PERSISTENCIA DE DATOS CON ActiveJDBC ORM

José Jesús Sánchez Farías
Tecnológico Nacional de México/Instituto Tecnológico de Celaya
jesus.sanchez@itcelaya.edu.mx

Rubén Torres Frías
Tecnológico Nacional de México/Instituto Tecnológico de Celaya
ruben.torres@itcelaya.edu.mx

Luis Alberto López González
Tecnológico Nacional de México/Instituto Tecnológico de Celaya
luislao@itcelaya.edu.mx

Juan Ignacio Cerca Vázquez
Tecnológico Nacional de México/Instituto Tecnológico de Celaya
nacho@itcelaya.edu.mx

Resumen

Se presenta un estudio realizado a ActiveJDBC un *framework* de persistencia de datos utilizado para el mapeo objeto-relacional (ORM) y aplicado en el lenguaje de programación Java. Se comienza con los fundamentos teóricos acerca de los ORM`s, se mencionan los principales ORM`s utilizados en Java, después se centra en el análisis y evaluación de ActiveJDBC a través de una aplicación de escritorio desarrollada para tal propósito. Esta aplicación se enlaza con una base de datos previamente creada y con miles de registros existentes para realizar pruebas de rendimiento del *framework*. Finalmente se muestran los resultados obtenidos como consecuencia de la aplicación de las pruebas y se comparan contra resultados obtenidos aplicando los métodos tradicionales de persistencia de datos como es JDBC.

Palabras Clave: ActiveJDBC, hibernar, JDBC, ORM, persistencia.

Abstract

An ActiveJDBC study is presented, it`s a data persistence framework for object-relational mapping (ORM) and implemented in the Java programming language. It begins with the theoretical foundations about ORM`s, the main ORM`s used in Java are mentioned, then it focuses on the analysis and evaluation of ActiveJDBC through a desktop application developed for this purpose. This application is linked to a database previously created with thousands of existing records for performance testing. Finally the results obtained as a result of the application of the tests are shown and compared against results obtained by applying traditional methods of data persistence as JDBC.

Keywords: *ActiveJDBC, hibernate, JDBC, ORM, persistence.*

1. Introducción

Todo programador que desarrolla software de aplicación tal como es el software empresarial, educativo, videojuegos, medicina, telecomunicaciones, automatización industrial, diseño asistido, etc., tarde o temprano se enfrenta con la necesidad de guardar la información que actúa como entrada o salida de los sistemas que desarrolla, busca que esta información persista incluso después de que la aplicación termina su ejecución. Los *frameworks* de mapeo objeto-relacional u ORM por sus siglas en inglés (*Object-Relational Mapping*) juegan un papel muy importante para el mejoramiento del proceso de persistencia de datos entre un lenguaje de programación y una base de datos. Los programadores profesionales siempre pretenden realizar bien su trabajo y además están en constante búsqueda de nuevas herramientas, metodologías, estándares que permitan mejorarlo. Este artículo describe brevemente los ORM's más populares en el mercado, específicamente se evalúa ActiveJDBC una alternativa que se pueden utilizar en el lenguaje de programación Java, presentando una fundamentación teórica, así como las ventajas y desventajas de su uso contra los métodos tradicionales de persistencia de datos.

Durante el desarrollo de un sistema de información buscamos herramientas que permitan agilizar el proceso, nos encontramos con *frameworks* que nos permiten

dar estructura y orden a nuestros proyectos, la mayoría de ellos *frameworks* de desarrollo. Es bueno también voltear hacia otro tipo de *frameworks*, como son los que nos permiten manejar la persistencia de datos con ORM:

