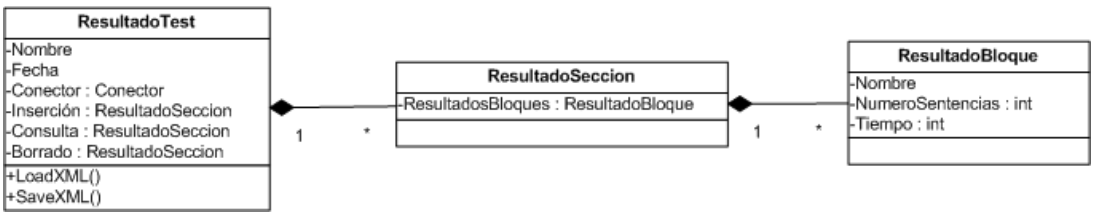
La clase **ResultadoTest,** tendrá el **nombre** descriptivo del test, la **fecha** de ejecución del test y tres secciones, **Inserción, Consulta** y **Borrado**.

Solo habrá una de cada tipo, aunque estas podrán contener varios bloques.

En esta clase, no aparece la sección **Creación**, ya que dicha sección del test es utilizada para preparación del esquema de la base de datos a la hora de ejecutar el resto de secciones, y no nos interesa a efectos de rendimiento.

También tendrá dos funciones:

- **LoadXML**: Leerá el fichero XML donde están guardados los resultados del test y cargara con estos datos el resto de clases.
- **SaveXML**: Se encargara de guardar en el fichero XML, el contenido del resultado del test.



8 Diagrama de clases - ResultadoTest / ResultadoSeccion / ResultadoBloque

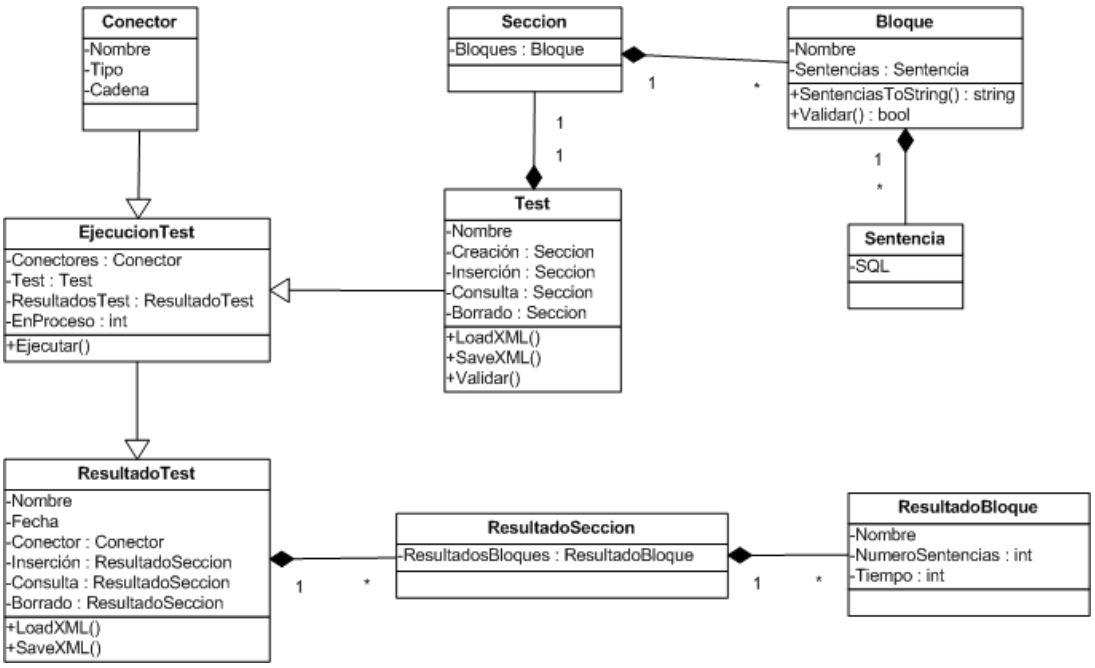
3.3.4. Diagrama de clases EjecuciónTest

Esta clase es la más importante, será la que realmente realizara el proceso principal del proyecto, la ejecución del test y recogida de los datos.

La clase **EjecuciónTest,** tendrá una colección de **Conectores**, ya que podremos lanzar el mismo test sobre varios conectores configurados en la aplicación, un **Test**, que será el que se va a ejecutar.

Al mismo tiempo tendrá una colección de **ResultadosTest,** que será el método **Ejecutar,** el encargado de ir rellenando esta colección de resultados e irlos guardando en disco llamando al método **SaveXML**, cada vez que finalice la ejecución de un test sobre un conector.

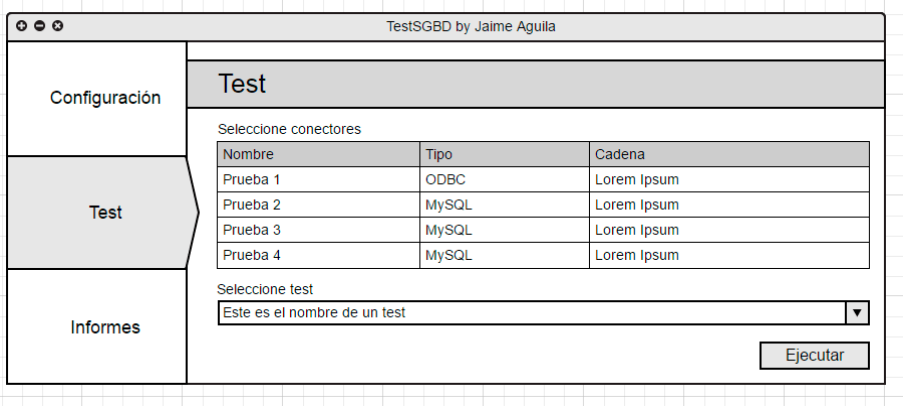
El método **Ejecutar,** será el encargado de realizar todo el proceso, encargándose de preparar el proceso, instanciar varios hilos de ejecución, cargar las sentencias que se deben ejecutar en cada hilo, preparar un contador de tiempo, poner en marcha cada uno de los hilos y esperar hasta que todos los hilos finalicen su ejecución, para obtener el tiempo que ha tardado en ejecutarse, y con ello rellenar la colección de ResultadosTest.



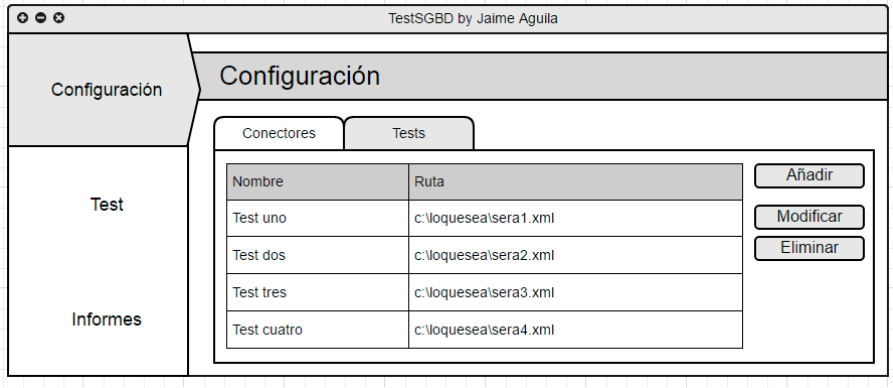
9 Diagrama de clases - EjecucionTest

3.4. Mockups

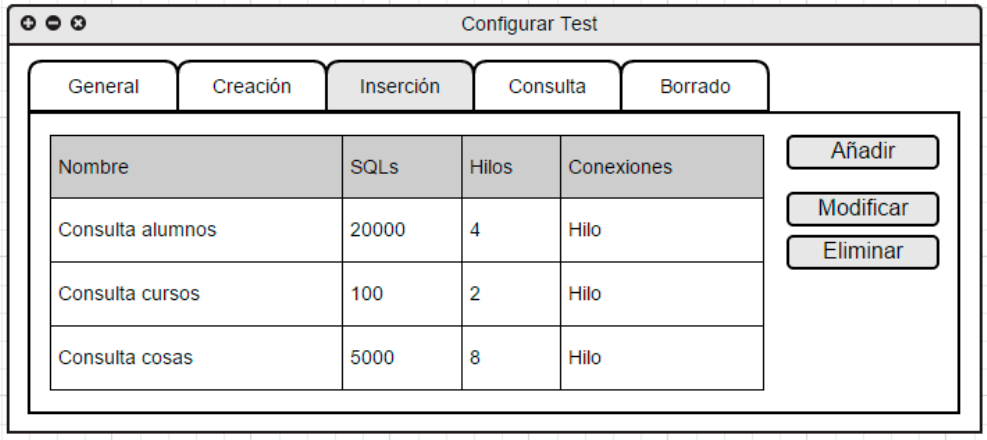
A continuación se incluyen imágenes de los diseños previos de algunas pantallas de la aplicación.



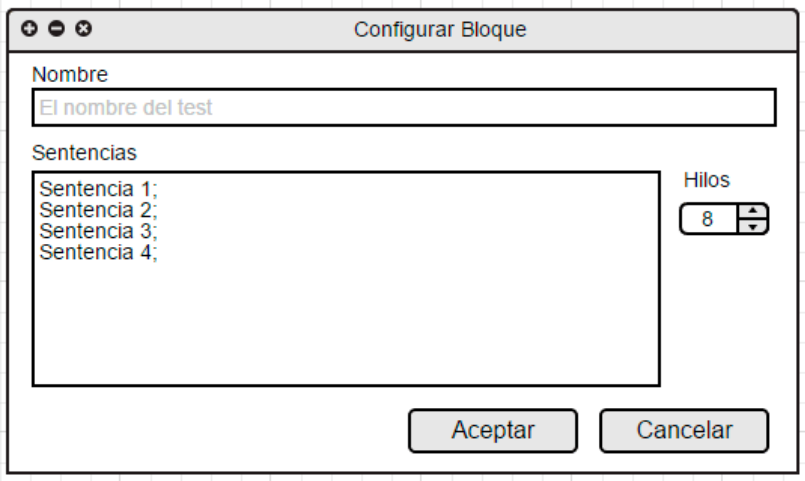
10 Diseño previo pantalla principal



11 Diseño previo pantalla configuración



13 Diseño previo pantalla configurar test

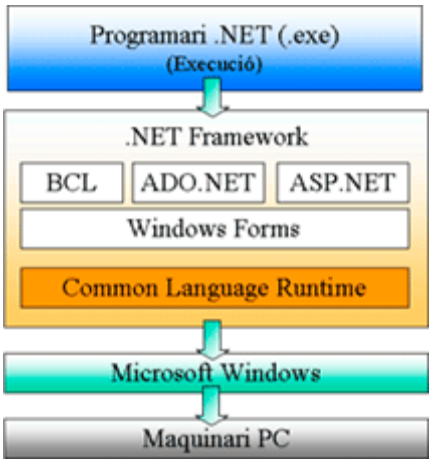


12 Diseño previo pantalla configurar bloque

4. Tecnologías y Herramientas

4.1. .Net Framework

.Net Framework es un entorno de desarrollo y ejecución de aplicaciones de Microsoft, que aglutina una serie de tecnologías (ASP.NET, Web Services, WinForms, ADO.NET, Biblioteca de clases .NET, WCF, WPF) específicamente diseñadas para poder crear todo tipo de aplicaciones.



