BLOQUE 3

6. Test de Rendimiento

En este proyecto se están estudiando los diferentes sistemas de bases de datos existentes en el mercado, para conocer sus características y contar así con suficiente información para determinar cuál de estas bases de datos es la más adecuada para los intereses particulares de una persona u organización.

En el caso particular de este proyecto, se va a realizar una aplicación que realice la compilación, ejecución y validación de código de alumnos de la asignatura Programación Orientada a Objetos del Grado de Ingeniería de Sistemas de Telecomunicación, y para dar soporte a ello, se va a hacer uso de una base de datos, que va a almacenar el código fuente, los resultados y calificaciones de los alumnos.

La mayor carga de procesos y datos se dará a la hora de compilar el código fuente y su ejecución. Estos códigos estarán escritos en lenguaje Java, y serán operados en servidor, por lo que es imprescindible que el tiempo asociado a la inserción, selección y/o modificación de archivos sea mínima para no sobrecargar más aún el proceso.

6.1 Objetivo del Test de rendimiento

En base a lo anteriormente expuesto, el objetivo de este test de rendimiento será el de encontrar el sistema gestor de bases de datos que opere con mayor velocidad en las operaciones de inserción, selección y actualización de datos y archivos.

Para lograr este propósito se van a estudiar los siguientes sistemas de bases de datos, muy destacados en cada una de las áreas:

Tipo de Base de Datos	Nombre
Relacional	MySQL
	SQL Server
NoSQL	MongoDB
XML Nativo	Marklogic

6.2 Programación

Para llevar a cabo este Test de rendimiento, se ha utilizado el lenguaje de programación C#, que muestra una flexibilidad muy potente y un amplio y rico soporte para los sistemas anteriormente mencionados, soportado por los propietarios o por la comunidad de usuarios.

Cada una de las soluciones estudiada tiene unos drivers para poder utilizar C# con el fin de comunicarse con estas bases de datos.

La programación de este test ha sido compleja ya no tanto por la dificultad de la prueba, sino por las importantes diferencias tecnológicas entre las bases de datos estudiadas: Relacionales, NoSQL y XML Nativas.

La base del programa desarrollado consiste en la inserción de los mismos datos N veces, con una única diferencia, que es la asignación de un ID único para cada uno de estos registros. Posteriormente se harán N selecciones, una por cada elemento, y finalmente, la actualización de un campo, (el de correo electrónico), para cada uno de los elementos.

A continuación se muestra la tipología de los datos empleados:

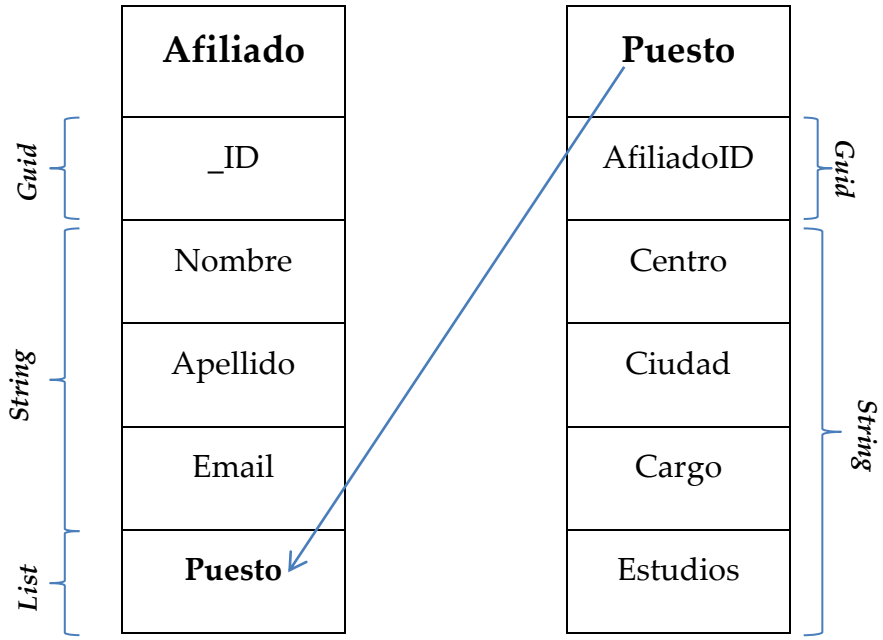

Figura 22: Estructura de datos manipuladas en las Bases de datos

El programa cuenta con dos clases principales, una de ellas es la principal, llamada *Program*, que contiene el método main() desde el que se llamará a todas las funciones, que estarán alojadas en la clase *Comandos*.

6.2.1 Conexión a las bases de datos

La conexión a las bases de datos depende completamente del driver con el que estemos trabajando. A continuación se muestra de manera somera cómo se ejecutaría dicha conexión, para profundizar puede visitar el anexo 2 Test de Rendimiento.