- Object Relational Mapping: Traducido como Mapeo Objeto-Relacional, nos permite acceder de una manera efectiva a las bases de datos desde un contexto orientado a objetos, creando una interfaz de comunicación formada por objetos que permiten acceder a la información. Los ORM`s nos permiten realizar cualquier tipo de operación sobre una base de datos como son: creación de una base de datos, creación de esquemas, lecturas y manipulación de datos. Las principales ventajas que se obtienen al utilizar estos son la reutilización de código, abstracción de datos, costos de mantenimiento y reducción errores.
- JDBC API y ActiveJDBC: Son APIs (*Application Program Interface*) para Java que permiten acceder y manipular la información de una base de datos relacional. Esta interfaz permiten trabajar con muchos DBMS como son: MySQL, PostgreSQL, Oracle, SQL Server, etc., para ello hacen uso de un driver exclusivo para cada manejador.
- ActiveJDBC es un ORM para el desarrollo Ágil de aplicaciones que acceden a bases de datos.
- JavaFX: Es un framework de medios/gráficos utilizado para el desarrollo de interfaces gráficas de usuario (GUI) en aplicaciones Java. Uno de sus propósitos es remplazar a Swing y convertirse en la librería estándar GUI para Java SE. Algunas de sus características son:
  - ✓ Mucho más poderoso que Swing y AWT.
  - ✓ Se utiliza para crear tanto aplicaciones de escritorio como aplicaciones Web.
  - ✓ Es liviano y utiliza aceleración de hardware.
  - ✓ Hace uso de FXML, un lenguaje de marcado basado en XML para la definición de UI's.
  - ✓ Inicialmente creado por la empresa Sun MicroSystems, actualmente mantenido por Oracle Corporation.

- Pruebas de Rendimiento: Este tipo de pruebas como su nombre lo dice, ponen a prueba el rendimiento de un software en tiempo de ejecución, en la cual están dadas las condiciones necesarias para la misma. Es muy común tener sistemas de información que funcionan a la perfección, realizan las tareas solicitadas por parte de los usuarios, pero al momento de llevarlos a un esquema de producción donde es accedido por cientos o tal vez miles de usuarios, estos colapsan. De aquí la importancia de hacer pruebas necesarias sobre el software, librerías o, como en este caso, *frameworks* que deseamos integrar en un sistema de información. Así también se pueden utilizar las pruebas de rendimiento para mejorar tareas individuales dentro de un sistema, esto es, si existe un módulo en el cual requiere acceso a miles o millones de registros y que constantemente sigue creciendo, es importante realizar las pruebas necesarias para obtener el máximo rendimiento y así evitar el disgusto por parte de los usuarios o incluso el fracaso del sistema completo.

2. Método

Cuando estamos desarrollando un software, existen una serie de errores comunes que se comenten y que en un determinado momento puede afectar el funcionamiento o el rendimiento de la aplicación. Entre esos posibles errores se encuentra recuperar una gran cantidad de registros para posteriormente mostrarlos a los usuarios, tal vez no es necesario mostrar todos los registros sino más bien filtrar bajo ciertos criterios; otro error común es abrir muchas conexiones a la base de datos e incluso dejarlas abiertas, provocando saturación y lentificar el sistema. Es por esto que resalta la importancia de las pruebas de software, además de la evaluación de las tecnologías antes de utilizarlas, ya que no queremos darnos cuentas de sus fallas una vez que hemos terminado el proyecto de software.

Pasos que se realizaron para llevar a cabo el uso, medición y evaluación de JDBC y ActiveJDBC:

- Primeramente se tomaron en cuenta dos factores para la medición del rendimiento de ActiveJDBC contra JDBC el método tradicional de persistencia de datos en Java. Estos son:
  - ✓ Tiempo de ejecución de ciertas operaciones sobre la base de datos. Se realizaron operaciones tanto de selección, inserción y eliminación de registros desde una aplicación de escritorio contra una base de datos MySQL.
  - ✓ Líneas de código utilizadas para realizar operaciones comunes a una base de datos. Al desarrollar un sistema de información es muy importante la cantidad de líneas de código escritas, esto se reflejará en los tiempos de desarrollo así como el rendimiento del sistema. Es por esto que este parámetro juega un papel muy importante al comparar JDBC contra ActiveJDBC, nos reflejará con cuál de ellos escribimos menos y más líneas de código para realizar operaciones básicas sobre una base de datos.
- Descripción de la aplicación y bases de datos utilizada, para lo cual se desarrolló una aplicación de escritorio en el lenguaje de programación Java, utilizando además el framework de interfaces gráficas de usuario JavaFX con FXML. Esta aplicación consiste de una interfaz que permiten listar todos los registros de empleados de una empresa, además de realizar otras operaciones como inserción y eliminación de los mismos.

