Modelado, implementación, optimización y generación de estadísticas de desempeño en una BDOO

Alejandro M. Aguirre
Universidad Católica de Santiago del Estero – Dirección Académica Rafaela
aaguirre8@gmail.com
Lorena D'Iorio
Universidad Católica de Santiago del Estero – Dirección Académica Rafaela
lorenadiorio@gmail.com
Marcela A. Vera
Universidad Católica de Santiago del Estero – Dirección Académica Rafaela
marcelaandrea.vera@gmail.com

Resumen

El presente trabajo, muestra los resultados obtenidos en el proyecto de investigación de cátedra realizado en la Sede Rafaela de la Universidad Católica de Santiago del Estero en un lapso de catorce meses iniciados en mayo del 2011, el cual aborda la problemática de la optimización en el diseño de una Base de Datos Orientada a Objetos (BDOO), garantizando bloqueos eficientes, concurrencia y acceso multiusuario.
Se realizó el diseño e implementación de una BDOO del material bibliográfico existente en la UCSE DAR, explotando los beneficios que estas proponen.
Al trabajar con gran cantidad de datos, es fundamental buscar en la Base de Datos tan eficazmente como sea posible. Para esto se probaron dos productos de BDOO lo que nos permitió realizar un análisis comparativo de la respuesta de estas, tiempos, bloqueos, flexibilidad, desempeño, etc.
A partir de las Bases de Datos generada, se realizaron diferentes optimizaciones sobre estas de manera tal que podamos comprobar y comparar el comportamiento ante consultas sobre muchos objetos, sobre objetos complejos, sin indexar e indexados.
Luego, todos los casos de prueba y mediciones de desempeño realizados, nos permitieron generar conclusiones acerca de las particularidades y similitudes en ambos Sistemas de Administración de Bases de Datos.

Palabras clave: Base de Datos, objetos, concurrencia, performance, optimización.

Introducción

Hasta la aparición de las Bases de Datos Orientadas a Objetos (BDOO), las Bases de Datos (BD) tradicionales no estaban diseñadas para almacenar objetos, con lo que al guardar los datos de un programa bajo el enfoque Orientado a Objetos incrementaba significativamente la complejidad del programa, dando lugar a más código y más esfuerzos de programación, así como al problema del Desfase de la Impedancia (diferencia de esquemas).
La tecnología utilizada en BDOO es la consumación de la programación orientada a objetos y la tecnología de BD (modelado de objetos y las BD tradicionales). Muchos de los propósitos de las BDOO son los mismos que los de las bases de datos tradicionales, pero con la ventaja adicional de poder representar modelos de datos más complejos (objetos cuyo valor de algún atributo es otro objeto) en un marco mucho más eficiente, además de permitir la persistencia de los objetos, manteniendo la integridad y las relaciones entre ellos.

En el proyecto de investigación de cátedra, nos planteamos como objetivos específicos:

- Realizar el diseño lógico y físico de una base de datos orientada a objetos, que contenga la información de la biblioteca de la Universidad Católica de Santiago del Estero, Departamento Académico Rafaela.
- Realizar un análisis comparativo de dos implementaciones específicas de Administración de Base de Datos orientadas a Objetos, en cuanto a su desempeño y performance, teniendo en cuenta principalmente factores como: concurrencia, carga superlativa de datos, performance y tiempos de respuesta.

Para el logro de los objetivos planteados, en el marco de las cátedras en donde nos desempeñamos: Base de Datos II y Métodos Ágiles de desarrollo de Software, se trabajó con alumnos de ambas materias que realizaron la recolección de datos, diseño de casos de prueba, instalación del software y generación de documentos de instalación y pruebas de performance.
Esta experiencia además, se utiliza como material y caso de estudio en ambas materias, mostrando a los alumnos la implantación en dos motores de BDOO un problema real, sobre el cual se realizaron pruebas de medición de desempeño que colaboran con el aprendizaje de los conceptos teóricos dados.

Desarrollo del proyecto de investigación

Justificación

En la actualidad, hay pocos trabajos que estudien el desempeño de implementaciones específicas de Bases de Datos orientadas a objetos, en cuanto a la concurrencia, desempeño y performance. Nos propusimos realizar un estudio comparativo, en dos implementaciones Open Source de BDOO, evaluando exhaustivamente su comportamiento, y revisando la flexibilidad, recursos insumidos, bloqueos y demás puntos de interés en ambas implementaciones.

