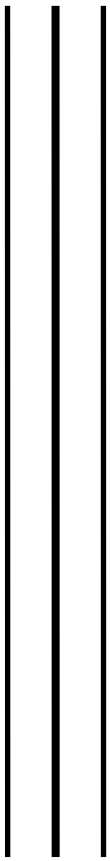
UNIVERSIDAD NACIONAL AUTÓNOMA

DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN



MEJORA DEL DESEMPEÑO EN SISTEMAS WEB CON HIBERNATE SOBRE SOFTWARE LIBRE

DESARROLLO DE UN CASO PRACTICO

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A :

JORGE PÉREZ GARCÍA

ASESOR DE TESIS:

M. en I. ELIO VEGA MUNGUÍA

ESTADO DE MÉXICO

2014

A mis Padres y hermanos, para aquellos que ya no estuvieron para verme terminar y a todo aquel que insistió colaboró, apoyó y motivó a terminar este proyecto.

1 Introducción

Este documento es el resultado del trabajo en el área de desarrollo con sistemas Web. Cuyo objetivo es crear sistemas confiables que cumplan con los requerimientos establecidos por los diferentes clientes y sus usuarios finales.

Hibernate es un framework de desarrollo que funciona como intérprete para que los objetos que forman parte de la aplicación interactúen con las bases de datos relacionales. Pero no basta con instalar y configurar Hibernate para que la aplicación sea optima, se debe probar el rendimiento y adecuarlo o realizar modificaciones para obtener un mejor desempeño.

Para ello a lo largo de este documento se establecerán las bases para entender el proceso de optimización de Hibernate en proyectos ya establecidos. Primeramente se abordará el marco teórico en el cuál se elabora un breve resumen de la importancia de cada una de las aplicaciones utilizadas junto con Hibernate en cada proyecto de desarrollo.

La segunda parte es una introducción a las características principales de Hibernate. Su instalación y configuración básica así y los listados de los atributos más importantes y la estructura de los archivos de mapeo y configuración.

El tercer capítulo es entonces, la parte más importante de este documento, en el se definen problemas reales presentados en 3 proyectos con 2 versiones de Hibernate, una versión para los dos primeros y otra versión para el tercero. De esta forma queda reflejado en este documento.

- La primera experiencia formal con Hibernate.
- Actualización de un primer proyecto con Hibernate por cambio de requerimientos y crecimiento de la aplicación.
- Desarrollo de un tercer proyecto con la experiencia de los otros 2.
- Diferencias entre las versiones de Hibernate.

Finalmente se mencionan las conclusiones de acuerdo a la experiencia práctica empleando Hibernate como capa de persistencia.

2 Marco Teórico

2.1 Descripción

El presente capítulo tiene como finalidad establecer el marco teórico en lo referente a herramientas y tecnologías que pueden relacionarse con Hibernate. El número de elementos cubiertos se limita a aquellas herramientas utilizadas en los 3 proyectos de desarrollo con Hibernate como capa de persistencia que forman parte de la experiencia de quien elabora este documento.

Desde el lenguaje para consultas de bases de datos, así como el lenguaje para definir datos XML el cual es pieza fundamental para establecer la correspondencia entre registros y objetos. El servidor de aplicación -Apache- hasta el lenguaje de desarrollo -Java - y el modelo vista controlador –MVC- empleado en la estructura de los 3 proyectos

ORACLE	APACHE	W3C	
Java	ORACLE JDEVELOPER	NetBeans	APACHE ANT
eclipse	MyEclipse 5.1	Sun microsystems	XML
HTML	MySQL	Java-Script	AJAX Asynchronous Javascript And XML

Tabla 2-1 Diferentes aplicaciones

2.2 SQL

Se trata de un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales permitiendo gran variedad de operaciones sobre los mismos.

2.2.1 Origen

En 1970 Edgar F. Codd propuso el modelo relacional y asociado a éste un sublenguaje de acceso de datos basado en el cálculo de predicados. Basándose en estas ideas, los laboratorios de IBM definen el lenguaje *SQL* (*Structured English QUEry Language*) que más tarde sería ampliamente implementado por el *SGBD*1. El lenguaje SQL terminaría siendo el predecesor de SQL, siendo este una versión evolucionada del primero. El SQL pasa a ser el lenguaje por excelencia de los diversos SGBD relacionales surgidos en los años siguientes, y es por fin estandarizado en 1986 por el ANSI, dando lugar a la primera versión estándar de este lenguaje, el SQL-86 o SQL1. Al año siguiente este estándar es también adoptado por la ISO.

Sin embargo este primer estándar GNU cubre todas las necesidades de los desarrolladores e incluye funcionalidades de definición de almacenamiento que se consideraron suprimir en las siguientes versiones. Así que en 1992 se lanza un nuevo estándar ampliado y revisado del SQL llamado SQL-92 o SQL2.

En la actualidad el SQL es el estándar de facto de la inmensa mayoría de los SGBD comerciales y, aunque la diversidad de añadidos particulares que incluyen las distintas implementaciones comerciales del lenguaje es amplia, el soporte al estándar SQL-92 es en general y muy amplio.

2.2.2 Características generales

Es un lenguaje declarativo de alto nivel, que gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros, y acceso a registros individuales, permite una alta productividad en

1 (*Sistema Gestionador de Bases de Datos*) experimental System R, desarrollado en 1977 también por IBM.

codificación. De esta forma una sola sentencia puede equivaler a uno o más programas utilizados en un lenguaje de bajo nivel orientado a registro.

Funcionalidad. Proporciona una amplia funcionalidad más allá de la simple consulta (o recuperación) de datos. Asume el papel de lenguaje de definición de datos (LDD), lenguaje de definición de vistas (LDV) y lenguaje de manipulación de datos (LMD). Además permite la concesión y denegación de permisos, la implementación de restricciones de integridad y controles de transacción, y la alteración de esquemas.

Modos de uso. El SQL permite fundamentalmente dos modos de uso el primero es un uso interactivo, destinado principalmente a los usuarios finales avanzados u ocasionales, en el que las diversas sentencias SQL se escriben y ejecutan en línea de comandos, o un entorno semejante. El segundo es un uso integrado, destinado al uso por parte de los programadores dentro de programas escritos en cualquier lenguaje de programación anfitrión. En este caso el SQL asume el papel de sublenguaje de datos.