14 Esquema .Net Framework

El corazón de .Net Framework es el **CLR (Common Language Runtime)**, que es el encargado de gestionar la ejecución de código compilado para la plataforma .NET. Puede asimilarse a la máquina virtual de Java.

Las aplicaciones son compiladas y ejecutadas dentro del CLR, de forma similar a las aplicaciones Java que son ejecutadas dentro de su máquina virtual.

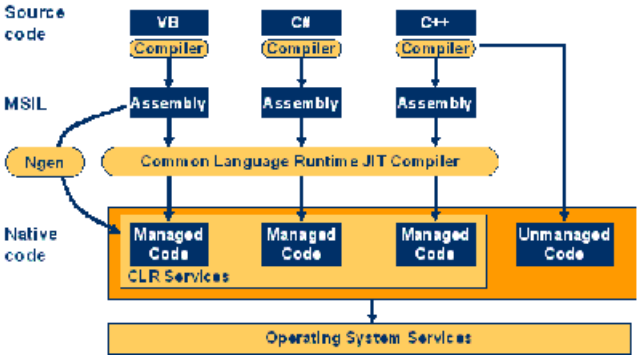
A estas aplicaciones les ofrece numerosos servicios para facilitar su desarrollo y mantenimiento y favorecen su fiabilidad y seguridad.

Las dos principales características del CLR son:

- **Ejecución multiplataforma**
- **Integración de lenguajes**

Para que un lenguaje de programación sea soportado por la plataforma .NET, ha de existir un compilador que traduzca de este lenguaje a MSIL ("managed code"). A la hora de ejecutar el código intermedio, éste es siempre compilado a código nativo.

Como se puede deducir de las características comentadas, el CLR lo que hace es gestionar la ejecución de las aplicaciones diseñadas para la plataforma .NET. Por esta razón, al código de estas aplicaciones se le suele llamar código gestionado, y al código no escrito para ser ejecutado directamente en la plataforma .NET se le suele llamar código no gestionado.



15 Managed code

4.2. C#

C#[1] (pronunciado si sharp en inglés) es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270).

C# combina los mejores elementos de múltiples lenguajes de amplia difusión como C++, Java, Visual Basic o Delphi. La idea principal detrás del lenguaje es combinar la potencia de lenguajes como C++ con la sencillez de lenguajes como Visual Basic, y que además la migración a este lenguaje por los programadores de C/C++/Java sea lo más inmediata posible.

El estándar **ECMA-334** lista las siguientes metas en el diseño para C#:

- Lenguaje de programación orientado a objetos.
- Inclusión de principios de ingeniería de software, como revisión estricta de tipos de datos, detección de variables no inicializadas, y recolección de basura automática.
- Capacidad para desarrollar componentes que se usen en ambientes distribuidos.
- Portabilidad del código fuente.
- Fácil migración del programador al nuevo lenguaje, especialmente para programadores familiarizados con C, C++ y Java.
- Soporte para internacionalización.
- Aplicaciones económicas en cuanto a memoria y procesado.

4.3. Winforms

Windows Forms (o **formularios Windows**) es el nombre dado a la interfaz de programación de aplicación gráfica (**GUI**[2]) que se incluye como parte de Microsoft .NET Framework, que proporciona acceso a los elementos de la interfaz de Microsoft Windows nativas envolviendo la API de Windows existente en código administrado.

Una aplicación de Windows Forms es una aplicación **orientada a eventos** con el apoyo de Microsoft .NET Framework. A diferencia de un programa por lotes, que pasa la mayor parte de su tiempo simplemente esperando a que el usuario haga algo, como relleno en un cuadro de texto o haga clic en un botón.

[1] En ocasiones representado como 'C♯', donde el segundo carácter es un *sostenido* (*sharp* en inglés)
[2] La interfaz gráfica de usuario, conocida también como GUI (del inglés graphical user interface)

4.4. Visual Studio 2010 - IDE

Microsoft Visual Studio es un entorno de desarrollo integrado (**IDE**[3]) de Microsoft.

Visual Studio incluye un editor de código de soporte IntelliSense (la finalización de código de componentes), así como la refactorización de código. El depurador integrado funciona tanto como un depurador a nivel de fuente como un depurador a nivel de máquina. Otras herramientas integradas incluyen un diseñador de formularios para la construcción de GUI aplicaciones, diseñador de páginas web, la clase de diseño y esquema de base de diseño.

Visual Studio es compatible con diferentes lenguajes de programación y permite la edición de código y el soporte de debug (en diversos grados) en casi cualquier lenguaje de programación, siempre que exista un servicio específico del lenguaje. Los lenguaje incorporados incluyen C, C++ y C++/CLI (a través de Visual C++), VB.NET (a través de Visual Basic.NET), C# (a través de Visual C#), y F# (a partir de Visual Studio 2010). El soporte para otros lenguajes como Python, Rubí, Node.js, y M entre otros está disponible a través de los servicios de lenguaje instalados por separado. También es compatible con XML/XSLT, HTML/XHTML, Javascript y CSS. Java (y J#) se apoyaron en el pasado.

4.5. Control de versiones

Comenzaremos explicando que es un sistema de control de versiones y porque es necesario.

Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Una versión, revisión o edición de un producto, es el estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación.

Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión dando lugar a los llamados sistemas de control de versiones. Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas (por ejemplo, para algún cliente específico). Ejemplos de este tipo de herramientas son entre otros: **CVS**, **Subversion**, **SourceSafe**, **Git**, **Mercurial**, o **Team Foundation Server.**

El control de versiones se realiza principalmente en la industria informática para controlar las distintas versiones del código fuente dando lugar a los sistemas de control de código fuente o SCM. Sin embargo, los mismos conceptos son aplicables a otros ámbitos como documentos, imágenes, sitios web, etc.

Un sistema de control de versiones debe proporcionar:

- Mecanismo de almacenamiento de los elementos que deba gestionar.
- Posibilidad de realizar cambios sobre los elementos almacenados.
- Registro histórico de las acciones realizadas con cada elemento o conjunto de elementos (normalmente pudiendo volver o extraer un estado anterior del producto).

[3] Un IDE es una herramienta que nos ayuda a desarrollar de una manera amigable nuestras aplicaciones, brindándonos ayudas visuales en la sintaxis, plantillas, asistentes, plugins y sencillas opciones para probar y hacer debug.

Aunque no es estrictamente necesario, suele ser muy útil la generación de informes con los cambios introducidos entre dos versiones, informes de estado, marcado con nombre identificativo de la versión de un conjunto de ficheros, etc.

4.5.1. Git

Git es un sistema de control de versiones distribuido lo que le aporta:

- Velocidad
- Diseño sencillo
- Fuerte apoyo al desarrollo no lineal (miles de ramas paralelas)
- Completamente distribuido
- Capaz de manejar grandes proyectos (como el núcleo de Linux) de manera eficiente (velocidad y tamaño de los datos)

Desde su nacimiento en 2005, Git ha evolucionado y madurado para ser fácil de usar y aún conservar estas cualidades iniciales. Es tremendamente rápido, muy eficiente con grandes proyectos, y tiene un increíble sistema de ramificación (branching) para desarrollo no lineal, además en la actualidad se ha convertido en un estándar de facto, sobre todo en el desarrollo de software libre.

4.5.2. GitHub

GitHub es un alojamiento web para alojar proyectos utilizando el sistema de control de versiones **Git**. El código se almacena en repositorios de forma pública, aunque también se puede hacer de forma privada, creando una cuenta de pago.

4.6. XML

XML, siglas en inglés de **eXtensible Markup Language** ("Lenguaje de Marcas Extensible"), es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible. Proviene del lenguaje SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML) para estructurar documentos grandes.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

Usaremos XML para el almacenamiento en disco de los ficheros, tanto de configuración, como los ficheros de resultados, con los que generaremos los informes.

5. Implementación

5.1. Metodología

Para el desarrollo del proyecto, se ha optado por usar metodologías agiles, dado que las metodologías tradicionales (Métrica 3, Cascada, en V, ITIL) son muy rígidas frente a los cambios, se deben seguir unas políticas y normas muy estrictas, y obligan a ceñirse al análisis inicial del desarrollo.

Por el contrario las metodologías agiles (XP[4], SCRUM, Kanban) tienen una mayor flexibilidad ante los cambios, los procesos están menos ceñidos a normas y permiten una mayor respuesta al cambio, y dado que el proyecto queremos que nos sea lo más útil posible, necesitábamos tener una mayor libertad para ir adaptándolo y modificándolo conforme fuéramos avanzando en él.

Los defensores de la XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

Se ha partido de un análisis genérico, para a continuación ir realizando cada sección en ciclos más cortos en los cuales se parte de un análisis, se buscan cada una se las tareas a realizar, se ordenan y se van implementando, en caso de encontrarnos con vacíos o mejoras a implementar, se añaden a la lista de tareas, para realizarlas en el siguiente ciclo.

5.1.1. Metodología mixta

Dado que prácticamente todos los roles definidos tanto en Scrum o XP, los hemos acaparado, no tenía mucho sentido seguir una metodología ágil al pie de la letra, ya que las reuniones de equipo (clientes, programadores, analistas, testers) no tenían sentido, así que hemos acabado realizando una mezcla de desarrollo Agile.

En vez de desarrollar todo el proyecto en bloque y no tener nada funcional hasta los últimos pasos del desarrollo, hemos dividido el proyecto en pequeñas partes totalmente funcionales y estas a su vez la hemos dividido en tareas mucho más sencillas, realizando pequeñas iteraciones que incluían planificación, análisis de requisitos, diseño, codificación, pruebas y documentación.

Teniendo gran importancia el concepto de "Finalizado" (Done), ya que el objetivo de cada iteración no es agregar toda la funcionalidad para justificar la finalización del producto, sino incrementar el valor por medio de "software que funciona" (sin errores).