- MySQL

```
string conex = "server=localhost; user id=root; password=passw;
database=test; pooling=false;";
MySqlConnection consql = new MySqlConnection(conex);
```

- SQL Server

```
string conex = @"Server=(localdb)\v11.0; Initial Catalog=test;Integrated
Security=True;";
SqlConnection consql = new SqlConnection(conex);
```

- MongoDB

```
string connectionString = "mongodb://127.0.0.1";
MongoServer server = MongoServer.Create(connectionString);
MongoDatabase test = server.GetDatabase("test");
MongoCollection<Afiliado> afiliados =
      test.GetCollection<Afiliado>("Afiliado");
```

- Marklogic

```
Uri uri = new Uri("xcc://user:pass@localhost:8888/Documents");
ContentSource contentSource = ContentSourceFactory.NewContentSource(uri);
Session session = contentSource.NewSession();
FileInfo file = new FileInfo("benchmarking.xml");
Content content=ContentFactory.NewContent("benchmarking.xml",file,
      options);
```

6.2.2 Inserción de datos

Para la inserción de datos en cada una de las bases de datos se han utilizado los siguientes datos, que si bien son irrelevantes en su contenido, pueden ayudar a su comprensión.

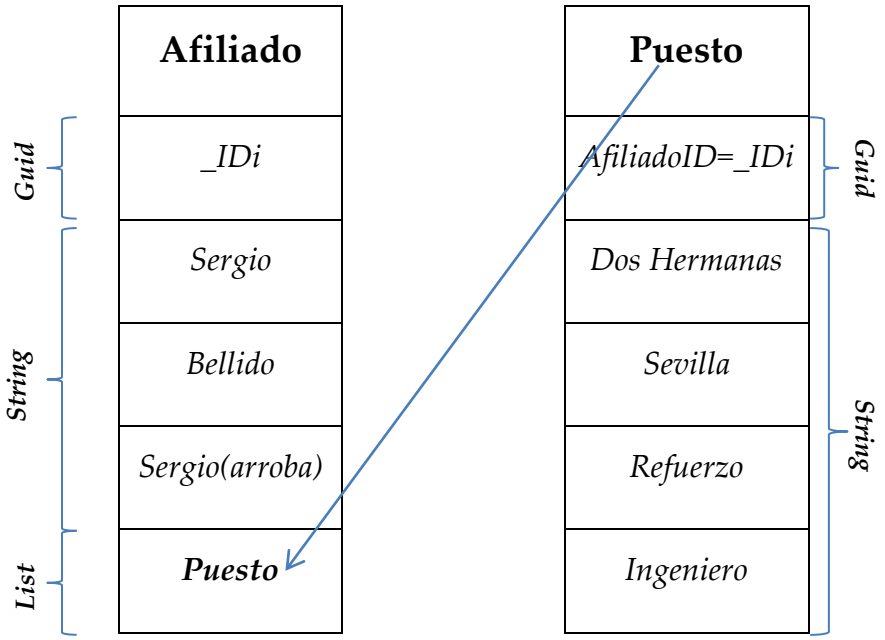

Figura 23: Datos concretos para la inserción

Previamente a la inserción de estos datos, se lleva a cabo la creación de IDs únicos aleatorios para ser asignados a cada registro, para lo que se hace uso de este sencillo código C#:

```
Guid[] guids = new Guid[500];
for (int i = 0; i < 500; i++)
    guids[i] = Guid.NewGuid();
Guid[] ids = guids;
```

MongoDB no trabaja con estos IDs, pero puede crearlos por sí mismos, por lo que se ejecuta la operación CreateIndex, que asigna un ID único al campo especificado de la instancia de la clase creada, *afiliados*:

```
afiliados.CreateIndex("_Id");
```

Posteriormente, para cada uno de las bases de datos estudiadas se llama a la función CrearFicha(), cuyo constructor se encuentra en la clase Comandos.

```
CrearFicha(ids[i],"Sergio","Bellido","sergio(arroba)","Dos Hermanas",
                                    "Sevilla", "Refuerzo","Ingeniero");
```

Esta función pasa como argumentos los datos que debe contener cada registro, e internamente crea estos valores que serán pasados a las bases de datos en futuras operaciones. Estos datos serán creados en las clases *Afiliado* y

Puesto. La función devuelve el tipo *Afiliado* creado para que pueda trabajar con ella la siguiente función: InsertaAfiliado() en sus diferentes versiones:

```
InsertarAfiliadoSQL(compasql);
InsertarSQLserver(compasqlserver);
InsertarMongoDb(compamongo);
InsertarMarklogic(compaxml, options);
```

Cada una de estas funciones tiene sus propias particularidades, adaptadas a cada driver de cada base de datos, y que pueden ser estudiadas en el Anexo 2 Test de Rendimiento. Destacar que las bases de datos relacionales utilizan lenguaje SQL para trabajar, y son ejecutadas con el comando ExecuteNonQuery().