Generar conocimiento e información comparativa, permitirá fehacientemente tomar decisiones más fundamentadas al seleccionar una implementación de BDOO, tanto en proyectos de carrera de los alumnos de la UCSE – DAR, como también permite generar conocimiento del comportamiento de las BDOO ante la concurrencia de múltiples usuarios, desempeño y performance, el cual consideramos que debería ser trasmitido a toda la comunidad en general, y a las empresas en particular que soliciten asesoramiento.
Deseamos brindar a los alumnos y docentes de la UCSE – DAR información comprobada del comportamiento de una BDOO, teniendo en cuenta todas las características específicas, comparando dos implementaciones (DB4O y GemStone) que tiene licencia GNU, para que de esta forma, la planificación de futuros proyectos de Base de Datos que utilicen estas tecnologías puedan realizarse en forma más exacta y detallada.

Diseño lógico del modelo

Para el diseño de la solución OO al problema planteado, se tomó como situación inicial el sistema de base de datos ya existente en la entidad universitaria, el cual está compuesto por una base de datos relacional implementada en tecnología Paradox.
Del análisis de las relaciones involucradas en esa base de datos, los datos cargados, y las entrevistas con el personal universitario acerca de las problemáticas actuales, se desarrolló la lista de casos de uso que la nueva solución de base de datos debería responder.
Utilizando como herramienta de modelado BOUML, se desarrolló el diagrama de clases (Figura 1).

Ambientes de trabajo

Para la implementación del modelo utilizando tecnologías de base de datos orientadas a objeto (OODMBS), se seleccionaron dos motores de diferentes tecnología: DB4Objects y Gemstone.

DB4Objects es un motor de base de datos orientado a objeto que prioriza la facilidad de uso, el alto rendimiento, y el bajo consumo de recursos de hardware.
Tiene una integración transparente con las aplicaciones desarrolladas en tecnologías de objetos, evitando el problema del cruce de paradigma y los mapeos objeto-relacional.
Se trata de un conjunto de librerías que se pueden utilizar directamente desde proyectos en Java o en .Net.
Si bien admite configuraciones distribuidas y cliente/servidor, su fuerte está en escenarios embebidos.
Gemstone es un motor de base de datos orientado a objetos basado en tecnología Smalltalk. Si bien ése es su lenguaje nativo, existen librerías y conectores para utilizarlo desde otros lenguajes de programación como Java.
Éste motor de base de datos puede ser utilizado tanto en arquitecturas cliente/servidor, como en ambientes distribuidos.
Está fuertemente enfocado en el manejo de grandes cúmulos de información, disponiendo de robustos mecanismos de control de concurrencia, de control de transacciones, de escalabilidad y de estabilidad.

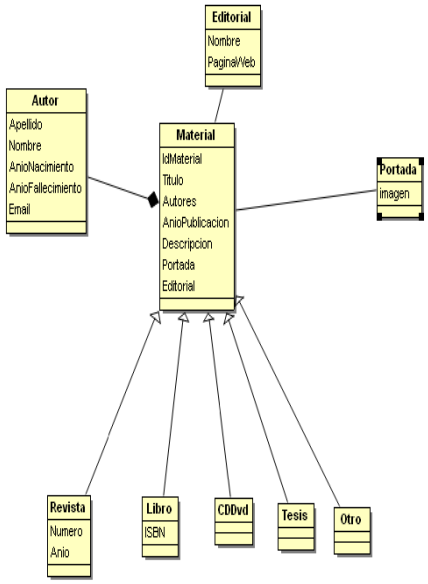

Figura 1: Diagrama inicial de clases de Recursos Bibliográficos

El ambiente de trabajo para DB4Objects se desarrolló sobre un entorno con las siguientes características:

- Windows XP/Vista/7
- Java JDK 7 o Java JRE 7
- Eclipse Indigo 3.7
- DB4Objetcs para Java 8.0
- ObjectManagerEnterprise para Java 8.0

El asistente de instalación de DB4Objects para Windows no deja ningún motor de base de datos residente ejecutándose en la máquina, sino simplemente despliega las librerías para .NET y Java, siendo éstas las encargadas de la gestión de las bases de datos y de la interfase con las aplicaciones.
El entorno de programación utilizado en el proyecto, tanto para la implementación del modelo lógico como para la realización de las pruebas fue Eclipse Indigo.
El ambiente de trabajo para Gemstone se desarrolló sobre un entorno con las siguientes características:

- Windows XP/Vista/7
- Java JDK 7 o Java JRE 7
- Eclipse Indigo 3.7
- Gemstone\S 32 bits 6.6
- GemBuilder for Java 3.0

La versión de Gemstone\S utilizada no cuenta con un asistente de instalación que encapsule el despliegue y la registración de todos los componentes necesarios para el funcionamiento del motor de base de datos, sino que cuenta con un instructivo multiplataforma detallado.
Dicho instructivo incluye la creación de variables de entorno, registración de nombres y de servicios, descompresión y despliegue de ficheros ejecutables, y carga de licencia de uso. Para la realización de éste proyecto, se utilizó una licencia educacional gestionada directamente con el proveedor del producto.
Como resultado del proceso de instalación, quedó implementado un motor de base de datos orientado a objetos ejecutándose como servicio Win32.

Carga de datos y testeo

Para la generación de los casos de prueba, se utilizaron las técnicas de TDD (Test Driven Development o Desarrollo Guiado por Pruebas), lo que permitió pensar los casos antes que el código de forma objetiva, y respondiendo a las necesidades reales planteadas, sin la subjetividad que se encuentra implícita cuando estas se generan con posterioridad a la generación del código.
El método a seguir cuando se utiliza TDD es sencillo. Consiste en elegir uno de los requisitos a implementar, buscar un primer ejemplo sencillo del requisito, crear una prueba unitaria, ejecutar la prueba, implementar el código mínimo para superar la prueba y ejecutar de nuevo la prueba para ver que se supera. Siguiendo este esquema de trabajo se generaron todos los casos de prueba y se realizaron las pruebas y mejoras en ambas Bases de Datos.
Además, al utilizar TDD para la generación de los casos de prueba y testeo, se comprobó de manera empírica las fortalezas de este método ágil. Por otro lado al realizar primero las pruebas se realiza un ejercicio previo de análisis, en profundidad, de los requisitos y de los diversos escenarios. Eliminando la mayor parte de variabilidad y encontrado aquellos aspectos más importantes o no contemplados en los requisitos.
Se utilizaron los mismos test, casos de uso, parámetros y sets de datos tanto para las pruebas sobre DB4Object como para Gemstone.
Para la creación de los sets de pruebas se partió de los datos alojados en el sistema relacional existente, enriquecidos con información adicional recolectada. La carga se realizó a través de archivos de texto maestros, y se utilizaron scripts personalizados para cada motor de base de datos.
Las colecciones de objetos resultantes, persistidas en ambos motores de bases de datos, fueron sometidas a una serie test, en donde se midieron los tiempos de respuesta y se validaron los resultados obtenidos ante diferentes cantidades de objetos persistidos en las colecciones. Adicionalmente, se probaron diferentes configuraciones de índices en distintos tipos de colecciones.
La ejecución de los tests, y el despliegue y llenado de datos en los motores de base de datos, se repitió en varios equipos computacionales de diferentes capacidades y recursos de hardware.

Índices y mecanismos de consulta

Para la implementación de los casos de pruebas en los motores de bases de datos, se utilizaron consultas normalizadas de acuerdo a la documentación y a las buenas prácticas particulares de cada tecnología.
Para los tests sobre Gemstone se utilizó índices tipo Equality. El único tipo de colección con la capacidad de indexación en éste motor de base de datos es el Set.

libros createEqualityIndexOn: 'titulo' withLastElementClass: Libro.

Para una correcta utilización de esos índices, se utilizaron consultas construidas a partir de bloques de selección normalizados. La sintaxis de los bloques de selección varía en algunos aspectos en relación al protocolo de consulta de colecciones estándar en Smalltalk.
Protocolo clásico Smalltalk:
resultado := libros select: [:unLibro | unLibro name = 'Harry Potter'].
Protocolo de bloques de selección en Gemstone:

resultado := libros select: {:unLibro | unLibro.nombre = 'Harry Potter'}.