Para ellos nos hemos apoyado totalmente en Github, su apartado de Issues, y en el uso de **PDA**s (Papel De Apuntar), cada tarea, mejora, error, adaptación, acababa apuntado en un papel, y en cada ciclo se agrupaban una serie de pequeñas tareas que tuvieran relación y se incluían en el proceso, una vez acabadas, se probaba su funcionamiento y se refactorizaba

[4] XP - eXtreme Programming o programación extrema, formulada por Kent Beck

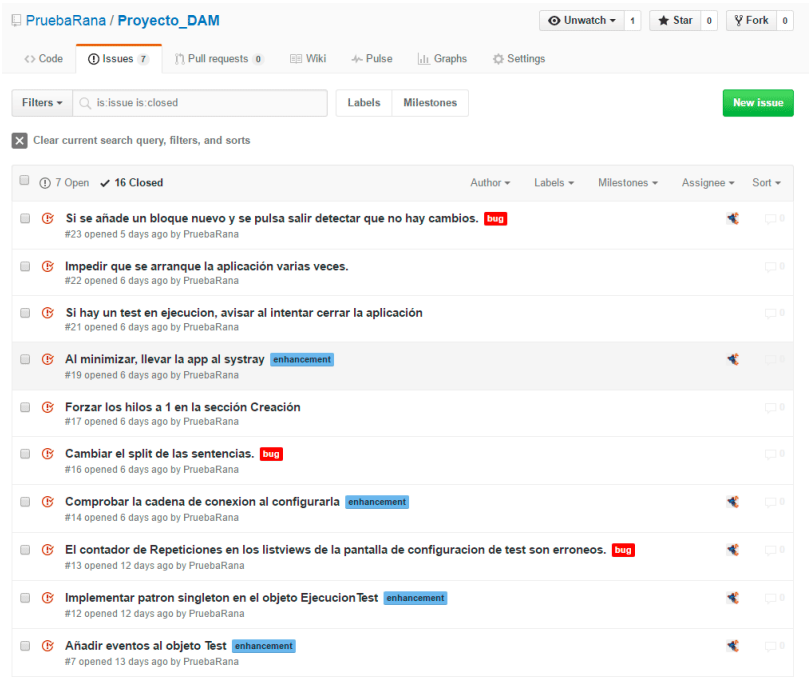
el código para darle un mínimo de calidad, si en algún momento se encontraban errores en otros apartados, o aparecía una deficiencia en el diseño que necesitara cambiar alguna parte ya realizada, acababan apuntados en papel, para que en el siguiente ciclo se tuvieran en cuenta.

Un ejemplo de esta plasticidad ante el cambio, nos ha ocurrido con el proceso de los test, en un principio pensamos que los bloques de una sección se ejecutarían en un número determinado de hilos, como se puede apreciar en los mockups del apartado **3.4 Mockups**, una vez que ya teníamos todos apartados de configuración y sus pantallas desarrollados, al llegar al apartado de ejecución del test, aparecieron un par de modificaciones que serían indispensables:

- Permitir configurar el bloque como un bucle for, en el cual indicar con cuantos hilos queremos repetir el bloque.
- Poder especificar como nos tenemos que comportar ante la apertura y cierre de la conexión a la base de datos, en un principio no se tenía en cuenta, y ahora necesitábamos poder decidir cómo se debería de ejecutar el bloque en cuestión, manteniendo la conexión abierta durante todo el proceso **BLOQUE,** que cada hilo, abriera su propia conexión **HILO,** que fuera cada sentencia la que se encargara de abrir su conexión **SENTENCIA.** Y no solo de una forma, sino que nos permitiera especificar una combinación de estas.

Estas dos modificaciones, no solo cambiaban aspectos de la configuración y las pantallas, implicaban un cambio mucho más profundo en la ejecución en sí de los test, y en los objetos de resultados, que pudimos realizar gracias a la metodología empleada.

También hemos usado GitHub, en su apartado de Issues, para identificar estos ciclos, y tener una visión más global de la situación, ya que nos permite clasificar cada Issue por colores para identificar los que son mejoras, de lo que son corrección de errores, y ver rápidamente cuantos tenemos ya cerrados, y cuantos hay abiertos, como se puede ver en la siguiente imagen.



16 Apartado de Issues de Github

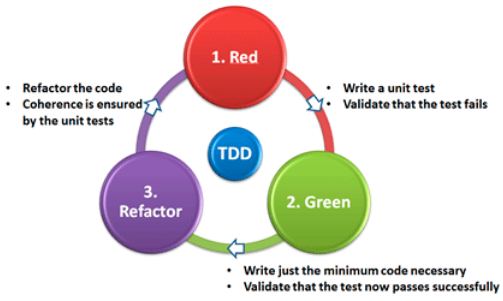
5.1.2. TDD

Aunque no hemos seguido una metodología totalmente TDD (Test-Driven Development) desarrollo guiado por pruebas, sí que hemos querido apoyarnos en los test unitarios, para poder realizar los procesos de refactorización teniendo mayor seguridad.

En TDD en primer lugar se debe definir una lista de requisitos y después se ejecuta el siguiente ciclo:

- **Elegir un requisito**: Se elige de una lista el requisito que se cree que nos dará mayor conocimiento del problema y que a la vez sea fácilmente implementable.
- **Escribir una prueba**: Se comienza escribiendo una prueba para el requisito. Para ello el programador debe entender claramente las especificaciones y los requisitos de la funcionalidad que está por implementar. Este paso fuerza al programador a tomar la perspectiva de un cliente considerando el código a través de sus interfaces.
- **Verificar que la prueba falla**: Si la prueba no falla es porque el requisito ya estaba implementado o porque la prueba es errónea.
- **Escribir la implementación:** Escribir el código más sencillo que haga que la prueba funcione. Se usa la expresión "Déjelo simple" ("Keep It small and simple"), conocida como principio **KISS**.
- **Ejecutar las pruebas automatizadas**: Verificar si todo el conjunto de pruebas funciona correctamente.
- **Eliminación de duplicación**: El paso final es la refactorización, que se utilizará principalmente para eliminar código duplicado. Se hace un pequeño cambio cada vez y luego se corren las pruebas hasta que funcionen.
- **Actualización de la lista de requisitos:** Se actualiza la lista de requisitos tachando el requisito implementado. Asimismo se agregan requisitos que se hayan visto como necesarios durante este ciclo y se agregan requisitos de diseño (P. ej que una funcionalidad esté desacoplada de otra).

Resumiendo en el TDD primero se escribe el Test, de forma que este al ejecutarse falla, después se escribe el código mínimo del programa para que el test no falle, después este código se refactoriza y se vuelve al principio para escribir nuevos test.



17 Ciclo TDD Rojo-Verde-Refactorizar

Es un enfoque que puede parecer contraproducente y muy lento al principio, pero que obliga a ir conociendo la lógica del negocio de forma más profunda para ir implementando tests más específicos que acaben dando como resultado el código mínimo que implementa toda la funcionalidad básica de la aplicación a la par que la cobertura del código mediante test es

del 100%, dando lugar a un código robusto y que está completamente comprobado por test, de forma que el mantenimiento y la ampliación sea más sencilla y segura.

En nuestro caso, no hemos desarrollado la aplicación siguiendo TDD, sí que empleábamos el mismo concepto de ciclos, lo único que en cada ciclo no comenzábamos directamente escribiendo las pruebas, sino que iban a la par, mientras implementábamos el código íbamos escribiendo pruebas y estas a su vez nos hacían reevaluarnos la implementación, una vez finalizado el ciclo, refactorizabamos el código, y los test nos facilitaban mucho la tarea, ya que sin tener pruebas unitarias, la refactorización puede ser una lotería.

Adjunto un par de ejemplos, y los explico a continuación:

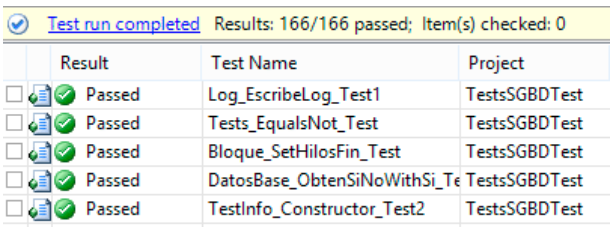
```
[TestMethod()]
public void Seccion_Constructor_Test2()
{
    // Arrange
    Seccion target1 = this._Item;
    Seccion target = new Seccion();
    // Act
    target.Bloque = target1.Bloque;
    // Assert
    Assert.AreEqual(3, target.Bloque.Count);
    Assert.AreEqual("Prueba", target.Bloque[0].Nombre);
    Assert.AreEqual("Prueba2", target.Bloque[1].Nombre);
    Assert.AreEqual("Prueba3", target.Bloque[2].Nombre);
}
[TestMethod()]
public void Seccion_Clone_Test()
{
    // Arrange
    Seccion target1 = this._Item;
    // Act
    Seccion target2 = target1.Clone();
    // Assert
    Assert.AreEqual(target1, target2);
}
```

18 Ejemplos de Test Unitario

Los test, se basan en la metodología AAA (**Arrange-Act-Assert);**

- **Arrange**: es la sección de preparación y asignación de valores
- **Act**: es actuar, ejecutar lo que se quiere probar
- **Assert**: es asegurarse de que se ha hecho lo que se tiene que hacer

Cuando se ejecuta el test, aparece marcado en verde y si falla en rojo.

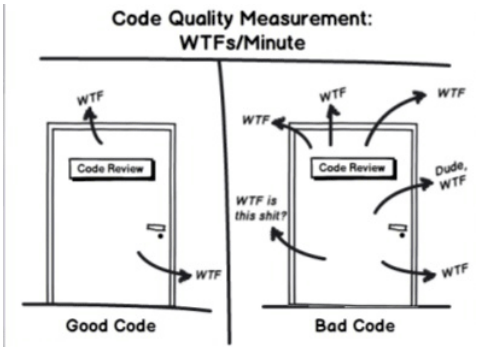


19 Ejecución de test unitarios en VS

5.1.3. Refactorización

Refactorizar es realizar una transformación al código **preservando su comportamiento**, modificando sólo su estructura interna para **mejorarlo**.

Es muy importante tener muy claro este concepto, ya que la refactorización no debe de modificar el comportamiento del código, esto es **fundamental.** Refactorizar no es corregir errores, ni añadir funcionalidades, ni hacer optimizaciones de código para que sea más rápido, es mejorar el código para que sea más robusto y fácil de mantener, sin que su comportamiento cambie, lo que se conoce informalmente por limpiar el código.



20 Identificar calidad del código

El objetivo es mejorar la facilidad de comprensión del código o cambiar su estructura y diseño y eliminar código muerto, para facilitar el mantenimiento en el futuro.

A finales de 1990, Kent Beck acuño el término **"Code Smell"** conocido por **código que huele o apesta.** Una forma de saber si nuestro código necesita ser refactorizado es intentar detectar estos "olores", conocidos como Bad Smell, Un Bad Smell es un indicio de que existe código de poca calidad, hay muchas formas distintas de que el código apeste y existen refactorizaciones específicas para cada uno de estos olores.

Aprender a identificar estos "malos olores", y conocer cómo se suelen Refactorizar para evitar que apesten, nos puede ser muy útil en el desarrollo de software, y sobre todo a la hora de ampliar funcionalidades o mantener el código.

No vamos a detallar cada uno de estos olores, y cada una de las formas o métodos de Refactorizar, vamos a poner unos ejemplos de cómo lo hemos aplicado.

5.1.3.1. Constructor EjecucionTest

En el constructor de EjecuciónTest, asignamos las propiedades principales y calculamos la cantidad de pasos totales que tendrá el test en sí, un test tiene tres secciones **(Inserción, Consulta y Borrado)** y estas secciones tienen una colección de Bloques, cada bloque tiene una lista de sentencias, una configuración de repeticiones de hilos, y una configuración de cómo manejar la conexión (**BLOQUE, HILO, SENTENCIA**).