Trabajar con MongoDB en este sentido es tremendamente sencillo, basta con llamar a la siguiente función, pasándole como parámetro la clase que contiene los datos que formarán parte del documento:

```
afiliados.Insert(user);
```

Esta facilidad para insertar datos dará posteriormente unos resultados altamente satisfactorios.

Las inserciones en Marklogic son un poco más tediosas, ya que se deben insertar individualmente cada elemento, o etiqueta XML con su contenido, lo que hace que la velocidad dependa mucho del equipo sobre el que se trabaje y la frecuencia a la que funcione.

6.2.3 Selección de Datos

La selección de datos es la función más sencilla de las creadas, ya que se pasa como parámetro un ID particular, que cada base de datos tendrá que buscar de manera interna en su sistema. Todas proporcionan funciones en C# que facilitan mucho la tarea.

Hay que tener en cuenta que en este caso, al haber una relación entre la clase Afiliado y Puesto, a través de la lista Puestos, se deberán hacer Joins para la consulta, lo que ralentiza mucho el proceso en el caso de las bases de datos relacionales.

Para profundizar en el código de selección puede visitar el anexo 2 Test de Rendimiento.

6.2.4 Actualización de Datos

La actualización de datos resultó ser la tarea más compleja de llevar a cabo, ya que el tratamiento que cada sistema da a los datos es muy diferente, y aquí resulta de gran relevancia lo bien realizado que esté el driver y su integración con la base de datos. De nuevo, veremos que MongoDB vuelve a alzarse sobre el resto con evidente superioridad.

La actualización de datos va a llevar aparejado de nuevo una creación de fichas, como para la inserción, pero en este caso se ha modificado un campo, Email, que deberá ser actualizado en la base de datos correspondiente cuando compruebe que el dato ha sido modificado. Se muestra a continuación como quedaría la nueva estructura de datos:

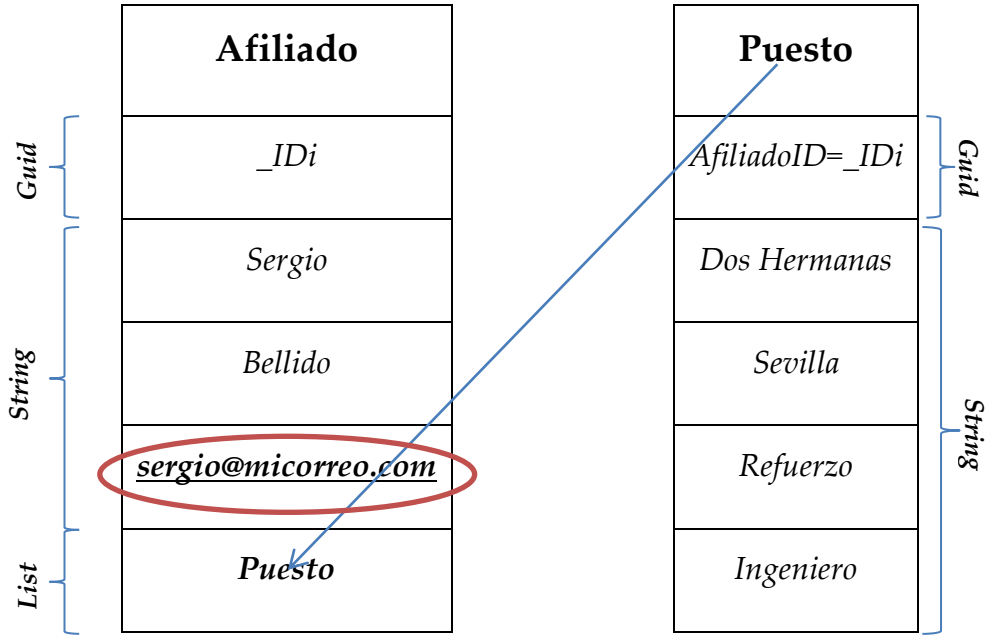

Figura 24: Datos a modificar, con variación en el campo Email

Con la creación de la ficha y su asignación a la clase *Afiliado* y *Puesto*, se llama a las siguientes funciones, dependiendo de la base de datos bajo estudio.

```
ActualizarSql(actualizado);
ActualizarSqlserver(actualizadoserver);
ActualizarMongoDb(actualizamongo);
ActualizarMarklogic(ids, "Sergio","Bellido","sergio@micorreo.com","Dos
                    Hermanas","Sevilla","Refuerzo","Ingeniero", options);
```