  La base de datos "Employees" fue utilizada para el desarrollo de la aplicación. Esta es una base de datos que se ofrece desde el portal oficial de MySQL y creada para fines educativos y pruebas del manejador. Consiste de seis tablas etiquetadas como: departments, dept_emp, dept_manager, employees, salaries y titles. La tabla "employees" fue utilizada para realizar las pruebas de medición, esta contiene originalmente 300,024 registros, considerados suficientes para las pruebas. Sus campos son: emp_no, birth_date, first_name, last_name, gender, hire_date, traducidos como número de empleado, fecha de nacimiento, nombres, apellidos, género y fecha de contratación.

- Definición y programación de pruebas a realizar con JDBC y ActiveJDBC. El equipo utilizado para realizar las pruebas tiene las siguientes características: Computadora MacBook Pro con Sistema Operativo MacOS Sierra Versión 10.12, Procesador 2.5 GHz Intel Core I5, Disco Duro de 500 GB y Memoria RAM de 16 GB 1600 MHz DDR3. Las pruebas fueron:
  - ✓ Selección de 300,024 registros. En esta prueba la aplicación desarrollada permite presionar un botón "Cargar", el cual cargará la cantidad indicada de registros en un componente ListView de JavaFX. Una vez terminado el proceso, se muestra en pantalla, debajo del renglón de botones, una leyenda con el tiempo que tardó en realizar la operación, medido en nano-segundos y convertido a segundos.
  - ✓ Inserción de 10,000 registros. En esta prueba la aplicación desarrollada permite presionar un botón "Insertar", el cual inserta la cantidad indicada de registros en la tabla employees, una vez terminada la operación, consulta y muestra los registros en el componente ListView.
  - ✓ Eliminación de 310,024 registros. En esta prueba la aplicación desarrollada permite presionar un botón "Eliminar", el cual elimina la cantidad indicada de registros de la tabla employees, una vez terminada la operación, actualiza el componente ListView para no mostrar más registros.
- Aplicación de pruebas. Para el desarrollo de esta sección, se tomaron capturas de pantalla de la aplicación en ejecución, así como del código utilizado para realizar cada prueba.
  - ✓ Selección de 300,024 registros. Se ejecutaron varias pruebas de selección de registros, esto para ver la variación de los tiempos. En las figuras 1 y 2 se muestra la aplicación después de haber cargado 300,024 registros, mostrando los tiempos obtenidos tanto para JDBC como ActiveJDBC. Ambas figuras muestran los mejores tiempos obtenidos. Así también, en la figura 3, se muestra el código utilizado para la selección y carga de datos.

Figura 1 Carga de registros por JDBC.

Figura 2 Carga de registros por ActiveJDBC.

	Código utilizado para la carga de datos

JDBC	`ObservableList<Employee> listEmployees = FXCollections.observableArrayList();`
	
	`try {`
	`  String query = "SELECT * FROM employees";`
	`  Statement st = Conexion.createStatement();`
	`  ResultSet rs;`
	`  rs = st.executeQuery(query);`
	`  while(rs.next()) {`
	`  listEmployees.add(new Employee(`
	`  rs.getInt("emp_no"),`
	`  rs.getDate("birth_date"),`
	`  rs.getString("first_name"),`
	`  rs.getString("last_name"),`
	`  rs.getString("gender"),`
	`  rs.getDate("hire_date")`
	`  ));`
	`  }`
	`} catch (SQLException ex) {`
	`  System.out.println("Error al recuperar información:" + ex.getMessage());`
	`}`

ActiveJDBC	`List<Employees> departments = Employees.findAll();`
	`ObservableList<Employees> obDepartments = FXCollections.observableArrayList(departments);`
	`listEmployees.setItems(obDepartments);`

Figura 3 Código para la carga de datos.