Para saber la cantidad de pasos totales, tenemos que calcular las repeticiones del FOR de hilos, multiplicarlas por la cantidad de conexiones, así sabremos cuantas veces se va a repetir ese bloque, sumar las repeticiones de cada bloque en sí, y repetir el proceso para cada una de las secciones, para acabar multiplicar este número, por la cantidad de conectores sobre los que se va a repetir el test.

El código en un principio era este:

```
private EjecucionTest(List<Conector> aConectores, Test aTest)
{
    this.C = aConectores;
    this.Test = aTest;
    this.Res = new List<ResultadoTest>();
    this.PA = 0;
    // Calcular la cantidad de pasos del test.
    int i = 0;
    // Insercion
    foreach (Bloque lItem in this._Test.Insercion.Bloque)
    {
        int x = 0;
        // Si es bloque, sumar 1.
        if (lItem.Conexion.HasFlag(Bloque.TipoConexion.BLOQUE)){ x++; }
        // Si es hilo, sumar 1.
        if (lItem.Conexion.HasFlag(Bloque.TipoConexion.HILO)){ x++; }
        // Si es sentencia, sumar 1.
        if (lItem.Conexion.HasFlag(Bloque.TipoConexion.SENTENCIA)){ x++; }
        // Calculamos las repeticiones, segun el for de hilos
        int liRepeticiones = ((lItem.Hilos_Fin-lItem.Hilos_Inicio)/lItem.Hilos_Step)+1;
        // multiplicamos las repeticiones, por la cantidad de bloques y los sumamos total
        i += liRepeticiones * x;
    }
    // Consulta
    foreach (Bloque lItem in this._Test.Consulta.Bloque)
    {
        int x = 0;
        // Si es bloque, sumar 1.
        if (lItem.Conexion.HasFlag(Bloque.TipoConexion.BLOQUE)){ x++; }
        // Si es hilo, sumar 1.
        if (lItem.Conexion.HasFlag(Bloque.TipoConexion.HILO)){ x++; }
        // Si es sentencia, sumar 1.
        if (lItem.Conexion.HasFlag(Bloque.TipoConexion.SENTENCIA)){ x++; }
        // Calculamos las repeticiones, segun el for de hilos
        int liRepeticiones = ((lItem.Hilos_Fin-lItem.Hilos_Inicio)/lItem.Hilos_Step)+1;
        // multiplicamos las repeticiones, por la cantidad de bloques y los sumamos total
        i += liRepeticiones * x;
    }
    // Borrado
    foreach (Bloque lItem in this._Test.Borrado.Bloque)
    {
        int x = 0;
        // Si es bloque, sumar 1.
        if (lItem.Conexion.HasFlag(Bloque.TipoConexion.BLOQUE)){ x++; }
        // Si es hilo, sumar 1.
        if (lItem.Conexion.HasFlag(Bloque.TipoConexion.HILO)){ x++; }
        // Si es sentencia, sumar 1.
        if (lItem.Conexion.HasFlag(Bloque.TipoConexion.SENTENCIA)){ x++; }
        // Calculamos las repeticiones, segun el for de hilos
        int liRepeticiones = ((lItem.Hilos_Fin-lItem.Hilos_Inicio)/lItem.Hilos_Step)+1;
        // multiplicamos las repeticiones, por la cantidad de bloques y los sumamos total
        i += liRepeticiones * x;
    }
    this.PT = i * this.C.Count;
}
```

21 Constructor EjecucionTest sin refactorizar

El código tiene incluso comentarios, y funciona correctamente, pero podemos detectar algunos malos olores, como **variables con nombres poco descriptivos**, **código duplicado**, y un **método demasiado grande**, para simplemente asignar cinco valores.

Así que para refactorizarlo, hemos asignado a todas las variables nombres más descriptivos, el cálculo de los pasos totales, lo hemos sacado a otra función, y además, este código repetido lo hemos simplificando mediante métodos que reciben objetos específicos para hacer el cálculo, quedando el siguiente código

```
private EjecucionTest(List<Conector> aConectores, Test aTest)
{
    this._Conectores = aConectores;
    this._Test = aTest;
    this._Resultados = new List<ResultadoTest>();

    this._PasosActual = 0;
    this._PasosTotales = this.ContarRepeticiones() * this._Conectores.Count;
}

private int ContarRepeticiones()
{
    int liCantidad = 0;
    liCantidad += ContarRepeticionesListaBloques(this._Test.Creacion);
    liCantidad += ContarRepeticionesListaBloques(this._Test.Insercion);
    liCantidad += ContarRepeticionesListaBloques(this._Test.Consulta);
    liCantidad += ContarRepeticionesListaBloques(this._Test.Borrado);
    return liCantidad;
}
private int ContarRepeticionesListaBloques(Seccion aLista)
{
    int liCantidad = 0;
    foreach (Bloque lItem in aLista.Bloque)
    {
        liCantidad += ContarRepeticionesBloque(lItem);
    }
    return liCantidad;
}
private int ContarRepeticionesBloque(Bloque aBloque)
{
     int liCantidad = (aBloque.Conexion.HasFlag(Bloque.TipoConexion.BLOQUE) ? 1 : 0) +
           (aBloque.Conexion.HasFlag(Bloque.TipoConexion.HILO) ? 1 : 0) +
           (aBloque.Conexion.HasFlag(Bloque.TipoConexion.SENTENCIA) ? 1 : 0);
     int liRepeticones = ( ( (aBloque.Hilos_Fin - aBloque.Hilos_Inicio) /
           aBloque.Hilos_Step) + 1 ) * liCantidad;
    return liRepeticones;
}
```

22 Constructor EjecucionTest refactorizado

Ahora cuando vemos el constructor, que ocupa solo 5 líneas, queda claro que es lo que hace, esta asignando las propiedades del objeto, y (**this._PasosTotales= this.ContarRepeticiones() * this._Conectores.Count;**) nos deja claro que multiplica las repeticiones del test, por el número de conectores sobre los que realizaremos el test, no necesitamos más para entenderlo.

5.1.3.2. frmPrincipal botones laterales

Otro ejemplo lo podemos encontrar en el formulario principal, los botones laterales (**Configuración, Test e Informes**) tenían asignados tres eventos, Click, MouseLeave y MouseEnter, simplemente para asignarles los colores, en total 9 métodos, que hacían lo mismo, el código sin refactoriar está en el **Apendice E**, en total 66 líneas muy sencillas, si en un futuro necesitáramos cambiar los colores, tendríamos que ir buscando y cambiándolos, ya que están en nueve sitios distintos, y no queda claro cuál es cual o si quisiéramos añadir nuevos botones, deberíamos replicar los eventos, y editarlos para adaptarlos al nuevo botón.

El código una vez refactoriazado está en el **Apendice F,** queda reducido a 35 líneas, un método **Click**, otro **MouseLeave** y otro **MouseEnter**, que lo comparten los tres botones, y tres métodos mas, para establecer los colores **ColorBotonDesactivado**, **ColorBotonActivado**, **ColorBotonResaltado**, de forma que cambiar los colores o incluso añadir nuevos botones sería una labor muy sencilla.

5.2. Programación orientada a objetos

Dada la complejidad del proyecto, se ha usado el paradigma de programación orientada a objetos, ya que de haber usado otros paradigmas, el proyecto se habría complicado demasiado y sería demasiado complejo de modificar y ampliar.

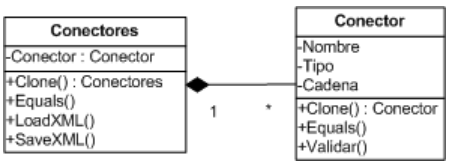
Aunque en muchas ocasiones no sea necesario, casi todos los objetos implementaran una serie de métodos por defecto, estos serán:

- **Dispose:** Los objetos que instancien componentes que deban ser liberados implementaran dicho método.
- **Clone:** Realizara una copia de sí mismo y retornara dicha copia, este método nos permitirá obtener una copia de un objeto de una forma rápida.
- **Equals**, **== y ¡=:** Estos métodos nos permitirán realizar comparaciones usando el método **.equals** o directamente los comparadores **==** o **¡=.** Permitiéndonos decidir en qué casos dos objetos serán idénticos. (determinadas propiedades deben de ser idénticas, pero por alguna razón en particular otras nos da igual que sean distintas) Así tenemos el control sobre las comparaciones, sin tener que realizar procesos externos de comprobación.

5.2.1. Objetos Conectores y Conector

El objeto Conectores, tendrá una colección de objetos Conector, y los métodos necesarios para clonar el objeto en sí, realizar comparaciones y los métodos encargados de persistir el objeto en disco en formato XML, en el **8.A** se encuentra un ejemplo de este XML

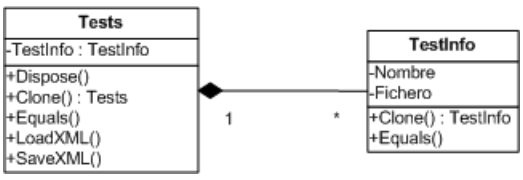
El objeto Conector, tendrá las propiedades **Nombre**, el nombre descriptivo del conector, **Tipo,** que nos indicara si es de tipo **ODBC** o **MySQL,** y **Cadena** que será la cadena de conexión para conectarse a una base de datos, también contara con los métodos **Clone, Equals y Validar.**



23 Objetos Conectores y Conector

5.2.2. Objetos Tests y TestInfo

El objeto Tests, tendrá una colección de objetos TestInfo, y los métodos necesarios para liberar recursos, clonar el objeto en sí, realizar comparaciones y los métodos encargados de persistir el objeto en disco en formato XML, en el **XML de** ejemplo del fichero Tests.xml se encuentra un ejemplo de este XML.



24 Objetos Tests y TestInfo

El objeto TestInfo, tendrá las propiedades **Nombre**, el nombre descriptivo del test y **Fichero**, que será la ruta al fichero XML donde estarán los datos del Test.

Se ha optado por no guardar en el objeto Tests una colección directa de objetos Test, ya que el objeto Test será el que contendrá todas las consultas que se van a lanzar, y el tamaño de este objeto puede ser demasiado grande, hemos preferido crear un objeto TestInfo, que contiene la ruta al XML del objeto Test en cuestión, de esta forma, el XML de Tests queda muy reducido y no nos penaliza a la hora de cargar las pantallas de la aplicación que dependen de este objeto.

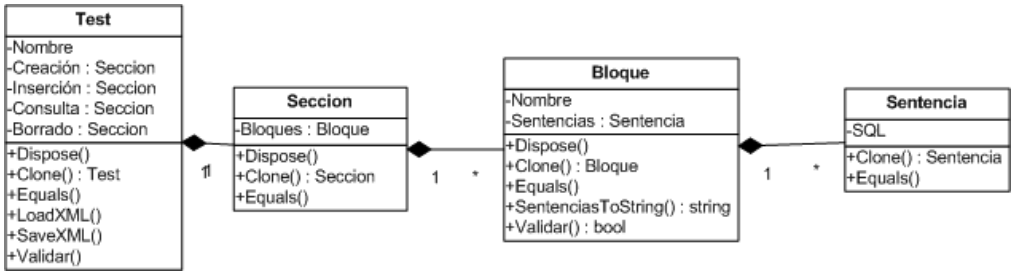
5.2.3. Objeto Test, Sección, Bloque y Sentencia

El objeto test, tendrá la propiedad de **Nombre**, el nombre descriptivo del test, **Creación**, **Inserción**, **Consulta** y **Borrado**, que serán objetos de tipo Sección, y los métodos necesarios para liberar recursos, clonar el objeto en sí, realizar comparaciones y los métodos encargados de persistir el objeto en disco en formato XML, en el **40** se encuentra un ejemplo de este XML.