En el caso de hacer un uso embebido del lenguaje se pueden utilizar dos técnicas alternativas de programación. En una de ellas, el lenguaje se denomina SQL estático, las sentencias utilizadas cambian durante la ejecución del programa. En la otra, donde el lenguaje recibe el nombre de SQL dinámico, se produce una modificación total o parcial de las sentencias en el transcurso de la ejecución del programa.

La utilización de SQL dinámico permite mayor flexibilidad y mayor complejidad en las sentencias, pero como contra parte se obtiene una eficiencia menor y el uso de técnicas de programación más complejas en el manejo de memoria y variables.

Optimización. Como ya se hizo mención anteriormente, y como suele ser común en los lenguajes de acceso a bases de datos de alto nivel, el SQL es un lenguaje declarativo. Es decir, que especifica que es lo que se quiere y como conseguirlo, por lo que una sentencia establece explícitamente un orden de ejecución.

El orden de ejecución interno de una sentencia puede afectar gravemente a la eficiencia del SGBD, por lo que será necesario que éste lleve a cabo una optimización antes de la ejecución de la misma.

2.3 XML

Extensible Markup Language. lenguaje de marcas extensible, es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Y permite definir la gramática de lenguajes específicos, de la misma manera que HTML es a su vez un lenguaje definido por SGML, por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunas de sus características son:

- Es extensible, lo que quiere decir que una vez diseñado un lenguaje y puesto en producción, igual es posible extenderlo con la adición de nuevas etiquetas de manera de que los antiguos consumidores de la vieja versión todavía puedan entender el nuevo formato.
- El analizador es un componente estándar, no es necesario crear un analizador específico para cada lenguaje. Esto posibilita el empleo de uno de los tantos disponibles. De esta manera se evitan bugs y se acelera el desarrollo de la aplicación.
- Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarlo ya que es autodescriptivo.
- Mejora y promueve la compatibilidad entre aplicaciones diferentes.

Una etiqueta consiste en una marca hecha en el documento, que señala una porción de éste como un elemento, un pedazo de información con un sentido claro y definido. Las etiquetas tienen la forma **<etiqueta>**, donde etiqueta es el nombre del elemento que se está señalando.

<nombre atributos />

…

<nombre>

…

Atributos

</nombre>

Los atributos están determinados por pares.

Atributo= “Valor”

- Los documentos han de seguir una estructura estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus elementos.
- Los documentos XML sólo permiten un elemento raíz del que todos los demás sean parte.
- Los valores atributos en XML siempre deben estar encerrados entre comillas simples o dobles.
- El XML es sensible a mayúsculas y minúsculas.
- Es necesario asignar nombres a las estructuras, tipos de elementos, entidades, elementos particulares, etc.
- Las construcciones como etiquetas, referencias de entidad y declaraciones se denominan marcas; son partes del documento que el procesador XML espera entender. El resto del documento entre marcas son los datos entendibles por las personas.

2.4 Linux

Es la denominación de un sistema operativo basado en UNIX y el nombre de un núcleo o kernel. Es uno de los paradigmas más prominentes del software libre y del desarrollo del código abierto, cuyo código fuente está disponible públicamente, para que cualquier persona pueda libremente usarlo, estudiarlo, redistribuirlo y, con los conocimientos informáticos adecuados, modificarlo.

Los primeros sistemas Linux se originaron en 1992, al combinar utilidades de sistema y librerías del proyecto GNU con el núcleo Linux, completando un sistema también conocido como GNU /Linux. Desde fines de 1990 Linux ha obtenido el apoyo de diversas empresas multinacionales del mundo de la informática, tales como IBM, Sun Microsystems, Hewlett-Packard y Novell.

Linux es usado como sistema operativo en una amplia variedad de plataformas de hardware y computadores, incluyendo computadoras de escritorio (PCs x86 y x86-64, y Macintosh y PowerPC), servidores, supercomputadores, mainframes, y dispositivos empotrados así como en teléfonos celulares.

2.4.1 Origen

Linux nació gracias a la idea de Linus Torvalds de crear un sistema basado en el sistema minix para máquinas i386; La historia de Linux está fuertemente vinculada a la del proyecto GNU. El proyecto GNU, iniciado en 1983, tiene como objetivo el desarrollo de un sistema Unix completo compuesto enteramente de software libre. Hacia 1991, cuando la primera versión del núcleo Linux fue liberada, el proyecto GNU había producido varios de los componentes del sistema operativo, incluyendo un

intérprete de comandos, una biblioteca C y un compilador, pero aún GNU no contaba con el núcleo que permitiera completar el sistema operativo.

Entonces, el núcleo creado por Linus Torvalds, quien se encontraba entonces estudiando en la Universidad de Helsinki, llenó el hueco final que el sistema operativo GNU exigía. Subsecuentemente, miles de programadores voluntarios alrededor del mundo han participado en el proyecto, mejorándolo continuamente. Torvalds y otros desarrolladores de los primeros días de Linux adaptaron los componentes de GNU y de BSD, así como de otros muchos proyectos como Perl, Apache, Python, etc. para trabajar con el núcleo Linux, creando un sistema operativo completamente funcional procedente de muchísimas fuentes diferentes, la mayoría libres.

2.4.2 Distribuciones Linux

Una distribución es un conjunto de aplicaciones reunidas por un grupo, empresa o persona para permitir instalar fácilmente un sistema Linux. Es un sabor de Linux. En general se destacan por las herramientas para configuración y sistemas de paquetes de software a instalar.

Existen numerosas distribuciones Linux, ensambladas por individuos, empresas y otros organismos. Cada distribución puede incluir cualquier número de software adicional, incluyendo software que facilite la instalación del sistema. La base del software incluido con cada distribución incluye el núcleo Linux y las herramientas GNU, al que suelen adicionarse también varios paquetes de software.

La creciente popularidad de Linux se debe a las ventajas que presenta ante otros tipos de software. Entre otras razones se debe a su estabilidad, al acceso a las fuentes (lo que permite personalizar el funcionamiento y auditar la seguridad y privacidad de los datos tratados), a la independencia de proveedor, a la seguridad, a la rapidez con que incorpora los nuevos adelantos tecnológicos (IPv6, microprocesadores de 64 bits), a la escalabilidad (se pueden crear clusters de cientos de computadoras), a la activa comunidad de desarrollo que hay a su alrededor, a su interoperatibilidad y a la abundancia de documentación relativa a los procedimientos.