El tratamiento que cada driver hace de la actualización es muy diferente:

- MySQL:

```
string cadena = "Update afiliado SET “elementos””
string cadena1 = "Update puesto SET “elementos””
```

```
MySqlCommand cmd = new MySqlCommand(cadena, consql);
consql.Open();
cmd.ExecuteNonQuery();
```

- SQL Server: Mismo proceso que para MySQL server, pero con otros constructores.

```
SqlCommand cmd = new SqlCommand(cadena, consql);
consql.Open();
cmd.ExecuteNonQuery();
```

- MongoDB:

```
afiliados.Save<Afiliado>(user);
```

- Marklogic: Debe recorrer todos los elementos, y posteriormente inserter la variación, siendo poco eficiente.

```
session.InsertContent(content);
```

Resulta sorprendente de nuevo la facilidad de trato que MongoDB da a los datos, con una simple función, Save(), MongoDB es capaz de entrar en el documento concreto de la colección y actualizar los datos que sean necesarios. Esto demostrará posteriormente una velocidad de procesado muy alta.

El siguiente apartado presentará los resultados obtenidos en el test cuyas partes más destacadas se acaban de ver.

6.3 Resultados del Test

En este apartado se van a presentar los resultados del test de rendimiento al que se han sometido las cuatro bases de dato bajo estudio. Se les han realizado bucles de 500, 2000 y 5000 ciclos, en los que se han insertado, seleccionado, y actualizado registros, generando unas estadísticas que aquí se muestran.

6.3.1 Ciclo de 500 operaciones

Para un conjunto de 500 operaciones de inserción, selección y actualización de la información, los resultados obtenidos son los siguientes:

Test de Rendimiento MySQL vs. SQLServer vs. MongoDB vs. Marklogic	Operación	MYSQL	SQL Server	MongoDB	Marklogic
		Tiempo empleado en milisegundos			
	Insertar	2743	1099	195	816
	Selección	6051	890	141	3894
	Actualizar	4784	1152	47	3226

Figura 25: Resultados del test para 500 ciclos

Se presentan gráficamente los resultados anteriores:

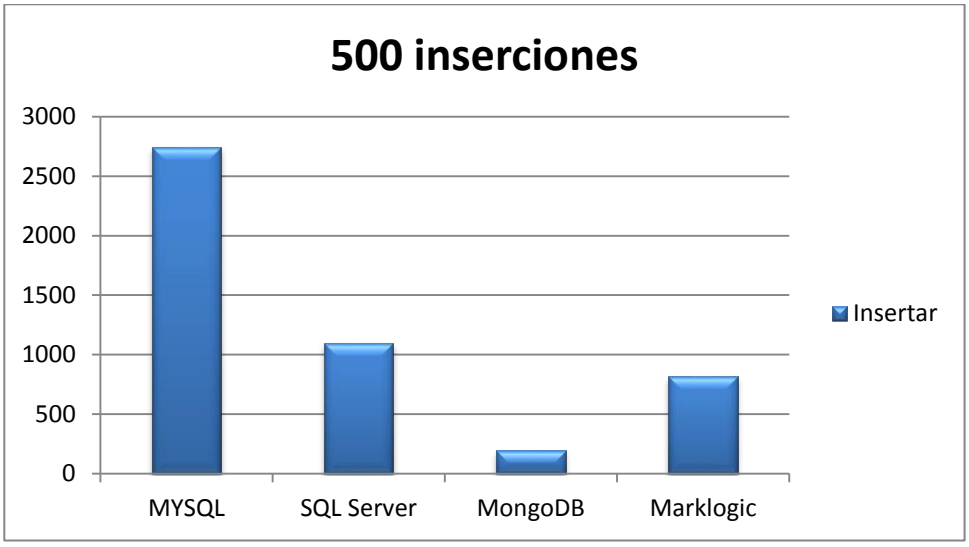

Figura 26: Tiempo en ms en realizar la operación de inserción de datos

La inserción de datos consiste en introducir una serie de datos en la base de datos, los mismos para cada una de las soluciones en este test. Los resultados demuestran una clara superioridad de MongoDB en este sentido.