El objeto Sección, tendrá una colección de objetos **Bloque**, y los métodos necesarios para liberar recursos, clonar el objeto en sí y realizar comparaciones.

El objeto Bloque, tendrá la propiedad de **Nombre**, el nombre descriptivo del bloque y una colección de objetos **Sentencia**, y los métodos necesarios para liberar recursos, clonar el objeto en sí, realizar comparaciones y además los métodos **Validar** y **SentenciasToString**

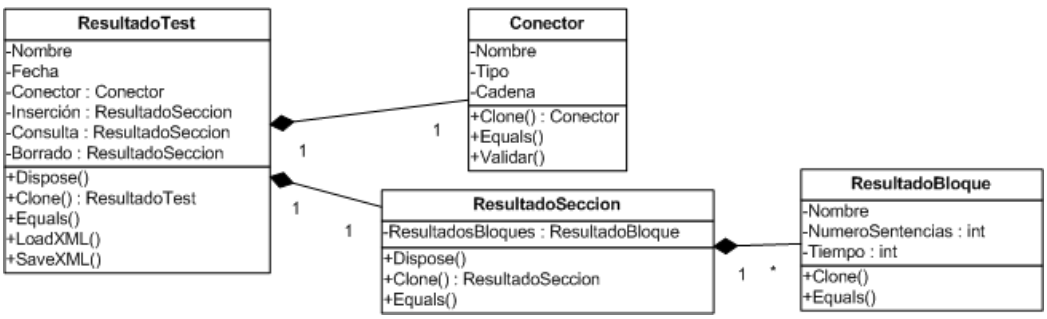
El objeto Sentencia, tendrá una propiedad **SQL**, que será la sentencia SQL en cuestión y los métodos necesarios para clonar el objeto en sí y realizar comparaciones.



25 Objetos Test, Sección, Bloque y Sentencia

5.2.4. Objeto ResultadoTest, ResultadoSeccion y ResultadoBloque

El objeto ResultadoTests, tendrá las propiedades **Nombre,** el nombre descriptivo del Test,



26 Objetos ResultadoTest, ResultadoSeccion y ResultadoBloque

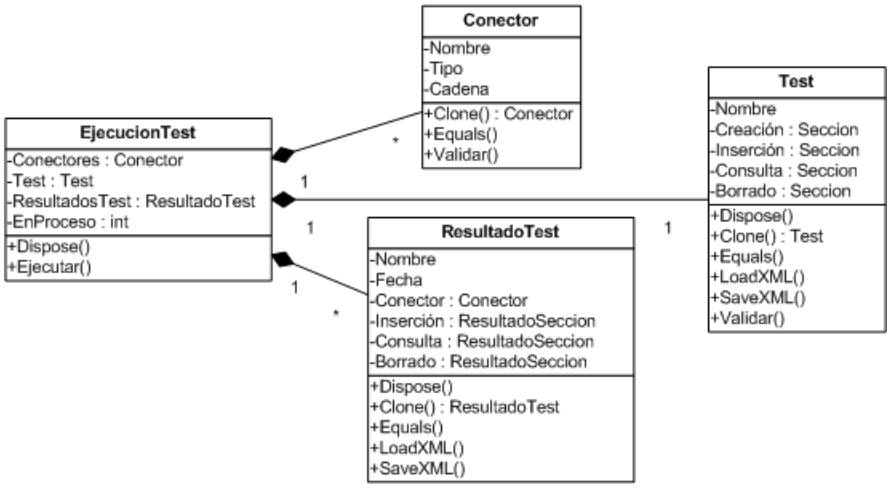
Fecha, la fecha en la que se realizó, **Conector,** que será un objeto Conector, **Insercion, Consulta y Borrado,** que serán objetos de tipo **ResultadoSeccion**, y los métodos necesarios para liberar recursos, clonar el objeto en sí, realizar comparaciones y los métodos encargados de persistir el objeto en disco en formato XML, en el **XML de** ejemplo del fichero Tests.xml se encuentra un ejemplo de este XML

5.2.5. Objeto EjecucionTest

Este objeto es el principal del proyecto, es donde recae todo el peso, recibe una colección de conectores sobre los cuales debe realizar las pruebas, y un test que es el que debe de ejecutar.

Tiene un método principal, **Ejecutar** que es el que se encargara de realizar el proceso, para ello mediante la biblioteca TPL (Task Parallel Library, biblioteca de procesamiento paralelo basado en tareas) usando Task, inicia el proceso, este proceso recorre los bloques del test y procesa cada uno de ellos, por cada bloque, comprueba la cantidad de hilos que tiene que lanzar y que tipo de conexión debe realizar (**BLOQUE/HILO/SENTENCIA**) entonces genera un array de Task por cada hilo, inicializa un cronometro **Stopwatch**, recorre las sentencias para ir rellenando los Task con dichas sentencias le pasa el objeto de conexión con la BD, objeto **DatosMySQL o DatosODBC** que son de tipo DatosBase, e inicializa el proceso en cada hilo, quedándose a la espera de que todos ellos finalicen.

Al finalizar todos, para el cronometro, y con los datos obtenidos, rellena el objeto **ResultadoTest,** al finalizar el test, llama al método SaveXML del objeto ResultadoTest que ha generado para que se guarde en disco. Y repite el proceso con el siguiente conector que tenga.



27 Objeto EjecucionTest

5.3. Patrones de diseño

En programación a veces nos encontramos con ciertos problemas que se repiten a lo largo de nuestra experiencia, como puede ser el caso típico, hacer una clase que pueda controlar múltiples tipos de conexiones, o una clase que una vez instanciada (creado un objeto) no se puede volver a crear una nueva instancia.

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su **efectividad** resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser **reutilizable**, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Fue a principios de la década de los 90, cuando los patrones de diseño tuvieron un gran éxito en el mundo de la informática a partir de la publicación del libro **Design Patterns** escrito por el grupo **Gang of Four (GoF)** compuesto por **Erich Gamma, Richard Helm, Ralph Johnson y John Vlisides**, en el que se recogían 23 patrones de diseño comunes.

Los patrones de diseño pretenden:

- Proporcionar catálogos de elementos reusables en el diseño de sistemas software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

Asimismo, no pretenden:

- Imponer ciertas alternativas de diseño frente a otras.
- Eliminar la creatividad inherente al proceso de diseño.

No es obligatorio utilizar los patrones, solo es aconsejable en el caso de tener el mismo problema o similar que soluciona el patrón, siempre teniendo en cuenta que en un caso particular puede no ser aplicable. "Abusar o forzar el uso de los patrones puede ser un error".

No vamos a detallar cada uno de estos patrones, simplemente vamos a comentar dos de ellos que hemos usado en el proyecto.

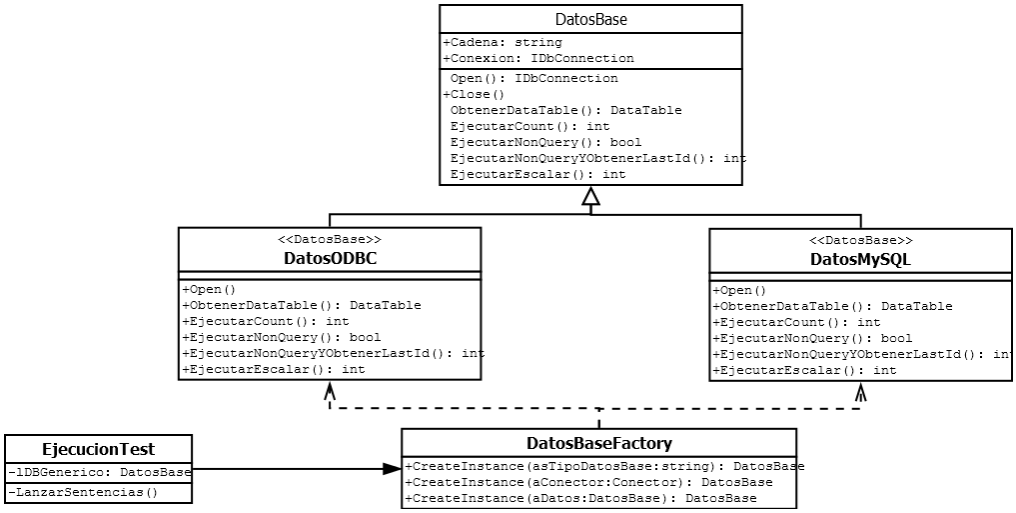
5.3.1. El patrón **Factory**

Se encarga de centralizar en una clase la creación de objetos de un subtipo de un tipo determinado.

Existen dos objetos **DatosMySQL** y **DatosODBC,** que heredan de la clase **DatosBase** que es de tipo abstract, estos dos objetos implementan los métodos de la clase base, para acceder a la base de datos, ya que cada tipo lo realiza usando objetos del tipo adecuado.

En el proyecto, el objeto **EjecucionTest**, al lanzar el proceso que ejecuta los test, requiere instanciar un objeto de tipo **DatosBase**, este objeto es una especificación abstracta, que no

se puede instanciar, ya que según el tipo de conexión, se deberá de acceder al tipo adecuado, para ello se llama al objeto **DatosBaseFactory**, que es el encargado de decidir que objeto en cuestión es el que se debe instanciar y retornarlo ya iniciado como tipo **DatosBase,** de esta forma se saca fuera de **EjecucionTest,** la dependencia de los tipos específicos.



28 Implementación patrón Factory

De esta forma se permite que en el futuro se pudieran añadir otros conectores a base de datos (SQL Server nativo o postgreSQL nativo) heredando de **DatosBase** implementando los métodos específicos y delegando en **DatosBaseFactory** la creación del nuevo tipo, sin necesidad de añadir código en cada parte de la aplicación que vaya a usar dichos objetos.

5.3.2. El patrón **Singleton**

Se encarga de garantizar la existencia de una única instancia de una clase y la creación de un mecanismo de acceso global a dicha instancia.

Hemos utilizado este patrón, para asegurarnos que el objeto principal del programa **EjecucionTest**, solo se instancie una vez, ya que es el encargado de ejecutar el proceso del test y no nos interesa que se pueda lanzar varias veces.

```
// Instancia de nuestro objeto
private static volatile EjecucionTest instance;
// Objeto que se usara a modo de bloqueo
private static object syncRoot = new Object();

// Contructor privado, para impedir que se instancie directamente.
private EjecucionTest(List<Conector> aConectores, Test aTest)
{
    this._Conectores = aConectores;
    this._Test = aTest;
    this._Resultados = new List<ResultadoTest>();
}

// Método estático que devuelve una única instancia de "EjecucionTest" patrón singleton
public static EjecucionTest GetInstance(List<Conector> aConectores, Test aTest)
{
    if (instance == null)
    {
        lock (syncRoot)
        {
            if (instance == null)
            {
                instance = new EjecucionTest(aConectores, aTest);
            }
        }
    }
    return instance;
}
```

29 Código implementación singleton

La forma de implementarlo es creando una propiedad privada (**instance**) que contendrá la instancia de nuestro objeto, otra propiedad privada de tipo objeto (**syncRoot**), que se usara únicamente a modo de bloqueo, mientras instanciamos nuestro objeto, para impedir que otros hilos pudieran instanciar en el mismo momento una copia.