2.5 Apache Struts

Struts es una herramienta de soporte para el desarrollo de aplicaciones Web bajo el patrón Modelo Vista controlador (MVC) empleando la plataforma J2EE (Java 2, Enterprise Edition). Struts se desarrollaba como parte del proyecto Jakarta de la Apache Software Foundation 2

Struts permite reducir el tiempo de desarrollo. Su carácter de "software libre" y su compatibilidad con todas las plataformas en que Java Entreprise esté disponible, lo convierte en una herramienta altamente disponible.

2.5.1 Funcionamiento de Struts

Struts se basa en el patrón del MVC, el cual se utiliza ampliamente y es considerado de gran solidez. De acuerdo con este modelo, el procesamiento se separa en tres secciones diferenciadas, llamadas el modelo, las vista y el controlador.

Cuando se programan aplicaciones Web con el patrón MVC, siempre surge la duda de usar un solo controlador o usar varios controladores, se considera mejor usar un solo controlador para que toda la lógica se ubique en un mismo lugar, aquí se presenta un grave problema, ya que el controlador se convierte en lo que se conoce como un "fat controller", es decir, un controlador de peticiones, Struts surge como la solución a este problema ya que implementa un solo controlador (ActionServlet) que evalúa las peticiones del usuario mediante un archivo configurable (struts-config.XML).

2.5.2 Componentes del modelo

Corresponden a la lógica del negocio con la cual se comunica la aplicación web. Usualmente el modelo comprende accesos a Bases de Datos o sistemas que funcionan independientemente de la aplicación web.

2 Actualmente es un proyecto independiente conocido como Apache Struts.

2.5.3 Componentes del control

Los componentes de control son los encargados de coordinar las actividades de la aplicación, que van desde la recepción de datos del usuario, las verificaciones de forma y la selección de un componente del modelo a ser llamado. Por su parte los componentes del modelo envían al control sus eventuales resultados o errores de manera de poder continuar con otros pasos de la aplicación.

Esta separación simplifica enormemente la escritura tanto de vistas como de componentes del modelo: Las páginas JSP tienen que incluir manejo de errores, mientras que los elementos del control simplemente deciden sobre el paso siguiente.

Entre las características de Struts se pueden mencionar:

- Configuración del control centralizada.
- Interrelaciones entre acciones y página u otras acciones se especifican por tablas XML en lugar de codificarlas en los programas o páginas.
- Componentes de aplicación, que son el mecanismo para compartir información bidireccionalmente entre el usuario de la aplicación y las acciones del modelo.
- Librerías de entidades para facilitar la mayoría de las operaciones que generalmente realizan las páginas JSP.

Struts contiene herramientas para validación de campos de plantillas bajo varios esquemas que van desde validaciones locales en la página (en javaScript) hasta las validaciones de fondo hechas a nivel de las acciones.

Struts permite que el desarrollador se concentre en el diseño de aplicaciones complejas como una serie simple de componentes del Modelo y de la vista intercomunicados por un control centralizado. Diseñando de esta manera puede obtenerse una aplicación más consistente y más fácil de mantener.

2.6 Tomcat

Tomcat (también llamado Jakarta Tomcat o Apache Tomcat) funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de Java Server Pages (JSP) de Sun Microsystems.

2.6.1 Entorno

Tomcat es un servidor web con soporte de servlets y JSPs. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache. Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

Tomcat es mantenido y desarrollado por miembros de la Apache Software Foundation y voluntarios independientes. Los usuarios disponen de libre acceso a su código fuente y a su forma binaria en los términos establecidos en la *Apache Software Licence*. Las primeras distribuciones de Tomcat fueron las versiones 3.0.x. Las versiones más recientes son las 6.x, que implementan las especificaciones de Servlet 2.4 y de JSP 2.0. A partir de la versión 4.0, Jakarta Tomcat utiliza el contenedor de servlets Catalina.

2.6.2 Características

- Tomcat 5.x.
- implementado a partir de las especificaciones Servlet 2.4 y JSP 2.0
- recolección de basura reducida
- capa envolvente nativa para Windows y Unix para la integración de las plataformas

2.6.3 Origen

Tomcat empezó siendo una implementación de la especificación de los servlets comenzada por James Duncan Davidson, quien trabajaba como arquitecto de software en Sun y que posteriormente ayudó a hacer el proyecto *open source* y en su donación a la Apache Software Foundation.

Duncan Davidson inicialmente esperaba que el proyecto se convirtiese en *open source* y dado que la mayoría de los proyectos open source tienen libros de O'Reilly asociados con un animal en la portada, quiso ponerle al proyecto nombre de animal. Eligió *Tomcat* (gato), pretendiendo representar la capacidad de cuidarse por sí mismo, de ser independiente.

2.7 Java

Es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 1990. Las aplicaciones Java están típicamente compiladas en un *byte code*, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el *byte code* es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del *bytecode* por un procesador Java también es posible.

Sun Microsystems proporciona una implementación GNU General Public License de un compilador Java y una máquina virtual Java, conforme a las especificaciones del Java Community Process, aunque la biblioteca de clases que se requiere para ejecutar los programas Java es software libre.

2.7.1 Historia

Java comenzó como un proyecto llamado "Green" y su objetivo inicial era crear un lenguaje que fuera capaz de ejecutarse en electrodomésticos que tuvieran microprocesadores pero se dieron cuenta que ese tipo de tecnología estaba aún muy lejos de poder existir. El proyecto dio como resultado un lenguaje muy parecido a C/C++3, el cual fue llamado "Oak" (en referencia al roble que se encontraba en el exterior de las oficinas de Sun Microsystems) por James Gosling, en junio de 1991 para usarse en un proyecto de receptor digital externo, pero descubrieron que ya existía un lenguaje con este nombre, luego alguien sugirió el nombre de Java y fue ese nombre el que quedó. La primera implementación pública fue Java 1.0 en 1995. Prometía "Escribir una vez, ejecutar en cualquier parte" ("Write once, run anywhere"), proporcionando ningún coste extra en el tiempo de ejecución en las plataformas populares. Era bastante seguro y su seguridad era configurable, permitiendo restringir el acceso a archivos o a una red. Los principales navegadores web pronto incorporaron la capacidad de ejecutar "applets" Java seguros dentro de páginas web. Java adquirió popularidad rápidamente. Con la llegada de "Java 2", las nuevas versiones tuvieron múltiples configuraciones pensadas para diferentes tipos de plataformas. Por ejemplo, J2EE era para aplicaciones de empresa y la versión reducida J2ME era para aplicaciones para móviles. J2SE era la designación para la Edición Estándar.

---

3 El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel como punteros.

En 2006, las nuevas versiones "J2" fueron renombradas a Java EE, Java ME y Java SE, respectivamente.