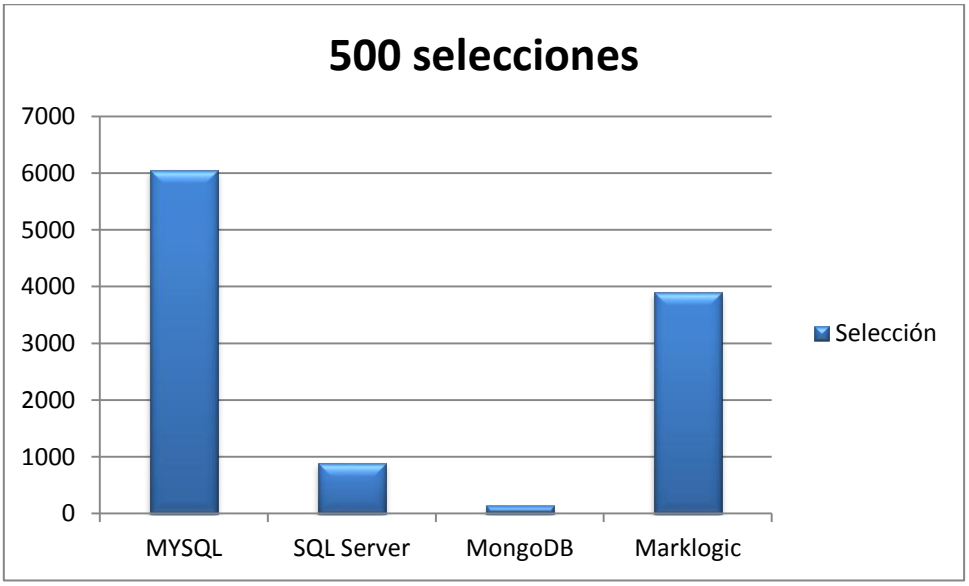

Figura 27: Test de rendimiento para 500 operaciones de selección de datos

La selección de datos se ha realizado a través del ID único asignado a cada registro. En este sentido, MongoDB se vuelve a alzar con los mejores resultados en milisegundos, seguido no muy de lejos por SQL.

Los resultados del test para la operación de actualización se presentan en la siguiente gráfica:

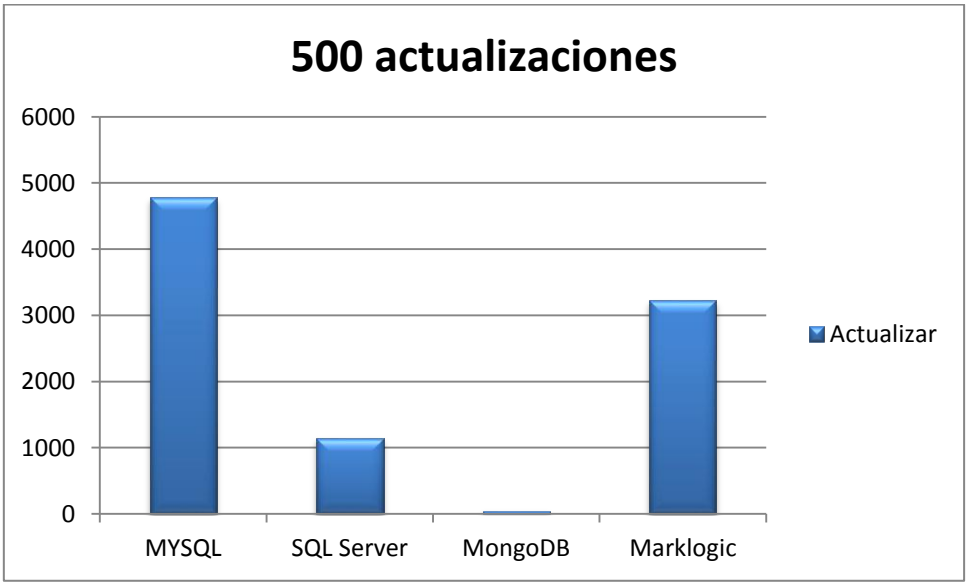

Figura 28: Resultados del Test de Rendimiento para 500 operaciones de actualización de datos

Hay que remarcar que la casi ausencia de color en la columna de MongoDB se debe a que el tiempo de respuesta en milisegundos es tan pequeño en comparación con el resto, que apenas puede apreciarse su columna.

6.3.2 Ciclo de 2000 operaciones

Para un conjunto de 2000 operaciones de inserción, selección y actualización de la información, los resultados obtenidos son los siguientes:

Test de Rendimiento MySQL vs. SQLServer vs. MongoDB vs. Marklogic	Operación	MYSQL	SQL Server	MongoDB	Marklogic
		Tiempo empleado en milisegundos			
	Insertar	12659	5625	205	1247
	Selección	68952	12274	343	14805
	Actualizar	39693	12367	197	44361

Figura 29: Resultados obtenidos para 2000 operaciones de inserción, selección y actualización

Se presentan de nuevo los resultados de forma visual para una mejor comprensión. Recordar que los resultados obtenidos se expresan en milisegundos.