El constructor del objeto lo haremos privado, para impedir que pueda ser instanciado directamente, y crearemos un método público que se encargue de instanciar a nuestro objeto, en este caso le hemos llamado **GetInstance**. Este método comprobara si nuestra propiedad **instance**, no está iniciada, entonces realiza un bloqueo sobre la propiedad **syncRoot**, y entonces, si **instance** sique estando sin iniciar, instancia nuestro objeto **EjecucionTest,** llamando al constructor privado y lo asigna a la propiedad **instance**, y finalmente retorna la instancia.

A raíz de los comentarios recibidos, como se comenta en el apartado **Control de calidad y pruebas**, por la gente que ha probado la aplicación en sus primeras versiones, también se ha implementado una variación del patrón singleton, a nivel de aplicación, para impedir que la propia aplicación pueda ser arrancada varias veces, en caso de intentar ejecutarla por segunda vez, la aplicación se cierra, trayendo al frente de la pantalla la primera ejecución.

En este caso, en el arranque de la aplicación instanciamos un objeto de tipo Mutex, este objeto nos permite realizar bloqueos a nivel del sistema operativo, se usa para sincronizar

hilos, aunque nosotros solo lo usaremos para realizar un bloqueo con un nombre, de forma que no se pueda volver a realizar dicho bloqueo, así la siguiente vez que se arranque, al intentar realizar este bloqueo fallara, indicándonos que ya estaba la aplicación arrancada.

```
[DllImport("user32.dll")]
private static extern int SetForegroundWindow(IntPtr hWnd);
[DllImport("user32.dll")]
static extern bool ShowWindow(IntPtr hWnd, int nCmdShow);
[DllImport("user32.dll")]
public static extern IntPtr FindWindow(string className, string windowTitle);

public frmPrincipal()
{
    // Mutex para controlar la ejecución de esta aplicación por una única instancia.
    _Mutex = new Mutex(true, "TestsSGBD", out appEnEjecucion);

    if (!appEnEjecucion)
    {
        // La aplicación se haya en ejecución, La traemos al frente y cerramos esta
        IntPtr wdwIntPtr = FindWindow(null, this.Text);
        if (!wdwIntPtr.Equals(IntPtr.Zero))
        {
            SetForegroundWindow(wdwIntPtr);
            ShowWindow(wdwIntPtr, SW_SHOWNORMAL);
        }
        this.Close();
    }
}
private void frmPrincipal_FormClosing(object sender, FormClosingEventArgs e)
{
    // Liberar el mutex
    if (this._Mutex != null)
    {
        this._Mutex.ReleaseMutex();
    }
}
```

30 Código arranque único de la aplicación

Es un sistema más simple y rápido que el recorrer los procesos que haya en memoria para comprobar que no esté ya arrancado, este era el sistema tradicional para realizar esto, pero requería el uso de llamadas a DLLs del operativo. El uso de Mutex es más simple, y solo requiere que lo liberemos al cerrar el form.

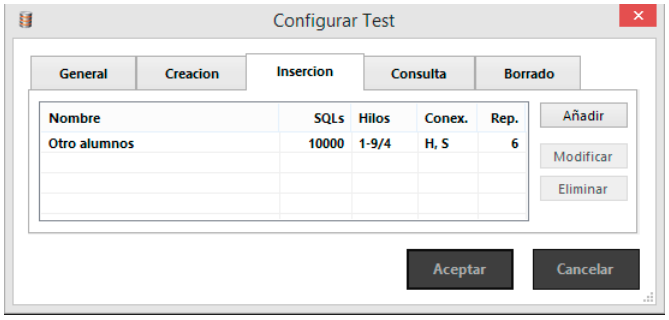
Para traer al frente la aplicación ya arrancada sí que debemos hacerlo mediante llamadas al sistema, usando **FindWindow**, para obtener el **hwnd** (handle window o identificador de ventana) de la aplicación, y **SetForegroundWindow, ShowWindow**, para traer al frente y mostrar la aplicación.

5.4. GUI

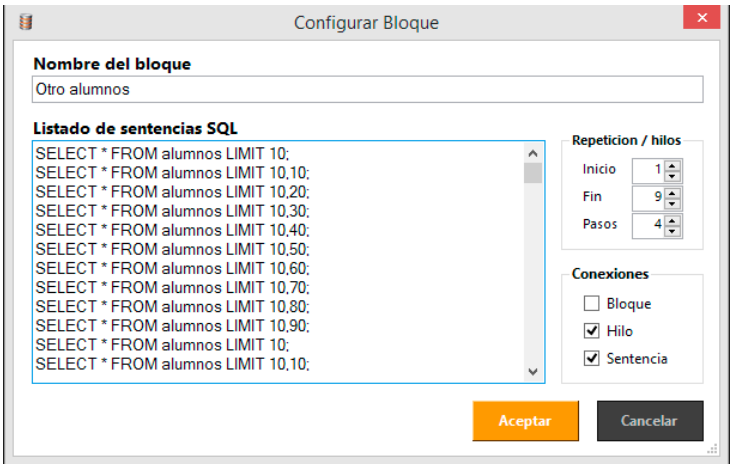
A continuación se incluyen imágenes de algunas de las pantallas finales de la aplicación.

Se ha mantenido la estética de los diseños previos, mostrados en el punto **Mockups**, aunque han aparecido nuevas opciones en la pantalla de configuración de bloques, ya que se ha ido adaptando el proyecto a los cambios para mejorar su funcionalidad.

31 Pantalla principal



32 Pantalla de configuración de Test



33 Pantalla configuración bloque

6. Control de calidad y pruebas

6.1. Test

Existen distintos tipos de test, y cada uno de estos sirve para asegurar el funcionamiento de una parte específica del proyecto.

- **Test de integración:** Como nuestra aplicación no es web, ni requiere de integración continua, ni nada por el estilo, no tiene ningún sentido que realizáramos este tipo de test.
- **Test de interfaz:** Continuando con el punto anterior, al no tratarse de una web, y al tener muy pocas pantallas, se tomó la decisión de no realizar este tipo de test, ya que son muy difíciles de realizar, y solo tienen sentido en aplicaciones más grandes o aplicación web más complejas, en una aplicación tan simple a nivel de interfaz, como la que nos ocupa, el tiempo empleado en implantar este tipo de test, no iba a ser rentable, para la poca utilidad que nos iba a aportar.
- **Test de funcionalidad:** Son test más específicos que los unitarios, y sus prueban se centran en la funcionalidad en sí, todavía está en proceso de aprobación el realizar este tipo de test.
- **Test unitarios:** Como ya se indicó en el apartado **TDD** se han realizado test unitarios con cada uno de los objetos de la aplicación, para tener cubierto al máximo posible el código de la aplicación, en estos momentos hay 166 test unitarios realizados.

6.2. QA

Para el control de calidad, decidimos presentar versiones Beta a un grupo de conocidos, explicándoles en qué consistía la aplicación y pidiéndoles que la probaran y nos reportaran sus opiniones y sus sugerencias para mejorar la aplicación.

De esta lista, seleccionamos las que nos parecieron más relevantes:

- Cuando haya un test en ejecución que de alguna forma sepamos que está funcionando, ya que hasta que finaliza, no sabemos si la aplicación está haciendo algo, o simplemente no funciona.
- Cuando haya un test en ejecución, que me impida cerrar la aplicación o al menos que me avise.
- Poder parar la ejecución de un test, y no tener que esperar a que este acabe.
- Que no pueda ejecutar varias veces la aplicación. Ya que puedo arrancarla muchas veces, ejecutando el test en cada una de las aplicaciones arrancadas.
- Que cuando algo no se pueda hacer, de un poco más de información, ya que los mensajes de error, son muy vagos.
- Cuando he ejecutado la aplicación varias veces, siempre me arranca en la sección de configuración, una vez que ya tengo la aplicación configurada, las siguientes veces me interesa arrancarla para ejecutar test o ver los informes de los que ya he realizado.

- Que exista algún tipo de ayuda, para saber que tenemos que hacer exactamente, ya que la primera vez puede resultar algo confuso.

En base a estos comentarios, modificamos la aplicación para integrar una serie de mejoras, que consideramos que eran necesarias.

- Incorporamos un Token de cancelación, al objeto **EjecucionTest**, para poder parar en cualquier momento un test en ejecución. Este Token lo reciben cada uno de los Task que se encargan de instanciar los hilos, y sirve para que cada hilo sepa si en algún momento hay pendiente la cancelación del proceso, y así cerrarse.
- Incorporamos eventos al objeto **EjecucionTest**, para que pudiera ir reportando en que paso estaba de la ejecución del test, y de esta forma poder incorporar esta información en la pantalla principal.
- Si la aplicación se intenta cerrar estando un test en ejecución, nos avisa y nos permite cancelar el cierre.
- La aplicación al arrancar comprueba que no esté en ejecución para impedir que se arranque varias veces.
- A la pantalla principal, le añadimos una barra de estado, que se refresca cada segundo, en esta barra, aparece en todo momento la memoria que está consumiendo la aplicación, y si un test está en ejecución nos informa del estado en el que se encuentra.
- Hemos personalizado los mensajes de error, para que sean más descriptivos y den información más detallada al usuario.
- En el arranque de la aplicación, comprobamos si existen resultados de test, para arrancar directamente en la sección **Informes**, si existen conectores y test configurados, para arrancar en la sección **Test,** o si nos falta algún tipo de configuración, para arrancar en la sección **Configuración.**
- Se han añadido unas pequeñas ayudas en algunos formularios de la aplicación y se deja pendiente el realizar una ayuda más completa y acorde.

7. Resultados y conclusiones

7.1. Estado actual

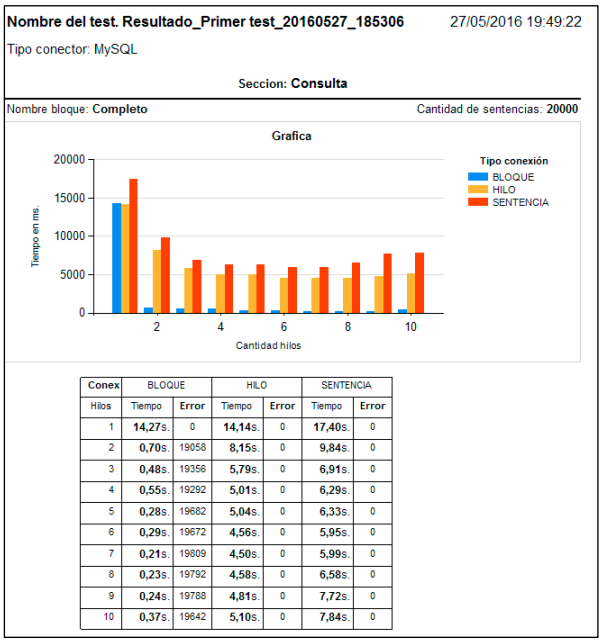
La aplicación se ha realizado completamente, teniendo una aplicación totalmente funcional que nos cubre todos los objetivos iniciales, menos uno.