En DB4Objects, para una correcta utilización de índices hay que tener en cuenta dos aspectos: indicar los índices utilizados en los parámetros de conexión a la base de datos, y la utilización de algunas de las sintaxis normalizadas de consulta (QBE, Native Query, SODA).
En nuestro caso, usamos la sintaxis de consulta nativa de DB4Objects para Java:

```
EmbeddedConfiguration configuration = Db4oEmbedded.newConfiguration();
// Configuración indicando la utilización de índices
```

```
configuration.common().objectClass(Libro.class).objectField("titulo").indexed(useIndexes);
configuration.common().objectClass(Material.class).objectField("titulo").indexed(useIndexes);
configuration.common().objectClass(Autor.class).objectField("apellido").indexed(useIndexes);
...
// Utilización de la configuración preparada en el momento de la conexión
ObjectContainer db = Db4oEmbedded.openFile(configuration, dbFileName);
...
// Consulta utilizando Native Query
List <Libro> results = db.query(new Predicate<Libro>() {
        public boolean match(Libro result) {

        return result.mostrarTitulo().equals("Harry Potter");
   }
}
...
db.close()
```

Resultados obtenidos

Los primeros tests se corrieron sobre colecciones de objetos persistentes sin índices. Para Gemstone se utilizaron dos tipos de colecciones diferentes: OrderedCollecion, y Sets. Los tiempos promedios de acceso, ante colecciones de diferentes tamaños fueron los siguientes:

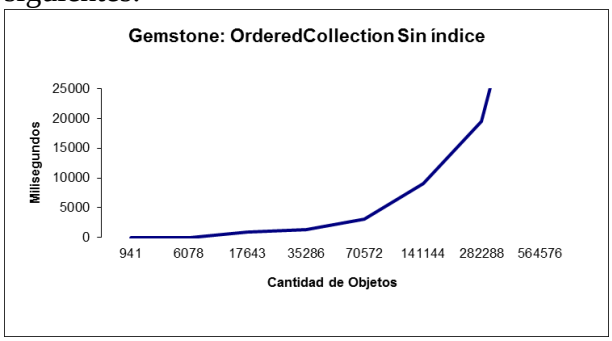

Figura 2: Gemstone sin índices, caso 1

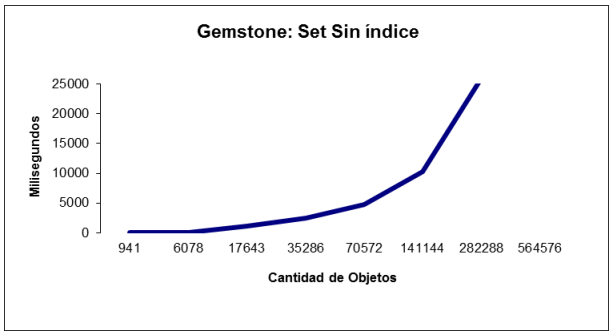

Figura 3: Gemstone sin índices, caso 2

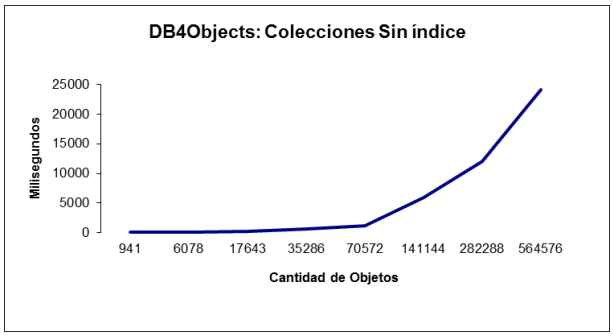

Figura 4: DB4Objects sin índice

Los gráficos muestran resultados similares en todos los casos: un aumento exponencial de los tiempos de respuesta a medida que aumenta el número de objetos en las colecciones.

La curva de DB4Objects es levemente menos pronunciada que en los casos de Gemstone. Sin embargo, en el primero se han detectado casos de soft-crash al trabajar con alta carga en equipos con bajos recursos en materia de memoria principal. No se han detectado problemas de inestabilidad en Gemstone.

La utilización de índices disminuye drásticamente los tiempos promedios de respuesta. En Gemstone únicamente se evaluó la colección tipo “Set”, ya que es la única que tiene la capacidad de indexado.