En 1997, Sun se dirigió al cuerpo de estándares ISO/IEC JTC1 y más tarde a Ecma International para formalizar Java, pero pronto se retiró del proceso. Java permanece como un estándar de facto propietario que está controlado a través del Java Community Process. Sun hace disponibles la mayoría de sus implementaciones Java sin cargo alguno, generando los ingresos con productos especializados como el Java Enterprise System. Sun distingue entre su Software Development Kit (SDK) y su Java Runtime Environment (JRE) que es un subconjunto del SDK, siendo la principal distinción que en el JRE GNU está presente el compilador.

2.7.2 Java y DB

Muchos programadores quizás tengan mayor interés en realizar programación basada conjunto a Bases de Datos, pues Java GNU se queda atrás, Java GNU implementa Bases de Datos, ya que sólo es un lenguaje de programación, pero implementa funciones que permiten al programador realizar conexiones entre la interfaz de usuario y el Gestor de Base de Datos.

Java permite conectarse por medio de puentes JDBC, o a través de Drivers, a programas gestores de bases de datos, su independencia entre ambos permite al usuario mantener siempre un enfoque, separando el diseño de la Base de Datos y el de la interfaz en dos mundos de pensamientos diferentes el mundo de los datos y el mundo de las interfaces.

Java es orientado a objetos, por ende da solidez a la aplicación evitando cortes bruscos del programa y permitiendo continuar de esta manera con la aplicación.

2.7.3 Historia reciente

Las aplicaciones Java son las más usadas en los teléfonos móviles por GNU ocupar tanto mucho en la memoria, dejando mucho espacio libre para otros archivos que GNU puede cambiar y que ocupan mucho espacio como los mp3 entre otros. Principalmente se utiliza Java para juegos, pero hay muchas otras formas interesantes de usos.

Respecto a aplicaciones dirigidas al cliente. La capacidad de la continuidad del uso de Java por el gran público. Flash está más extendido para animaciones interactivas y los desarrolladores están

empezando a usar la tecnología AJAX también en este campo. Java suele usarse para aplicaciones más complejas como la zona de juegos de Yahoo, Yahoo! Games, o reproductores de video.

En la parte del servidor, Java es más popular que nunca, con muchos sitios empleando páginas Java Server, conectores como Tomcat para Apache y otras tecnologías Java.

En el PC de escritorio. Aunque cada vez la tecnología Java se acerca más y más al PC de sobremesa, las aplicaciones Java han sido relativamente raras para uso doméstico, por varias razones.

Las aplicaciones Java pueden necesitar gran cantidad de memoria física.

La Interfaz Gráfica de Usuario (GUI) GNU sigue de forma estricta la Guía para Interfaces Humana (Human Interface Guidelines. La apariencia de las fuentes GNU tiene las opciones de optimización activadas por defecto, lo que hace aparecer al texto como si fuera de baja calidad.

Las herramientas con que cuenta el JDK GNU son suficientemente potentes para construir de forma simple aplicaciones potentes. Aunque el uso de herramientas como Eclipse, un IDE con licencia libre de alta calidad, facilita enormemente las tareas de desarrollo.

2.7.4 Orientado a Objetos

La primera característica, orientado a objetos (“OO”), se refiere a un método de programación y al diseño del lenguaje. Aunque hay muchas interpretaciones para OO, una primera idea es diseñar el software de forma que los distintos tipos de datos que use estén unidos a sus operaciones. Así, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el “comportamiento” (el código) y el “estado” (datos). El principio es separar aquello que cambia de las cosas que permanecen inalterables. Frecuentemente, cambiar una estructura de datos implica un cambio en el código que opera sobre los mismos, o viceversa. Esta separación en objetos coherentes e independientes ofrece una base más estable para el diseño de un sistema software. El objetivo es hacer que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos. Otra de las grandes promesas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización del software entre proyectos, una de las premisas fundamentales de la Ingeniería del Software. Un objeto genérico “cliente”, por ejemplo, debería en teoría tener el mismo conjunto de comportamiento en diferentes proyectos, sobre todo

cuando estos coinciden en cierta medida, algo que suele suceder en las grandes organizaciones. En este sentido, los objetos podrían verse como piezas reutilizables que pueden emplearse en múltiples proyectos distintos, posibilitando así a la industria del software para construir proyectos de envergadura empleando componentes ya existentes y de comprobada calidad; conduciendo esto finalmente a una reducción drástica del tiempo de desarrollo. Podemos usar como ejemplo de objeto el aluminio. Una vez definidos los datos (peso, maleabilidad, etc.), y su "comportamiento" (soldar dos piezas, etc.), el objeto "aluminio" puede ser reutilizado en el campo de la construcción, del automóvil, de la aviación, etc.

2.8 Hibern8IDE

Hibern8IDE es una herramienta gráfica interactiva para Hibernate que carga la configuración de Hibernate y los archivos de mapeo de la aplicación con el objetivo de

- codificar y ejecutar consultas tanto en HQL
- codificar y ejecutar consultas tanto en lenguaje nativo
- codificar y ejecutar consultas tanto mediante Criteria(por ejemplo, QBE)
- ver el resultado de forma gráfica
- esta basado en swing y puede utilizarse de manera individual en el IDE o directamente desde la aplicación Hibernate, se encuentra disponible en el Hibernate Extensions package.
-
- Está basado en los siguientes components open source:
-
- Docking framework: http://www.eleritec.net/projects/docking.html
- Look'n'Feel & Layout: https://jgoodies.dev.java.net/
- Propertysheet: http://www.L2FProd.com
- Graph component: http://www.jgraph.org
- Java Object Inspector: http://www.programmers-friend.org/JOI

3 Hibernate

3.1 Descripción

Es una herramienta de Mapeo objeto-relacional para la plataforma Java, que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones.

Figura 1 Logo de Hibernate.

Hibernate es software libre, distribuido bajo los términos de la LGPL (Licencia Pública General Menor de GNU).

La razón principal por la que se utilizó Hiberante es el ahorro de líneas codificadas dedicadas sólo a transacciones con la base de datos, de igual manera tener un mejor control en cuanto al manejo de errores de la BD y evitar elaborar complejos algoritmos para cada uno de los casos de excepciones posibles así como las consecuencias que estos acarrean a otras tablas, dejando a Hibernate todo ese trabajo.

En este capítulo se describen los elementos mas importantes de Hibernate. Desde la instalación y configuración básica como los elementos más comunes junto con una breve descripción de cada uno de ellos.