- ✓ Inserción de 10,000 registros. Para esta prueba se registraron la cantidad de registros indicada en repetidas veces para ver la variación de los tiempos. En las figuras 4 y 5 se muestra la aplicación después de haber insertado 10,000 registros, mostrando los tiempos obtenidos tanto para JDBC como ActiveJDBC. Ambas figuras muestran los mejores tiempos obtenidos. Así también, en la figura 6, se muestra el código utilizado para la inserción de datos.

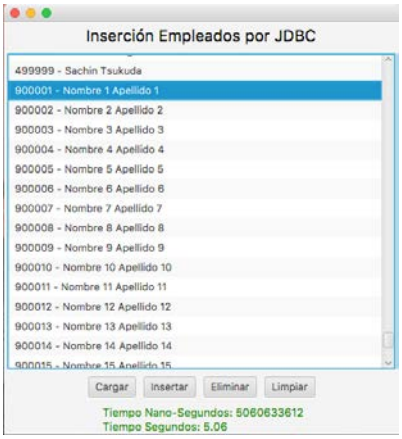

Figura 4 Inserción de registros por JDBC.

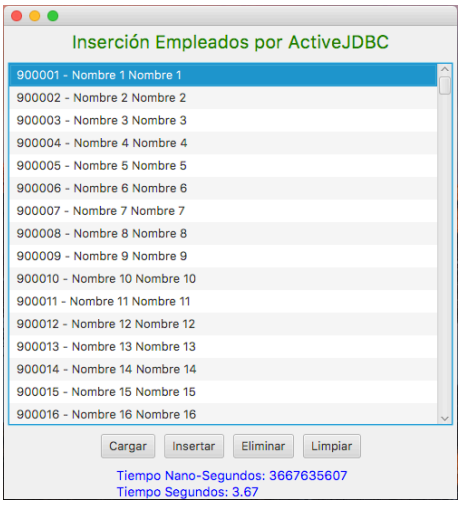

Figura 5 Inserción de registros por ActiveJDBC.

- ✓ Eliminación de 310,024 registros. Para esta prueba se eliminaron la cantidad de registros indicados en repetidas veces para ver la variación de los tiempos. En las figuras 7 y 8 se muestra la aplicación

después de haber eliminado 310,024 registros, mostrando los tiempos obtenidos tanto para JDBC como ActiveJDBC. Ambas figuras muestran los mejores tiempos obtenidos. Así también, en la figura 9, se muestra el código utilizado para la eliminación de datos.

	Código utilizado para la inserción de datos

JDBC	`for(int i = 1; i<=10000; i++) { conn.insertEmployee(new Employee(900000 + i, new Date(2016, 10, 5), "Nombre " + i, "Apellido " + i, "M", new Date(2016, 10, 4))); }`
	
	`public Boolean insertEmployee(Employee empleado) { try { String query = "insert into employees (emp_no, birth_date, first_name, last_name, gender, hire_date) values (?, ?, ?, ?, ?, ?)"; PreparedStatement st = Conexion.prepareStatement(query); st.setInt(1, empleado.getEmp_no()); st.setDate(2, empleado.getBirth_date()); st.setString(3, empleado.getFirst_name()); st.setString(4, empleado.getLast_name()); st.setString(5, empleado.getGender()); st.setDate(6, empleado.getHire_date()); st.execute(); return true; } catch (SQLException e) { System.out.println(e.getMessage()); } return false; }`

ActiveJDBC	`for(int i = 1; i<=10000; i++) { Employees emp = new Employees(); emp.set("emp_no", 900000 + i); emp.set("birth_date", new Date(2016, 10, 5)); emp.set("first_name", "Nombre " + i); emp.set("last_name", "Apellido " + i); emp.set("gender", "M"); emp.set("hire_date", new Date(2016, 10, 4)); emp.saveIt(); }`

Figura 6 Código utilizado para la inserción de datos.