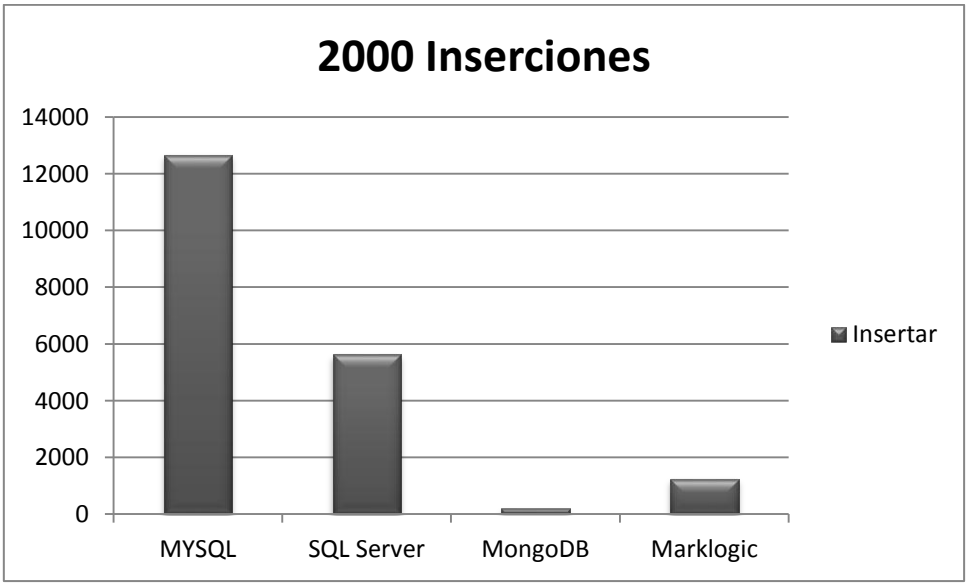

Figura 30: Resultados en ms para 2000 inserciones

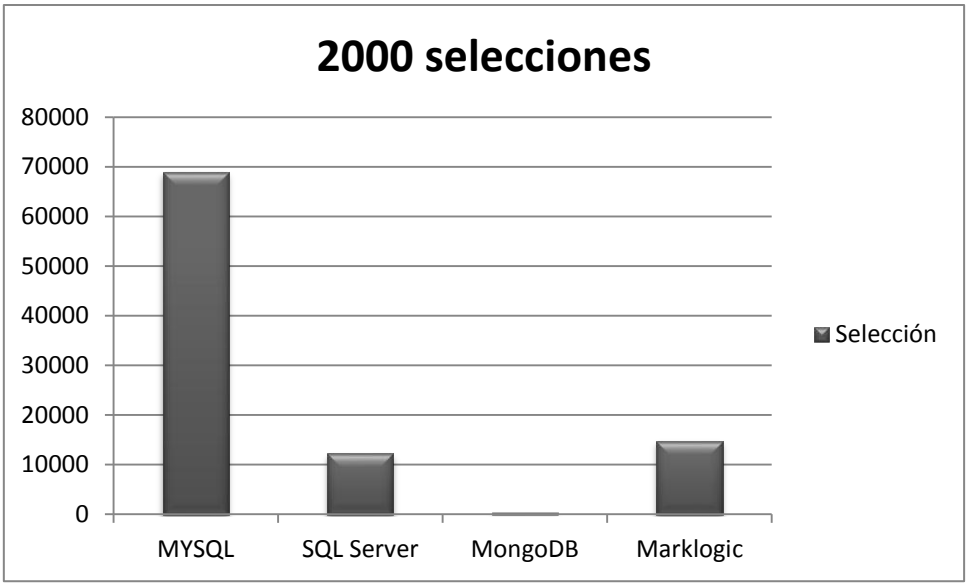

Figura 31: Resultados en ms para 2000 selecciones

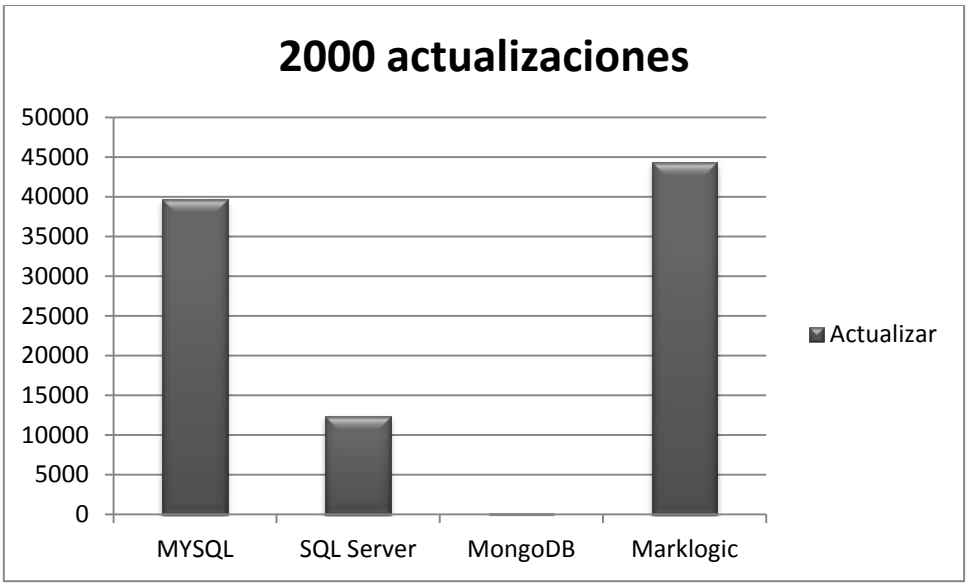

Figura 32: Resultados en ms para 2000 actualizaciones

De los resultados obtenemos una superioridad llamativa de MongoDB, que se muestra tremendamente rápida en todas las áreas bajo estudio, lo que hace pensar que su arquitectura es realmente eficaz, si bien sabemos por lo que se explicó en puntos anteriores, que se debe a no cumplir con algunas de las reglas que confieren estabilidad a los sistemas relacionales. SQL Server también presenta un buen comportamiento, muy regular en todas las áreas, mientras que Marklogic se muestra algo irregular, con buenos datos para inserciones y selecciones pero peores resultados en actualizaciones.

6.3.3 Ciclo de 5000 operaciones