- Configurar los conectores a Base de Datos que podremos usar.
- Configurar los Test, con cada una de sus secciones.
- Permitir ejecutar un Test sobre una lista de conectores
- Guardar los resultados en XML
- Permitir abrir estos resultados y convertirlos en PDF con un formato más visual con gráficas.

El único objetivo que no ha sido cubierto, es el permitirnos comparar varios resultados, sí que es cierto que la aplicación nos permite abrir varios informes, pero directamente no realiza comparaciones entre ellos, dejando esta parte en manos del usuario.

7.2. Conclusiones

Con los datos recogidos de la aplicación (se adjuntan imágenes), y en la máquina que hemos probado la aplicación, un i7 con 4 núcleos y 8 hilos, hemos llegado a una serie de conclusiones.



Nombre del test. Resultado_Primer test_20160527_185306 27/05/2016 19:49:22

Tipo conector: MySQL

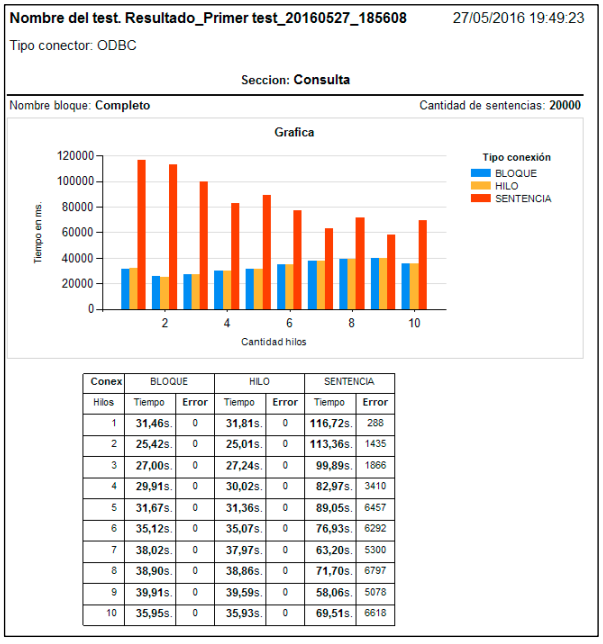
Seccion: Consulta

Nombre bloque: **Completo** Cantidad de sentencias: **20000**

Grafica

Conex	BLOQUE		HILO		SENTENCIA	
Hilos	Tiempo	Error	Tiempo	Error	Tiempo	Error
1	14,27s.	0	14,14s.	0	17,40s.	0
2	0,70s.	19058	8,15s.	0	9,84s.	0
3	0,48s.	19356	5,79s.	0	6,91s.	0
4	0,55s.	19292	5,01s.	0	6,29s.	0
5	0,28s.	19682	5,04s.	0	6,33s.	0
6	0,29s.	19672	4,56s.	0	5,95s.	0
7	0,21s.	19809	4,50s.	0	5,99s.	0
8	0,23s.	19792	4,58s.	0	6,58s.	0
9	0,24s.	19788	4,81s.	0	7,72s.	0
10	0,37s.	19642	5,10s.	0	7,84s.	0

34 Resultado conector nativo MySQL



Nombre del test. Resultado_Primer test_20160527_185608 27/05/2016 19:49:23

Tipo conector: ODBC

Seccion: Consulta

Nombre bloque: **Completo** Cantidad de sentencias: **20000**

Grafica

Conex	BLOQUE		HILO		SENTENCIA	
Hilos	Tiempo	Error	Tiempo	Error	Tiempo	Error
1	31,46s.	0	31,81s.	0	116,72s.	288
2	25,42s.	0	25,01s.	0	113,36s.	1435
3	27,00s.	0	27,24s.	0	99,89s.	1866
4	29,91s.	0	30,02s.	0	82,97s.	3410
5	31,67s.	0	31,36s.	0	89,05s.	6457
6	35,12s.	0	35,07s.	0	76,93s.	6292
7	38,02s.	0	37,97s.	0	63,20s.	5300
8	38,90s.	0	38,86s.	0	71,70s.	6797
9	39,91s.	0	39,59s.	0	58,06s.	5078
10	35,95s.	0	35,93s.	0	69,51s.	6618

35 Resultado conector ODBC

- **ODBC**

Los conectores **ODBC** nos aportan muchas ventajas al permitirnos conectarnos con cualquier origen de datos que el sistema reconozca, pero al ser una capa sobre el conector nativo del origen de datos, siempre va a ser una solución más lenta, como queda reflejado en los test, en el caso de MySQL, el conector ODBC en el mejor de los casos en ambos es cinco veces más lento que el nativo. (**Nativo 7 hilos en 4,50s.frente ODBC 2hilos 25.01s.**)

También hemos comprobado que ODBC tiene una gran penalización en la apertura y cierre de las conexiones, y cuando tenemos varios hilos que realizan una conexión por cada sentencia SQL que van a lanzar, el conector ODBC comienza a dar errores de conexión, volviéndose muy inestable. Si bien es cierto que para llegar a esta situación una aplicación web debería de tener un gran tráfico, más de 200 peticiones por segundo, y en estos casos no creo que hubiera ningún problema económico como para no poder migrarla y solucionar el problema.

También hemos detectado que dependiendo de la versión de ODBC, no está muy optimizada para la concurrencia, siendo muy similares los resultados obtenidos independientemente el número de hilos que se lanzaran.

- **MySQL nativo**

Es más rápido que ODBC, esta optimizado para la concurrencia, notando una gran mejora entre ejecutarlo en un único hilo o en varios dependiendo del procesador de la maquina donde se ejecute, y decayendo muy poco, cuando son más hilos de los que la maquina realmente posee, soportando una gran concurrencia sin demasiada penalización.

También hemos comprobado, que el conector nativo, al estar tan adaptado a la concurrencia, cuando configuramos una única conexión y la intentamos usar desde distintos hilos, comienza a dar errores y es demasiado inestable, algo totalmente lógico, ya que no es un uso razonable de la conexión, y se producen bloqueos entre hilos. Aunque el conector ODBC parece que internamente se encargue de gestionar estos bloqueos, ya que por ODBC no se produce ni un solo fallo.

Para finalizar, la conclusión clara de las pruebas realizadas es que si tienes un conector nativo para tu origen de datos úsalo solo en el caso de no tener un conector nativo, usa el conector por ODBC, y jamás establezcas una conexión global, que cada hilo ejecute su propia conexión.

7.3. Posibilidades futuras

Debido a las limitaciones del proyecto, se ha tenido que sacrificar algunas opciones que queríamos realizar desde el principio, y mientras realizábamos el proyecto, nos han surgido otras ideas que tampoco tenían cabida realizarlas.

- Implementación de un pseudo lenguaje, muy simple, pero que nos permitiera generar de forma automática las sentencias de cada uno de los bloques del test, y no tener que generarlas a mano, algo del tipo "(SELECT * FROM tabla WHERE nombre='randomAlf(8)') x 1000".
- Permitir comparar un bloque con otro, que tenga sentencias que retornen los mismos resultados, para comprobar cual SQL es más adecuada, para ello habría que recoger y guardarse también el resultado de cada sentencia, o en su defecto ir calculando un hash de los resultados, para poder compararlos estos.
- Implementar apartados de importación/exportación, para poder llevarnos test de una maquina a otra.
- Permitir comparar un test contra otro, y sacar un PDF de la comparación. En este punto nos surgen más opciones:
    - Deberíamos de guardarnos un hash que identifique al test, de forma que solo se pudieran comparar dos test idénticos entre sí, sino no tendría ningún sentido.
    - Deberíamos de guardarnos un hash de la maquina donde se ha ejecutado el test, de forma que al compararlos, si son distintos, nos indique en la comparativa que pertenecen a maquinas diferentes y las comparaciones no son fiables.
- Generar una ayuda en condiciones, que nos sirva tanto como ayuda en línea **F1** como manual de usuario.

7.4. Valoración personal

Personalmente estoy muy satisfecho con el proyecto en sí, he cubierto todos los objetivos iniciales, menos uno, que no ha sido posible, no por su dificultad sino por el proyecto en sí, es un proyecto académico con un margen de tiempo reducido, y hay que ser realistas en lo que se puede o no incluir.

Además he aprovechado el proyecto, para cubrir otros asuntos:

- Aprender a usar TDD dentro de .NET, era una deuda técnica personal, y he acabado muy contento con ello, ya que he comprobado que dentro de .NET el realizar test unitarios es un proceso muy sencillo y aporta una gran seguridad.
- Aprender a usar los Reports de .NET, ya que dentro de .NET siempre he usado Crystal Report y en java Jasper Report, y quería librarme de esta dependencia, si bien es cierto, que los Reports de .NET no se pueden comparar con Crystal, para informes sencillos son perfectos.

  Aunque los informes del proyecto, parezcan muy simples, el obtener las agrupaciones de las tablas, por hilos en horizontal y por bloques en vertical y los gráficos, agrupados por bloques en cada grupo de hilos, tiene su complejidad.

- He practicado los principios del Clean Code en la medida de lo posible y no he dejado de Refactorizar, aunque esto es algo que se debe intentar seguir siempre ya que es un camino y no un destino.