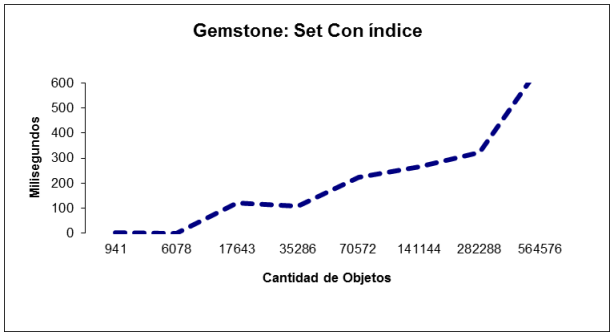

Figura 5: Gemstone con índice

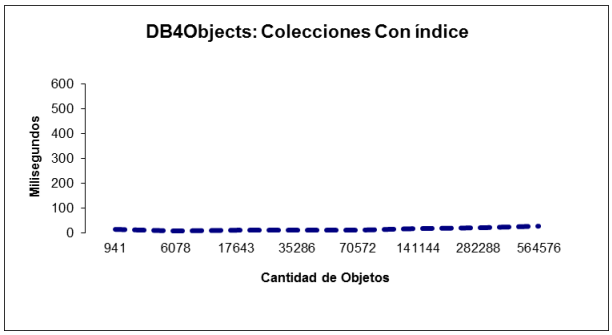

Figura 6: DB4Objects con índice

Los gráficos muestran un mejor desempeño en DB4Objects. Estas diferencias son atribuibles a que en Gemstone se requiere manejar estructuras de datos mucho más complejas para el manejo de índices, que le garantizan mayor robustez en materia de concurrencia.

La variabilidad de los resultados de los test, comparando equipos computacionales de diferentes capacidades de HW, indica una marcada heterogeneidad en las muestras tomadas en Gemstone, mientras que en DB4Objects el comportamiento fue mucho más estable.

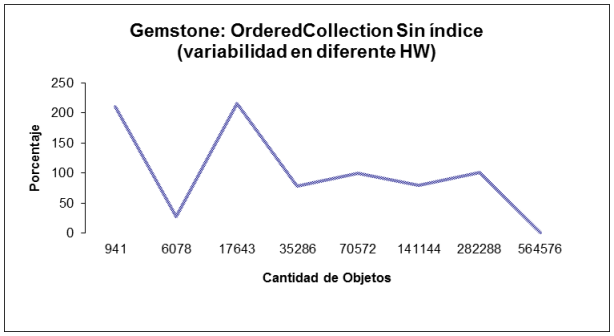

Figura 7: Gemstone en diferentes equipos, sin índice, caso 1.

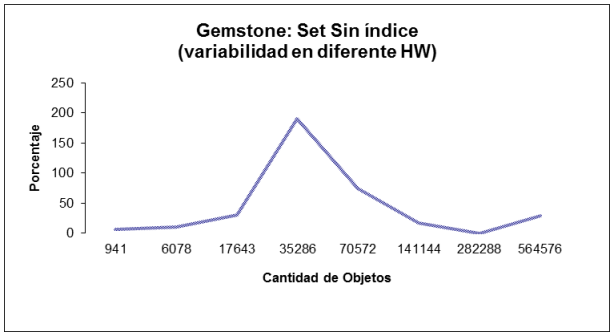

Figura 8: Gemstone en diferentes equipos, sin índice, caso 2.

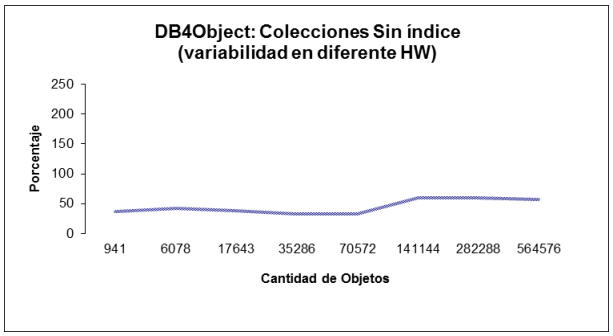

Figura 9: DB4Objects en diferentes equipos, sin índice.

En ambos gestores de Bases de Datos, la variabilidad utilizando equipos con recursos diferentes disminuye notoriamente ante la utilización de índices. Esto queda claramente indicado en los gráficos que se muestran a continuación.

Ésta tendencia uniforme en DB4Objects y aleatoria en Gemstone se manifestó también entre las diferentes corridas ante el mismo set de datos en un mismo equipo.