Figura 7 Eliminación de registros por JDBC.

Figura 8 Eliminación de registros por ActiveJDBC.

	Código utilizado para la eliminación de datos
JDBC	`try { String query = "delete from employees"; PreparedStatement st = Conexion.prepareStatement(query); st.execute(); return true; } catch (SQLException e) { System.out.println(e.getMessage()); }`
ActiveJDBC	`Employees.deleteAll();`

Figura 9 Código para la eliminación de registros.

3. Resultados

Se concentró la información obtenida como consecuencia de la ejecución de pruebas aplicadas. En la tabla 1 se muestran los resultados obtenidos tanto por JDBC como ActiveJDBC. Ahí se indican las pruebas realizadas, la cantidad de registros utilizados para cada prueba, los mejores y peores tiempos obtenidos medidos en nano-segundos y segundos.

La comparación de resultados se muestra en la figura 10, donde hay una gráfica en la cual se reflejan y comparan los valores de los tiempos obtenidos. En las dos gráficas superiores se muestran los mejores tiempos obtenidos tanto para JDBC como ActiveJDBC en las tres operaciones de carga, inserción y eliminación. En las

dos gráficas inferiores se muestra la comparación de los mejores tiempos entre JDBC y ActiveJDBC en las tres operaciones de carga, inserción y eliminación.

Tabla 1 Resultados de pruebas por JDBC y ActiveJDBC.

Resultados de pruebas por JDBC

Prueba	Registros	Mejor Tiempo		Peor Tiempo	
		Nanosegundos	Segundos	Nanosegundos	Segundos
Carga	300024	628255612	.63	1137803876	1.14
Inserción	10000	5060633612	5.06	5165179657	5.17
Eliminación	310024	1852930384	1.85	1937150815	1.94

Resultados de pruebas por ActiveJDBC

Prueba	Registros	Mejor Tiempo		Peor Tiempo	
		Nanosegundos	Segundos	Nanosegundos	Segundos
Carga	300024	943281044	.94	2259384949	2.26
Inserción	10000	3667635607	3.67	4332518454	4.33
Eliminación	310024	1818698165	1.82	1903945909	1.9