Para un conjunto de 2000 operaciones de inserción, selección y actualización de la información, los resultados obtenidos son los siguientes:

Test de Rendimiento MySQL vs. SQLServer vs. MongoDB vs. Marklogic	Operación	MYSQL	SQL Server	MongoDB	Marklogic
		Tiempo empleado en milisegundos			
	Insertar	39.659	18.985	396	1.466
	Selección	414.550	78.836	843	37.266
	Actualizar	209.398	72.711	500	264.366

Figura 33: Resultados obtenidos para 5000 operaciones de inserción, selección y actualización

Se presentan las últimas gráficas desarrolladas para este test de rendimiento, en el que se han realizado pruebas de 5.000 operaciones para cada base de datos.

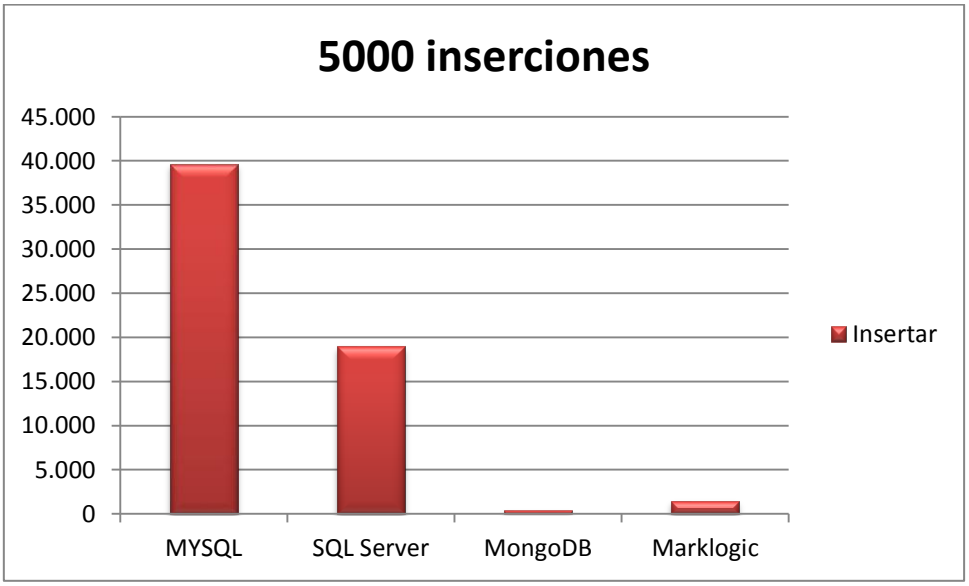

Figura 34: Resultados en ms para 5000 inserciones

La gráfica arroja que los mejores resultados en cuanto a inserción masiva de datos los tienen en primer lugar MongoDB, seguido de Marklogic.