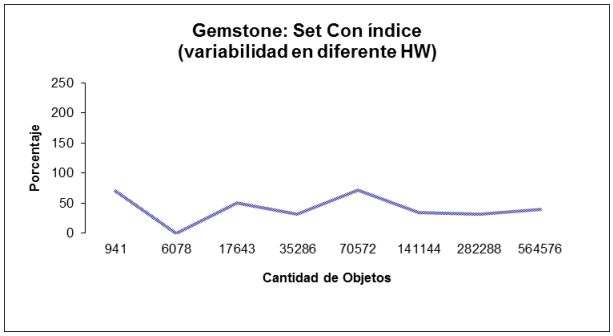

Figura 10: Gemstone en diferentes equipos, colección con índices.

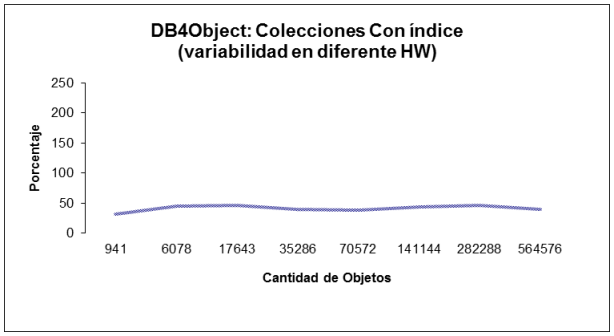

Figura 11: DB4Objects en diferentes equipos, con índices.

Este efecto responde a los diferentes mecanismos de cacheo que disponen ambos motores de base de datos. Las bases de datos DB4Objects al ser generalmente abiertas en forma exclusiva por una aplicación, no requiere de complejos algoritmos en cuando al mantenimiento de la cache de objetos. Gemstone por el contrario, necesita mecanismos eficientes y seguros que permitan compartir objetos en cache entre diferentes sesiones o aplicaciones conectadas en forma concurrente.

Por último, se analizó el efecto de tener colecciones homogéneas (tamaños similares) y colecciones heterogéneas (tamaños disímiles) en una misma base de datos.
En las pruebas sobre colecciones sin índices, el desempeño de DB4Objects se vio muy disminuido, tanto para las consultas sobre colecciones chicas como para la de colecciones grandes. Gemstone no mostró el mismo comportamiento, ya que si bien las consultas sobre colecciones grandes sin índices tuvieron altos tiempos de respuesta, las consultas sobre colecciones chicas en la misma base de datos se mantuvieron dentro de los tiempos normales.

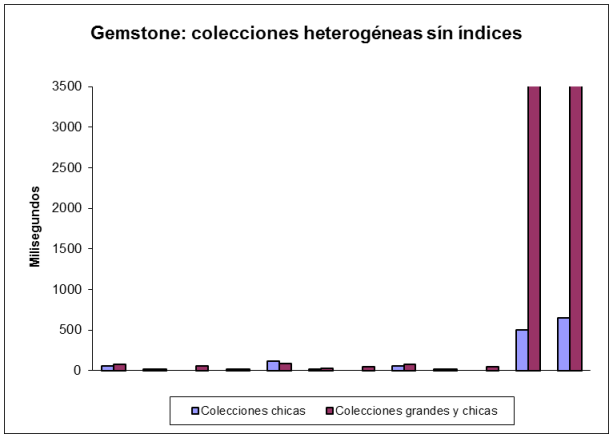

Figura 12: Gemstone con colecciones heterogéneas sin índice.

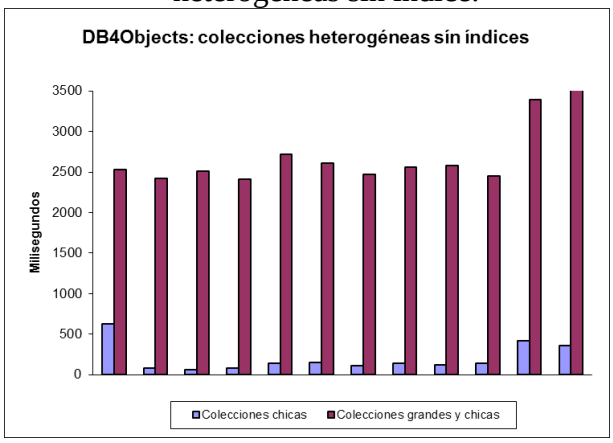

Figura 13: DB4Objects con colecciones heterogéneas sin índices.

Esa situación se normaliza al realizar las mismas consultas sobre iguales colecciones, pero indexadas.