Figura 2. Gavin King. Fundador del proyecto Hibernate

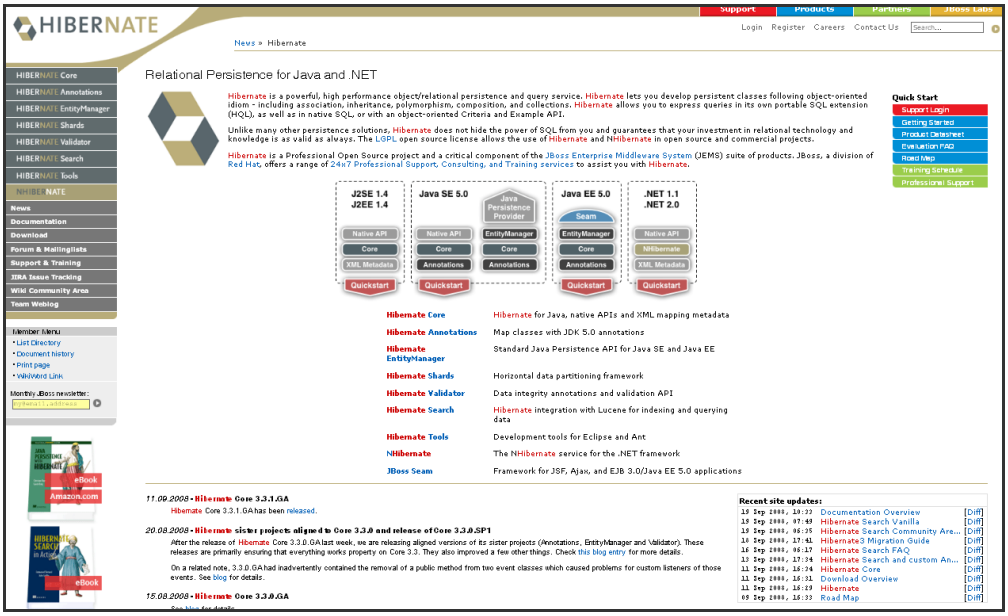


Figura 3. Sitio de Hibernate.

3.1.1Estructura

Hibernate se adapta al proceso de desarrollo de software, sin importar si se parte de una base de datos en blanco o de una ya existente.

La característica principal de Hibernate es el mapeo de clases en Java a tablas de una base de datos, y de tipos de datos de Java hacia tipos de datos de SQL, ofreciendo también consulta de datos y facilidades de recuperación. Hibernate genera las sentencias SQL y libera al desarrollador del manejo

manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todas las bases de datos con un ligero incremento en el tiempo de ejecución.

Hibernate para Java puede ser utilizado en aplicaciones Java independientes o bajo aplicaciones Java EE haciendo uso de servlets o EJB beans de sesión.

3.1.2 Origen

Hibernate fue una iniciativa de un grupo de desarrolladores dispersos alrededor del mundo conducidos por Gavin King. Tiempo después, JBoss Inc. (empresa comprada por Red Hat) contrató a los principales desarrolladores de Hibernate y trabajó con ellos en brindar szoporte al proyecto.

La rama actual de desarrollo de Hibernate es la 3.x, la cual incorpora nuevas características, como una nueva arquitectura Interceptor/Callback, filtros definidos por el usuario, y —opcionalmente— el uso de anotaciones para definir la correspondencia en lugar (o conjuntamente con) los archivos XML. Hibernate 3 también guarda cercanía con la especificación EJB 3.0 (aunque apareciera antes de la publicación de dicha especificación por la Java Community Process) y actúa como la espina dorsal de la implementación de EJB 3.0 en JBoss.

3.1.3 Estructura de Hibernate

Hibernate funciona como capa de persistencia objeto/relacional y es también un generador de sentencias SQL para bases de datos que permite diseñar objetos persistentes que podrán manejar características propias de la programación orientada a objetos como polimorfismo, sobrecarga, herencia relaciones, colecciones, y un gran número de tipos de datos propios de Java. De manera rápida y optimizada, permite generar y manipular bases de datos en cualquiera de los entornos soportados por Hibernate tales como: Oracle, Informix, DB2, MySql, entre muchos otros.

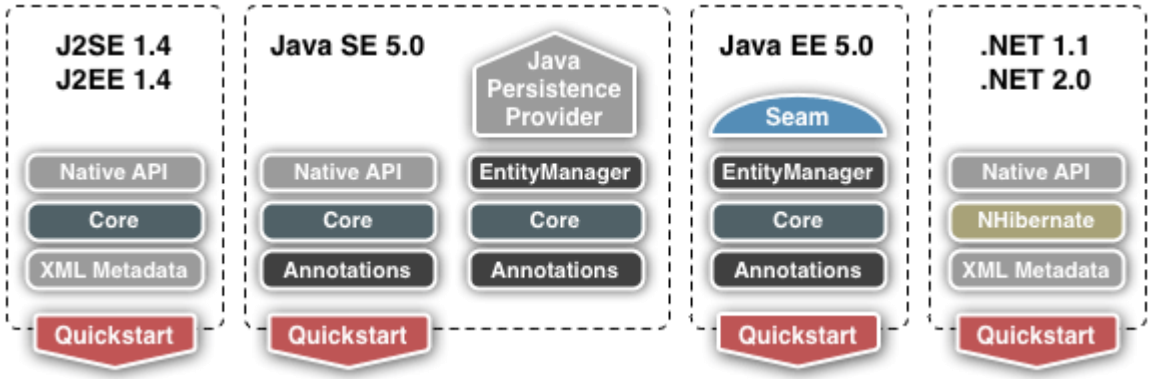


Figura 4. Estructura de Hibernate

- Hibernate Core Hibernate for Java, native APIs and XML mapping metadata
- Hibernate Annotations Map classes with JDK 5.0 annotations
- Hibernate EntityManager Standard Java Persistence API for Java SE and Java EE
- Hibernate Shards Horizontal data partitioning framework
- Hibernate Validator Data integrity annotations and validation API
- Hibernate Search Hibernate integration with Lucene for indexing and querying data
- Hibernate Tools Development tools for Eclipse and Ant
- NHibernate The NHibernate service for the .NET framework JBoss Seam Framework for JSF, Ajax, and EJB 3.0/Java EE 5.0 applications

En este tipo de configuración se puede observar como Hibernate utiliza la base de datos y la configuración de los datos para proporcionar los servicios y los objetos persistentes a la aplicación que se encuentre justo “arriba” de él. Cabe mencionar que Hibernate trabaja con transacciones, es decir, existe una secuencia de estados a lo largo de la operación, cada estado desencadena un subproceso que permite concretar la operación si todo ha salido bien, o abortar todas las sentencias realizadas dentro de la operación si hubo un error en alguna de esas operaciones.