8. Anexos

A. XML de ejemplo del fichero Conectores.xml

```
<?xml version="1.0"?>
<Conectores xmlns="https://pruebarana.dyndns-remote.com/CFGS/DAM">
  <Conector Nombre="MySQL Nativo">
    <Tipo>MySQL</Tipo>
    <Cadena>
      Server=127.0.0.1; Port=3305; Database=aptavs_aplicacion_8765; Uid=aptavs;
      Pwd=unacontraseña;
    </Cadena>
  </Conector>
  <Conector Nombre="MySQL ODBC">
    <Tipo>ODBC</Tipo>
    <Cadena>
      Driver={MySQL ODBC 5.3 Unicode Driver}; Server=localhost; Port=3305;
      Database=aptavs_aplicacion_8765; User=aptavs; Password=unacontraseña;
    </Cadena>
  </Conector>
</Conectores>
```

B. XML de ejemplo del fichero Tests.xml

```
<?xml version="1.0"?>
<Tests xmlns="https://pruebarana.dyndns-remote.com/CFGS/DAM">
  <TestInfo
     Nombre="Primer test"
     File="E:\NuevoRana\FP\Proyectos\DAM\Config\Primer test.XML" />
</Tests>
```

C. XML de ejemplo del objeto Test

```
<?xml version="1.0"?>
<Test Nombre="Primer test" xmlns="https://pruebarana.dyndns-remote.com/CFGS/DAM">
  <Creacion>
    <Bloque Conexion="BLOQUE" Nombre="Creacion de alumnos"
            Hilos_Step="1" Hilos_Inicio="1" Hilos_Fin="1">
    </Bloque>
  </Creacion>
  <Insercion>
    <Bloque Conexion="HILO" Nombre="Alta alumnos"
            Hilos_Step="1" Hilos_Inicio="1" Hilos_Fin="1">
      <Sentencia SQL="INSERT INTO Alumnos (nombre, dni, correo) VALUES ('Jaime
          Aguila', '25413368R', 'jaguila@pymsolutions.com')" />
      <Sentencia SQL="INSERT INTO Alumnos (nombre, dni, correo) VALUES ('Juan',
          '25413367R', 'juan@pymsolutions.com')" />
      <Sentencia SQL="INSERT INTO Alumnos (nombre, dni, correo) VALUES ('Perico',
          '25413366R', 'perico@pymsolutions.com')" />
      <Sentencia SQL="INSERT INTO Alumnos (nombre, dni, correo) VALUES ('Andres',
          '25413365R', 'andres@pymsolutions.com')" />
      <Sentencia SQL="INSERT INTO Alumnos (nombre, dni, correo) VALUES ('Prueba
          33', '25413364R', 'prueba@pymsolutions.com')" />
      <Sentencia SQL="INSERT INTO Alumnos (nombre, dni, correo) VALUES ('Pepito 2',
          '25413363R', 'pepito@pymsolutions.com')" />
      <Sentencia SQL="INSERT INTO Alumnos (nombre, dni, correo) VALUES ('Goyo',
          '25413362R', 'goyo@pymsolutions.com')" />
    </Bloque>
    <Bloque Conexion="HILO" Nombre="Modificacion alumnos"
            Hilos_Step="1" Hilos_Inicio="1" Hilos_Fin="1">
      <Sentencia SQL="UPDATE Alumnos SET nombre = 'prueba' WHERE nombre = 'Prueba
          33'" />
      <Sentencia SQL="UPDATE Alumnos SET nombre = 'pepito pelayo' WHERE dni =
          '25413363R'" />
    </Bloque>
  </Insercion>
  <Consulta>
    <Bloque Conexion="HILO SENTENCIA" Nombre="Consulta de alumnos"
            Hilos_Step="1" Hilos_Inicio="1" Hilos_Fin="5">
      <Sentencia SQL="SELECT * FROM alumnos LIMIT 10;" />
      <Sentencia SQL="SELECT * FROM alumnos LIMIT 20,10;" />
      <Sentencia SQL="SELECT * FROM alumnos LIMIT 30,10;" />
      <Sentencia SQL="SELECT * FROM alumnos LIMIT 40,10;" />
      <Sentencia SQL="SELECT * FROM alumnos LIMIT 50,10;" />
      <Sentencia SQL="SELECT * FROM alumnos LIMIT 60,10;" />
      <Sentencia SQL="SELECT * FROM alumnos LIMIT 70,10;" />
      <Sentencia SQL="SELECT * FROM alumnos LIMIT 80,10;" />
      <Sentencia SQL="SELECT * FROM alumnos LIMIT 90,10;" />
      <Sentencia SQL="SELECT * FROM alumnos LIMIT 100,10;" />
    </Bloque>
  </Consulta>
  <Borrado>
    <Bloque Conexion="BLOQUE" Nombre="Borrados de alumnos"
            Hilos_Step="1" Hilos_Inicio="1" Hilos_Fin="1">
    </Bloque>
  </Borrado>
</Test>
```

D. XML de ejemplo del objeto ResultadoTest

```
<?xml version="1.0"?>
<ResultadoTest
       Nombre="Resultado_Primer test_20160513_153444"
       Fecha="2016-05-13T15:34:44.130494+02:00"
       xmlns="https://pruebarana.dyndns-remote.com/CFGS/DAM">
  <Conector Nombre="MySQL Nativo">
    <Tipo>MySQL</Tipo>
    <Cadena>Server=127.0.0.1; Port=3305; Database=aptavs_aplicacion_8765;
        Uid=aptavs;Pwd=****;</Cadena>
  </Conector>
  <Insercion>
    <Bloque Nombre="Alta alumnos" NumeroSentencias="10000">
      <Conexion Tipo="HILO">
        <Hilo Cantidad="1" Tiempo="8177" Errores="0" />
        <Hilo Cantidad="3" Tiempo="3456" Errores="0" />
        <Hilo Cantidad="5" Tiempo="2945" Errores="0" />
      </Conexion>
      <Conexion Tipo="SENTENCIA">
        <Hilo Cantidad="1" Tiempo="9561" Errores="0" />
        <Hilo Cantidad="3" Tiempo="4232" Errores="0" />
        <Hilo Cantidad="5" Tiempo="4112" Errores="0" />
      </Conexion>
    </Bloque>
  </Insercion>
  <Consulta>
    <Bloque Nombre="Consulta de alumnos" NumeroSentencias="10000">
      <Conexion Tipo="HILO">
        <Hilo Cantidad="1" Tiempo="8177" Errores="0" />
        <Hilo Cantidad="9" Tiempo="3456" Errores="0" />
        <Hilo Cantidad="17" Tiempo="2945" Errores="0" />
        <Hilo Cantidad="25" Tiempo="2894" Errores="0" />
        <Hilo Cantidad="33" Tiempo="2968" Errores="0" />
        <Hilo Cantidad="41" Tiempo="3092" Errores="0" />
      </Conexion>
      <Conexion Tipo="SENTENCIA">
        <Hilo Cantidad="1" Tiempo="9561" Errores="0" />
        <Hilo Cantidad="9" Tiempo="4232" Errores="0" />
        <Hilo Cantidad="17" Tiempo="4112" Errores="0" />
        <Hilo Cantidad="25" Tiempo="4654" Errores="0" />
        <Hilo Cantidad="33" Tiempo="4571" Errores="0" />
        <Hilo Cantidad="41" Tiempo="4487" Errores="0" />
      </Conexion>
    </Bloque>
  </Consulta>
  <Borrado>
    <Bloque Nombre="Borrados de alumnos" NumeroSentencias="0">
    </Bloque>
  </Borrado>
</ResultadoTest>
```

E. Código frmPrincipal botones laterales sin refactorizar

```
private void btnConfiguracion_Click(object sender, EventArgs e)
{
    this._Seccion = "btnConfiguracion";
}
private void btnConfiguracion_MouseLeave(object sender, EventArgs e)
{
    if(this._Seccion == "btnConfiguracion")
    {
        btnConfiguracion.BackColor = Color.FromArgb(64, 64, 64);
        btnConfiguracion.ForeColor = Color.FromArgb(255, 153, 0);
    }
    else
    {
        btnConfiguracion.BackColor = Color.FromArgb(64, 64, 64);
        btnConfiguracion.ForeColor = Color.Silver;
    }
}
private void btnConfiguracion_MouseEnter(object sender, EventArgs e)
{
    btnConfiguracion.BackColor = Color.FromArgb(255, 153, 0);
    btnConfiguracion.ForeColor = Color.White;
}
private void btnTest_Click(object sender, EventArgs e)
{
    this._Seccion = "btnTest";
}
private void btnTest_MouseLeave(object sender, EventArgs e)
{
    if(this._Seccion == "btnTest")
    {
        btnTest.BackColor = Color.FromArgb(64, 64, 64);
        btnTest.ForeColor = Color.FromArgb(255, 153, 0);
    }
    else
    {
        btnTest.BackColor = Color.FromArgb(64, 64, 64);
        btnTest.ForeColor = Color.Silver;
    }
}
private void btnTest_MouseEnter(object sender, EventArgs e)
{
    btnTest.BackColor = Color.FromArgb(255, 153, 0);
    btnTest.ForeColor = Color.White;
}
private void btnInforme_Click(object sender, EventArgs e)
{
    this._Seccion = "btnInforme";
}
private void btnInforme_MouseLeave(object sender, EventArgs e)
{
    if(this._Seccion == "btnInforme")
    {
        btnInforme.BackColor = Color.FromArgb(64, 64, 64);
        btnInforme.ForeColor = Color.FromArgb(255, 153, 0);
    }
    else
    {
        btnInforme.BackColor = Color.FromArgb(64, 64, 64);
        btnInforme.ForeColor = Color.Silver;
    }
}
private void btnInforme_MouseEnter(object sender, EventArgs e)
{
    btnInforme.BackColor = Color.FromArgb(255, 153, 0);
    btnInforme.ForeColor = Color.White;
}
```

F. Código frmPrincipal botones laterales refactorizado

```
private void btnSeccion_Click(object sender, EventArgs e)
{
    this._Seccion = (Button)sender).Name;
}
private void btnSeccion_MouseLeave(object sender, EventArgs e)
{
    ColorBotonDesactivado(btnConfiguracion);
    ColorBotonDesactivado(btnTests);
    ColorBotonDesactivado(btnInformes);

    Button lB = (Button)this.Controls.Find(this._Seccion, true)[0];
    if (lB != null)
    {
        ColorBotonActivado(lB);
    }
}
private void btnSeccion_MouseEnter(object sender, EventArgs e)
{
    ColorBotonResaltado((Button)sender);
}


private void ColorBotonDesactivado(Button aButton)
{
    aButton.BackColor = Color.FromArgb(64, 64, 64);
    aButton.ForeColor = Color.Silver;
}
private void ColorBotonActivado(Button aButton)
{
    aButton.BackColor = Color.FromArgb(64, 64, 64);
    aButton.ForeColor = Color.FromArgb(255, 153, 0);
}
private void ColorBotonResaltado(Button aButton)
{
    aButton.BackColor = Color.FromArgb(255, 153, 0);
    aButton.ForeColor = Color.White;
}
```

9. Referencias y bibliografía

Enlaces de internet

- .Net Framework - https://msdn.microsoft.com/es-es/library/hh425099(v=vs.110).aspx
- .Net Framework - https://es.wikipedia.org/wiki/Microsoft_.NET
- C# - https://es.wikipedia.org/wiki/C_Sharp#cite_note-1
- Visual Studio - https://en.wikipedia.org/wiki/Microsoft_Visual_Studio
- ADO.NET - http://www.ehu.eus/mrodriguez/archivos/csharppdf/ADONET/ADONET.pdf
- XML - https://es.wikipedia.org/wiki/Extensible_Markup_Language
- TDD - https://es.wikipedia.org/wiki/Desarrollo_guiado_por_pruebas
- Patrones de diseño - https://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o
- XP wiki - https://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema
- Refactorización Wiki - https://es.wikipedia.org/wiki/Refactorizaci%C3%B3n
- Code Smell Wiki - https://es.wikipedia.org/wiki/Hediondez_del_c%C3%B3digo

Libros

- **Extreme Programming Explained**, Kent Beck
- **Test Unitarios, metodología AAA** - http://librosweb.es/libro/tdd/capitulo_5/las_tres_partes_del_test_aaa.html
- **Código Limpio**: A Handbook of Agile Software Craftsmanship, Robert C. Martin
- **Design Patterns: Elements of Reusable Object-Oriented Software**, Gang of Four