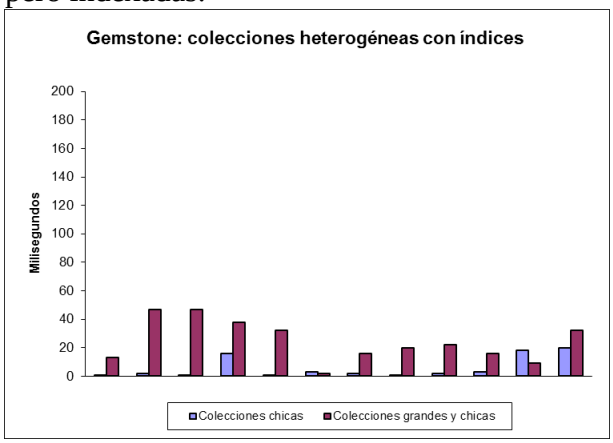

Figura 14: Gemstone con colecciones heterogéneas con índices.

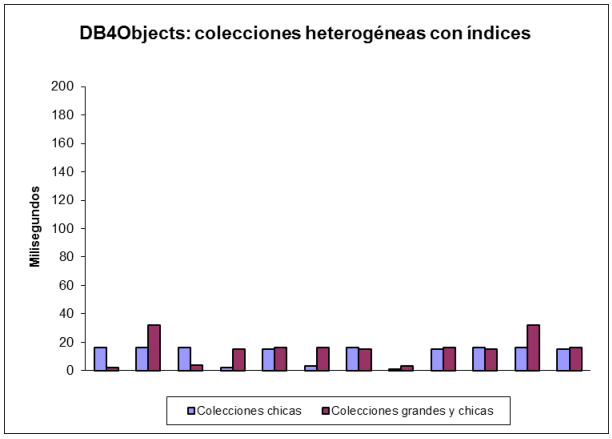

Figura 15: DB4Objects con colecciones heterogéneas con índices.

Conclusiones

Las Bases de Datos Relacionales no están diseñadas para almacenar Objetos, con lo que al guardar los datos de un programa OO incrementa significativamente la complejidad del programa, dando lugar a más código y más esfuerzos de programación, así como al problema del Desfase de la Impedancia, definido esto como la diferencia de esquemas entre la característica del repositorio y los elementos a almacenar.
Por ello, las Bases de Datos Orientadas a Objetos (BDOO) se propusieron con la idea de satisfacer las necesidades de estas aplicaciones más complejas aprovechando las ventajas que ofrece la Programación Orientada a Objetos (POO) como una forma de resolver problemas utilizando modelos que se han organizado en base a conceptos del mundo real.
Las bases de datos tradicionales almacenan sólo datos, mientras que las BDOO almacenan objetos, con una estructura arbitraria y un comportamiento.
Las BDOO aparecen a finales de los 80 y están estructuradas para simplificar la programación Orientada a Objetos, permiten almacenar los objetos directamente en la Base de Dato (BD), utilizando las mismas estructuras y relaciones que los lenguajes de Programación Orientada a Objetos, destacando que surgen de la combinación de éstos con las Bases de Datos tradicionales.
Dentro de las principales características de las BDOO podemos nombrar:

- Permiten trabajar de una manera transparente y eficiente en un entorno de programación basado en objetos, soportan todos los conceptos de la Orientación a Objetos.
- Gestión de datos complejos como por ejemplo los datos multimedia (imagen, video, entre otros)
- Permiten la persistencia transparente de los objetos.
- Presentan en muchos casos una arquitectura distribuida.
- Procesamiento transaccional que soporta la concurrencia.

Adicionalmente, en general soportan las siguientes características (aunque depende de cada Sistemas de Gestión de Base de Datos Orientados a Objetos (SMBDOO)) la Integridad de datos, versionamiento de objetos, indexación, seguridad, y tolerancia a fallos, entre otras.
En cuanto a la implementación, los OODBMS seleccionados han tenido comportamientos similares en algunos aspectos, y dispares en otros.
Las pruebas sobre ambos DMBS coincidieron en el pobre desempeño y la falta de escalabilidad de la solución al trabajar con colecciones sin índices.
Es fundamental en todo diseño de base de datos, trabajar en la optimización del modelo haciendo uso de los diferentes tipos y características de índices que proveen los motores de bases de datos.
Por el contrario, los resultados obtenidos sobre ambos DBMSs difirieron en cuestiones relacionadas con el desempeño, la estabilidad y la escalabilidad.
DB4Objects se mostró más rápido y más predecible en cuanto a sus tiempos de respuesta, pero tuvo problemas para funcionar en condiciones desfavorables, como equipos