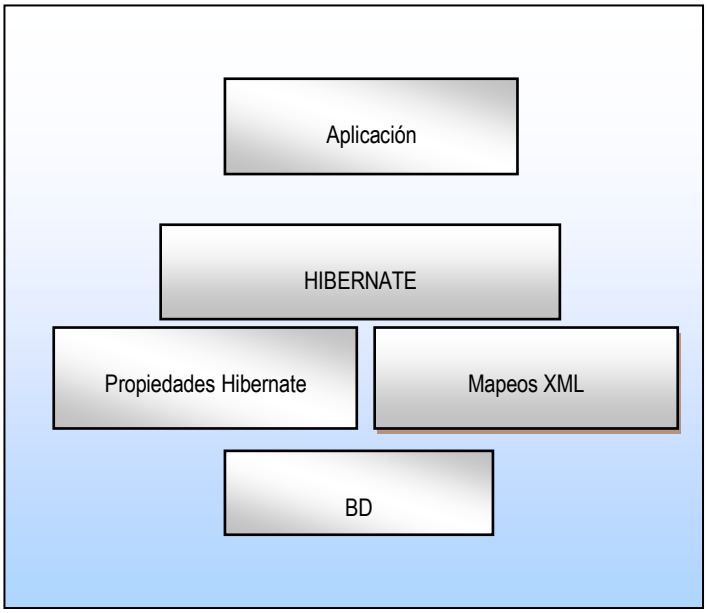


Figura 5 Arquitectura base

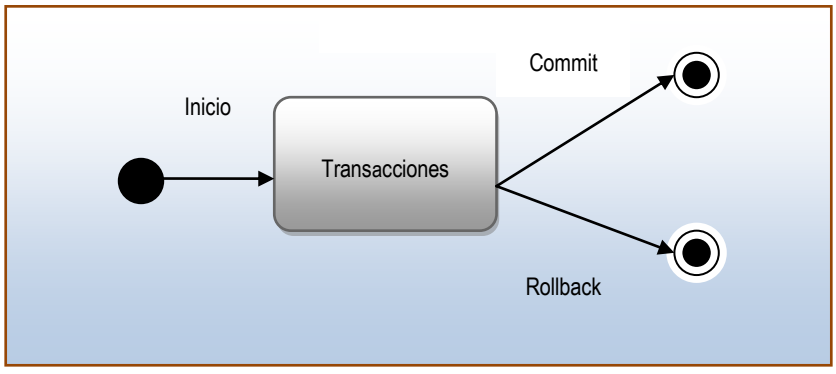


Figura 6 Manejo de transacciones en Hibernate

3.1.4ORM (mapeador objeto relacional)

Es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos, y el utilizado en una base de datos. Esto logra el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo). Hay paquetes comerciales y de uso libre disponibles que desarrollan el mapeo relacional de objetos, Hibernate es uno de ellos.

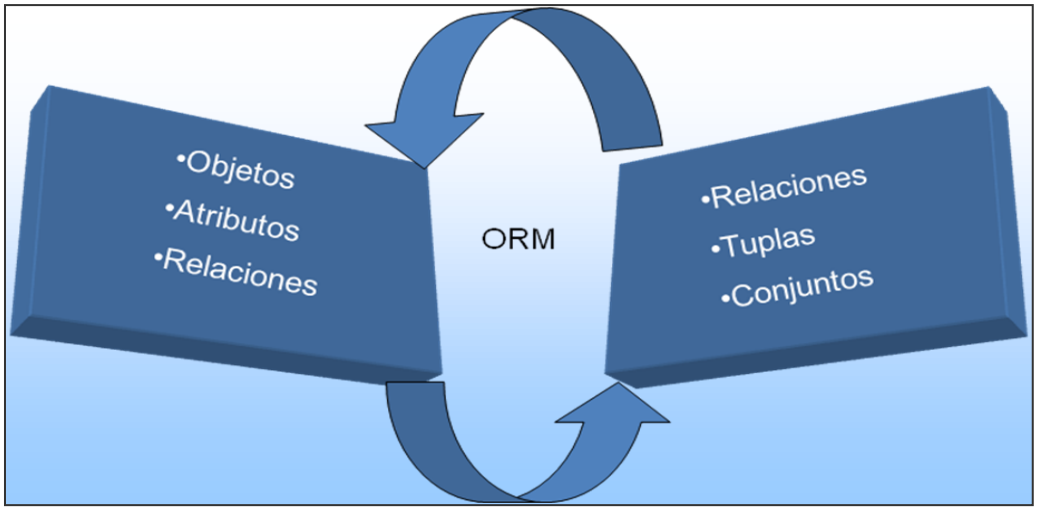


Figura 7. Mecanismo de un ORM

3.2 Puesta en marcha

Antes de comenzar desarrollar en pleno, es necesario cubrir una serie de pasos que forman parte del ciclo de desarrollo, para ello se requiere en primer termino de un diseño sobre el cuál se podrán basar los desarrolladores para implementar los componentes del sistema, este diseño contempla tanto a la base de datos como a las clases que manipularan las tablas dentro de la base de datos, una vez que el diseño de la base de datos esté terminado y validado se procede a establecer los diferentes componentes de acuerdo a las necesidades de la aplicación. Unir estas dos fases es la tarea del ORM, que en cuestión de tiempos puede realizarse poco antes y durante el desarrollo. Las tablas obtenidas y los componentes se convierten en clases y archivos de Hibernate, archivos de mapeo, que son archivos de XML que siguen la DTD de mapeo que requiere Hibernate. Con los archivos XML terminados se podrá generar el código de los objetos persistentes en clases Java y también crear la

base de datos independientemente del entorno seleccionado. La siguiente figura muestra una de las posibles configuraciones de Hibernate.

- Creación de la base de datos
- Librerías
- generación de archivos
- Archivos de configuración
- Archivos de Hibernate

Para la creación de la base de datos se genera y valida la estructura de la base de datos, se toman en cuenta criterios como el tipo, tamaño, formato y las relaciones entre las tablas para asegurar la configuración de los archivos de Hibernate.