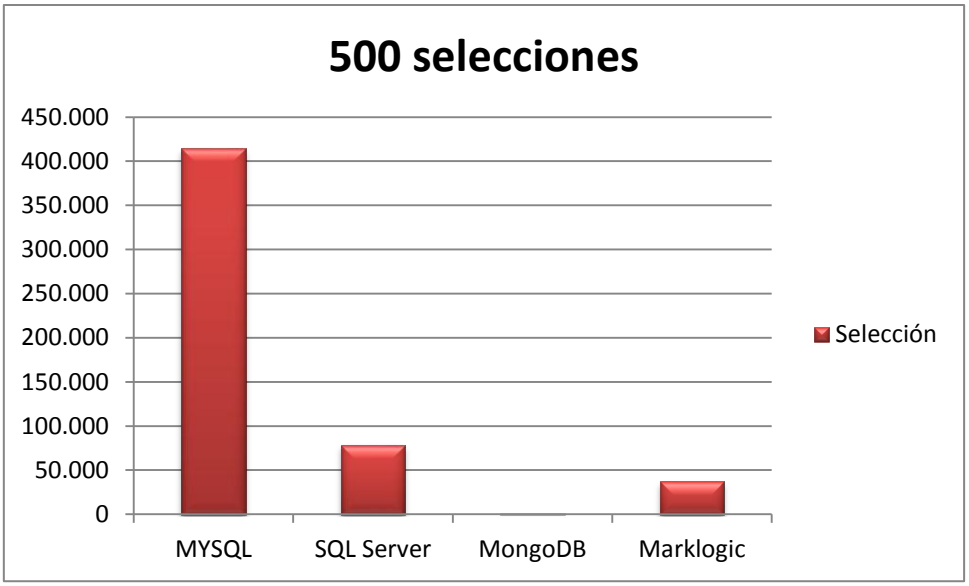

Figura 35: Resultados en ms para 5000 selecciones

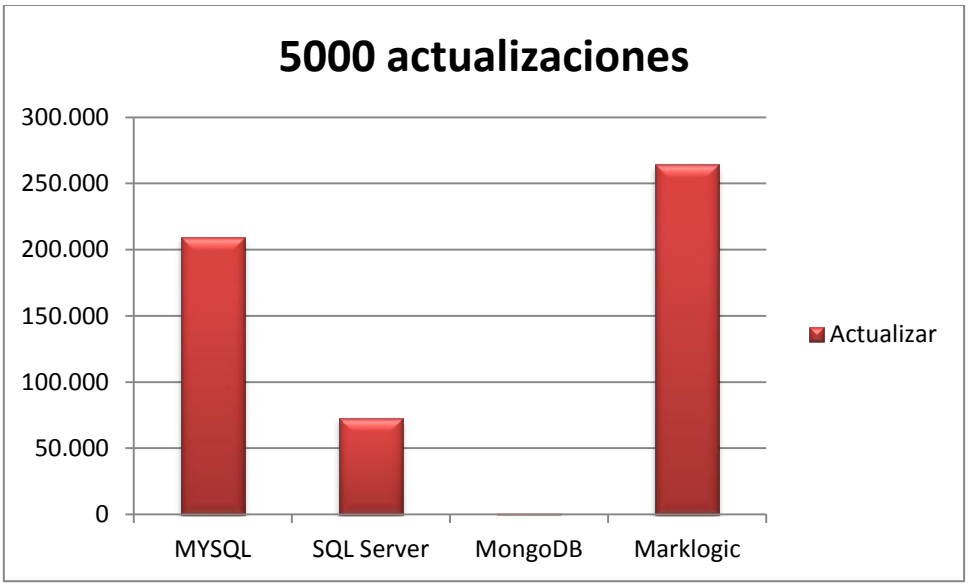

Figura 36: Resultados en ms para 5000 actualizaciones

Para selección y actualización masiva de datos, MongoDB sigue presentando los mejores resultados, mientras que Marklogic muestra buenos datos en selecciones, no así para actualizaciones.

6.4 Conclusiones del Test de Rendimiento

En base a lo visto en el apartado 6.3, los resultados de MongoDB son los más destacados de entre las cuatro bases de datos estudiadas. Esto refuerza la idea de utilizar bases de datos NoSQL como soporte para almacenamiento de la herramienta de e-Assessment que se verá en el siguiente capítulo.

Los resultados de SQL Server han sido muy regulares, con datos buenos, aunque peores que MongoDB. En cuanto a Marklogic cabe resaltar sus buenos datos en selecciones e inserciones de datos, pero unos resultados más pobres en cuanto a actualizaciones de los mismos. En lo que respecta a MySQL, sus resultados en general han sido los que peores marcas ha registrado.

Con todo lo anterior, se reafirma la idea de utilizar MongoDB como base de datos de referencia para este Proyecto Final de Carrera.

Como nota aclaratoria, indicar que no se ha utilizado ningún tipo de optimización, típicas en bases de datos relacionales, y que harían mejorar los resultados obtenidos; sin embargo, se ha querido hacer hincapié en los resultados ante igualdad de condiciones, con el código base de cada solución y sin aditivos que pueden conseguirse a base de invertir dinero en los sistemas.

El siguiente capítulo está dedicado a la herramienta desarrollada para este Proyecto Final de Carrera, una herramienta de e-Assessment que va a permitir evaluar el trabajo realizado por los usuarios.