de bajos recursos sometidos a altas cargas sobre colecciones no indexadas.
Gemstone por su parte se mostró mucho más estable bajo todas las condiciones de la muestra, lo que indica que es un motor de base de datos más escalable.
Los resultados brindan una clara visión acerca de los escenarios de aplicabilidad de cada motor de base de datos. DB4Objects es más apto para aplicaciones chicas, sin concurrencia, en donde prime la velocidad, y la disminución del tiempo y de los costos de desarrollo, de instalación y de mantenimiento.
Gemstone tiene su campo de acción en sistemas mucho más grandes y corporativos, ya que privilegia la estabilidad y la escalabilidad, brindando mecanismos más robustos de control de concurrencia, seguridad y caching de objetos, a costa de una pequeña penalidad en el desempeño, y mayores complejidades de instalación y operación.
Debemos concluir que las Bases de Datos Orientadas a Objetos están aquí para quedarse, y tienen la capacidad de atender las necesidades de datos de aplicaciones donde la tecnología relacional tiene problemas de desempeño, de escalabilidad y/o flexibilidad de mantenimiento. Es por esto que consideramos fundamental estudiar las alternativas disponibles para conocer sus ventajas, performance y campos de uso de cada una de ellas.
Además, al utilizar TDD para la generación de los casos de prueba y testeo, se comprobó de manera empírica las fortalezas de este método ágil. Por otro lado al realizar primero las pruebas se realiza un ejercicio previo de análisis, en profundidad, de los requisitos y de los diversos escenarios. Eliminando la mayor parte de variabilidad y encontrado aquellos aspectos más importantes o no contemplados en los requisitos.
El hecho de que además solo se implemente el código necesario para resolver un caso de prueba concreto, pasar la prueba, hace que el código creado sea el mínimo necesario, quitando redundancia y los típicos bloques de código de "por si acaso" que habitualmente se convierten en código.
Finalmente, podemos concluir que este trabajo pretende ser una primer etapa de evaluación de dos implementaciones de BDOO, que debe ampliarse realizando estudios más profundos de comportamiento ante concurrencia, alta carga de trabajo, consultas de objetos complejos, etc.
A modo de resumen citamos dos autores que representan muy claramente lo que significa la Orientación a Objetos:
"Las Técnicas Orientadas a Objetos modifican el punto de vista de los analistas de sistemas de información acerca del mundo, en vez de pensar en los procesos y su descomposición, piensan en Objetos y su comportamiento"(Martin, y otros, 1994, 3).
"El enfoque Orientado a objetos es ambicioso: abarca todo el ciclo de vida de desarrollo software. Cuando se examinan soluciones orientadas a objetos se debe comprobar que el método y el lenguaje, así como las herramientas de apoyo, sean aplicables al análisis y al diseño al igual que la implementación y el mantenimiento" (Meyer, 1998, 9)

Bibliografía

- (Date, 2001). *Introducción a los sistemas de Bases de Datos*. C. J. Date, Addison Wesley – 2001
- (Silberschatz, Korth, Sudarchan, 2002). *Fundamentos de Bases de Datos*. A. Silberschatz, H.Korth, S. Sudarchan, Mc Graw Hill – 2002
- Análisis y diseño orientado a Objetos – Martin, James, Odell, James J. – Prentice Hall – 1994
- (Elmasri, Ramez, Navathe, Shamkant, 2002). *Fundamentos de sistemas de bases de datos*. Elmasri, Ramez, Navathe, Shamkant B., Prentice Hall – 2002

- (Hughes, 1991). *Object – Oriented Databases*. J.G. Hughes, Prentice Hall – 1991

- (Cattell, Roderic, Galton, 2000). *The object data standard: ODMG 3.0*. Cattell, Roderic, Galton y Otros , Academic Press – 2000

- (Bertino, Martino, 1995). *Sistemas de Base de datos Orientado a Objetos: Conceptos y Arquitecturas*. Elisa Bertino, Lorenzo Martino, Addison Wesley – 1995

- (Meyer, 1998). *Construcción de Software Orientada a Objetos.* Meyer, Bertrand. (Segunda edición), España : Prentice Hall - 1998

- Gemstone: http://www.gemstone.com/

- DB4Objects: http://www.db4o.com/