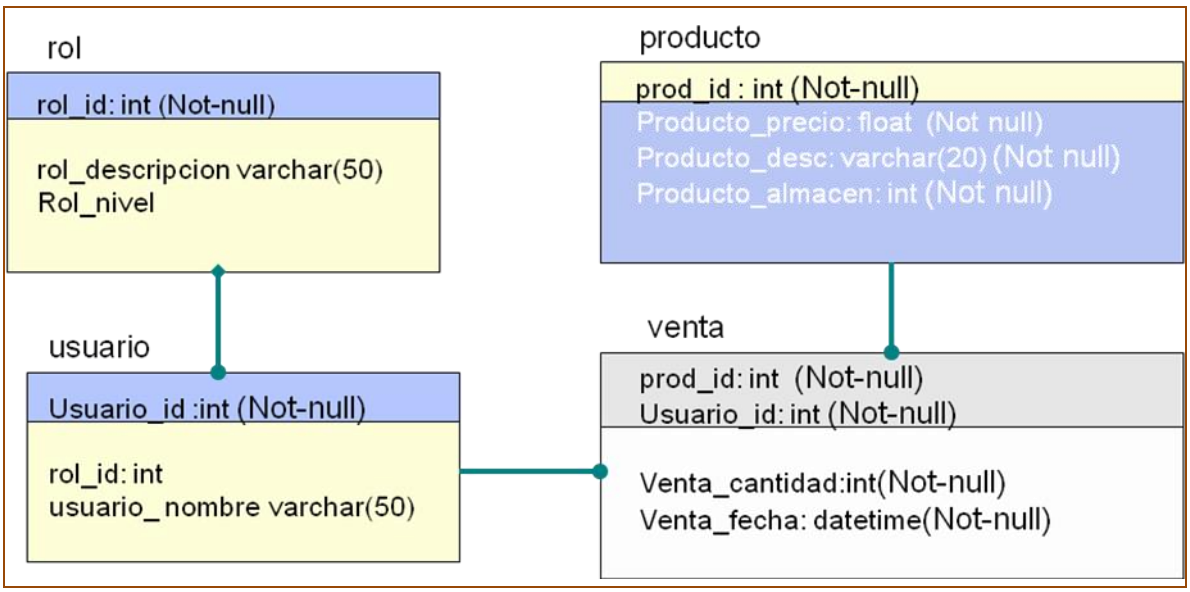


Figura 8 Ejemplo de un DER

Las librerías necesarias para trabajar con Hibernate son:

- antlr.jar
- cglib.jar
- asm.jar
- asm-attrs.jars
- commons-collections.jar
- commons-logging.jar
- hibernate3.jar
- jta.jar

- dom4j.jar
- log4j.jar

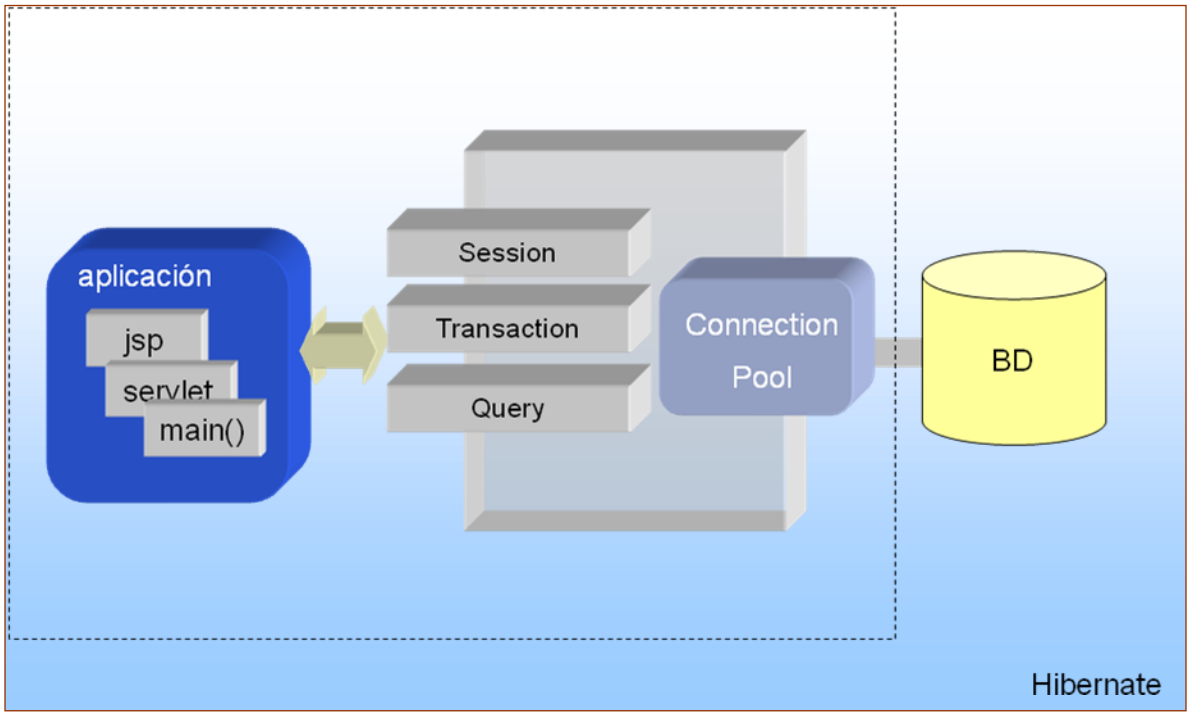


Figura 9. Estructura de la capa de persistencia.

3.2.1Tipos de archivo

Son los diferentes archivos que deben existir para operar con Hibernate, no todos los archivos ni el nombrado de los mismos es obligatorio; depende de la herramienta con la cual se construyan y el esquema elegido por el equipo de desarrollo, el siguiente listado es uno de los más comunes.

- Archivos tipo abstract
- Archivos tipo entity
- Archivos de mapeo (hbm.xml)
- Archivos DAO
- Archivos de configuración

3.2.2 La API de transacciones de Hibernate

En esta interface se encuentran los métodos para realizar las transacciones con la base de datos.