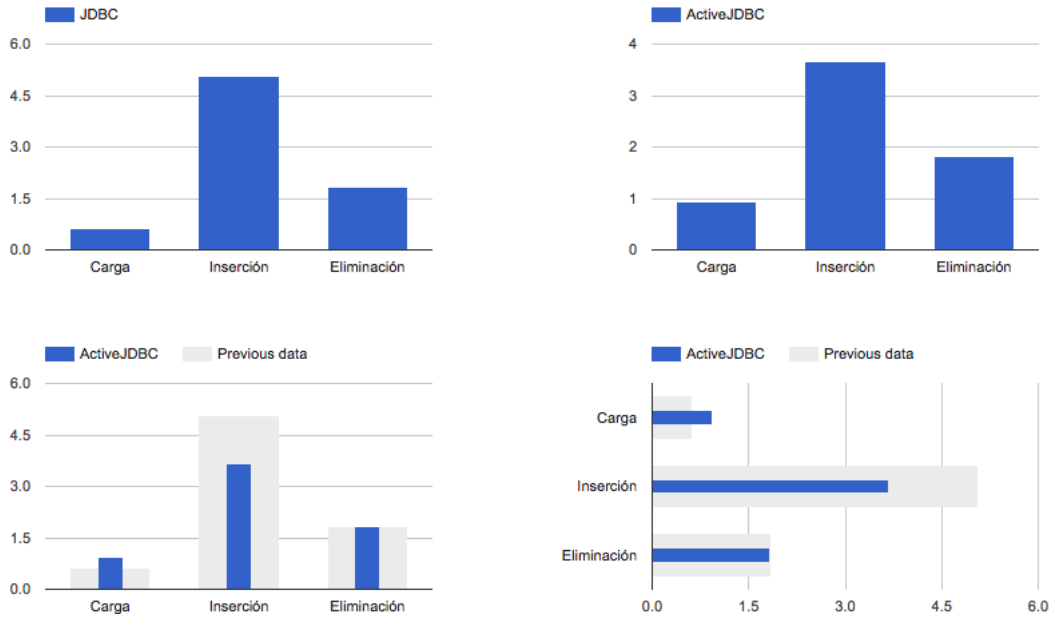

Figura 10 Comparación entre JDBC y ActiveJDBC.

Finalmente, en la tabla 2 se muestran los resultados de ambas tecnologías y coloreado en verde el ganador en cada una de las pruebas. Realmente no existe una diferencia muy significativa en cada una de ellas, la más marcada es en la prueba de inserción donde hay una diferencia de 1.39 segundos.

Tabla 2 Comparativa entre JDBC y ActiveJDBC.

Prueba	**JDBC**		**ActiveJDBC**	
	Nanosegundos	**Segundos**	**Nanosegundos**	**Segundos**
Carga	628255612	.63	943281044	.94
Inserción	5060633612	5.06	3667635607	3.67
Eliminación	1852930384	1.85	1818698165	1.82

En cuanto a la cantidad de líneas de utilizadas para la codificación de las distintas pruebas de carga, inserción y eliminación, se nota una gran diferencia, se requiere en promedio un 50% menos código para programar en ActiveJDBC que con JDBC.

4. Discusión

Desde un punto de vista personal, existen más beneficios y ventajas al utilizar un ORM para el acceso y manipulación de información en una base de datos que hacerlo de la manera tradicional con JDBC, claro está que se sacrificará un poco el rendimiento de la aplicación al realizar ciertas operaciones. En cuanto a los resultados obtenidos se deduce un empate técnico entre ambas tecnologías, ya que no existen diferencias significativas que declarar a un ganador.

ActiveJDBC es una buena opción para el desarrollo de aplicaciones pequeñas y medianas, en aplicaciones grandes y alto rendimiento podrían verse mayores diferencias en el rendimiento de la misma. Además, ActiveJDBC es una buena opción para comenzar en el mundo de los ORM`s, aprender y aplicarlo en proyectos estudiantiles, incluso empresariales, para posteriormente migrar y adentrarse a otros ORM`s más profesionales y con mucho camino recorrido como son Hibernate, JPA o myBatis.

5. Bibliografía y Referencias

[1] Google Developers. (2016, 17 de septiembre). *Diff Charts*. De Google Corporation. https://goo.gl/3ISAM1.

[2] JavaLite. (2016). *ActiveJDBC Fast ORM for agile development*. De JavaLite http://javalite.io/activejdbc.

[3] Topley, K. (2011). *JavaFX Developer`s Guide*. Boston, MA.: Addison-Wesley.

[4] Oracle. (2015a). *JavaFX*. De Oracle Corporation. [Sitio web]. Recuperado de https://goo.gl/VNkGk8

[5] Oracle. (2015b). *JDBC(TM) Database Access*. De Oracle Corporation. [Sitio web]. https://goo.gl/WobpX1.

[6] Venkatasubramaniam Iyer, Elizabeth Hanes Perry, Brian Wright, Thomas Pfaeffle. (2010). *Oracle Database JDBC Developer`s Guide and Reference*. [Libro Electrónico]. Oracle Corp.

[7] Wikipedia. (2016, 17 de septiembre). “List of object-relational mapping software”. De *Wikipedia The Free Encyclopedia*. https://goo.gl/dzT9uO.