Hibernate requiere un objeto sesisionFactory declarado en el archivo de configuración para poder operar, este objeto generalmente se declara en una sola clase del tipo XXX, es el mismo objeto el que se relaciona con todas las solicitudes realizadas, Las sentencias básicas en la API de transacciones junto con la sentencia correspondiente son:

- Abrir una sesión. Proporciona los métodos para realizar transacciones, en caso de no poderse abrir lanza una excepción.
- Session session = sessions.openSession();
- Inicializar una Variable de Transacción
- Transaccion Tt = null;
- Iniciar una transacción
- AbrirTransaccion = session.beginTransaction(); Es obligatorio colocar un bloque try-catch para terminar o abortar una transacción. En caso de ocurrir un error Hibernate lanza una excepción del tipo HibernateException si esto ocurre la transacción debe abortarse sin realizar ninguna sentencia Tt.Rollback(); Si todo salió bien los cambios se hacen permanentes mediante Tt.commit();
- Cerrar la sesión
- session.close();

3.2.3 Niveles de aislamiento

Una transacción puede bloquear los datos de la tabla que esta por modificar de manera momentánea para que otros accesos no tengan control de ellos y no los modifiquen o les asignen un valor diferente al que la primera transacción está por cambiar. Los diferentes niveles de aislamiento disponibles en Hibernate son:

- Lost update
- Dirty read
- Unrepeatable read
- Second lost updates problem
- Phantom read

3.2.4Niveles

- Read uncommitted
- Read committe
- Repeatable read
- Serializable

3.2.5 Niveles de caché

Primer nivel. En Hibernate la caché de primer nivel corresponde al objeto de sesión4 que se obtiene de una factoría de sesiones5. Esta caché de primer nivel guarda todos los objetos que se van recuperando de la base de datos, de modo que si se vuelven a pedir, se ahorra ese acceso a dicha base de datos; además está caché tiene otras ventajas ya que representa un único punto de acceso a los objetos lo que evita representaciones conflictivas de los mismos, o que los cambios realizados en dichos objetos no sean visibles. Además, como bien se comenta en la documentación, esta sesión actúa como fachada y situaciones como errores de accesos, lazos circulares en las referencias, etc., no afectarán a la totalidad del motor de persistencia sino que sólo afectarán a la sesión involucrada, manteniendo a salvo el resto de sesiones abiertas.

Cuando se realizan operaciones *find()*, *update()*, *save()*, *saveOrUpdate()*, *get(), delete()*, y en general todas las operaciones ofrecidas por la interfaz *Session*, se está interactuando de manera transparente con la caché de primer nivel, ya sea realizando consultas, actualizando o eliminando objetos de la memoria caché.

Cuando se realizan operaciones de actualización, inserción o eliminación de registros, estas operaciones no se realizan directamente, sino que se almacenan en la caché de primer nivel. Para volcar estas operaciones a base de datos es necesario realizar una llamada al método *flush()* de la interfaz *Session*, o al método *commit()* de la transacción abierta para esa sesión(*session.beginTransaction()*). En caso de haber colocado objetos en la caché que ya no se quiera que estén ahí, se puede utilizar el método *evict()* que se elimine de la caché el objeto pasado como

4 net.sf.hibernate.Session

5 net.sf.hibernate.SessionFactory

parámetro. Existe también un método *clear()* que permite eliminar todos los objetos que se encuentren en ese momento en la caché, limpiándola así por completo. Ambos métodos son útiles para políticas de sincronización entre cachés.

Segundo nivel. El problema de las actualizaciones concurrentes supone un grave problema para cualquier sistema de caché. Sea cual sea el modelo que se utilice, siempre habrá problemas con las actualizaciones concurrentes, en mayor o menor medida. La gran ventaja de una caché de segundo nivel, es que ayuda a mitigar estos problemas, al tiempo que alivia de la responsabilidad de almacenar datos temporales, a la caché de primer nivel; responsabilidad que, por otra parte, era la causa de que los modelos de sesión por transacción de aplicación y de sesión por unidad de trabajo pudiesen ralentizar las aplicaciones.

Una caché de segundo nivel se acoplará a la sesión de Hibernate absorbiendo todos los fallos que se produzcan en ésta. Como ya he comentado, la gran ventaja de utilizar una caché de segundo nivel es que desaparecen los problemas de actualizaciones concurrentes entre sesiones. La caché de segundo nivel se sitúa al mismo nivel que el objeto *SessionFactory* de Hibernate, recogiendo y coordinando los objetos con los que trabajan las diferentes sesiones.

La recomendación general con una caché de segundo nivel es utilizar una aproximación de sesión por transacción de aplicación o de sesión por unidad de trabajo, según convenga. Con una caché de segundo nivel, el grave problema de no aprovechamiento de la caché de primer nivel del que adelolecían estos modelos, desaparece parcialmente. Ahora, si un objeto no se encuentra en la caché de primer nivel, Hibernate tratará de obtenerlo de la caché de segundo nivel, de modo que en caso de encontrarse ahí se habrá ahorrado un hit en la base de datos. Cerrar las sesiones pues, ya no es un problema grave, ya que aunque se tenga que realizar cientos y cientos de consultas recurrentes, los datos estarán en la caché de segundo nivel, y la pérdida de rendimiento ya no será grande.

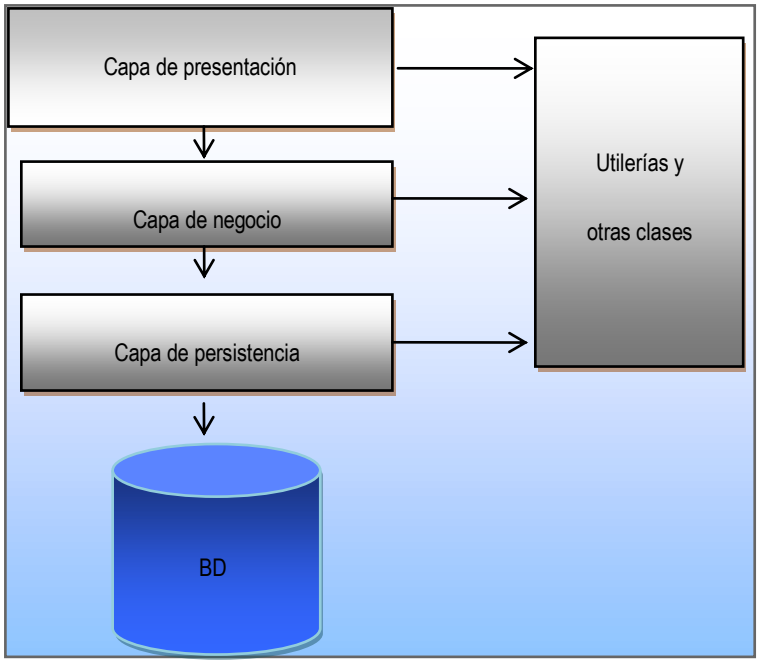


Figura 10 Funcionamiento de la capa de persistencia.

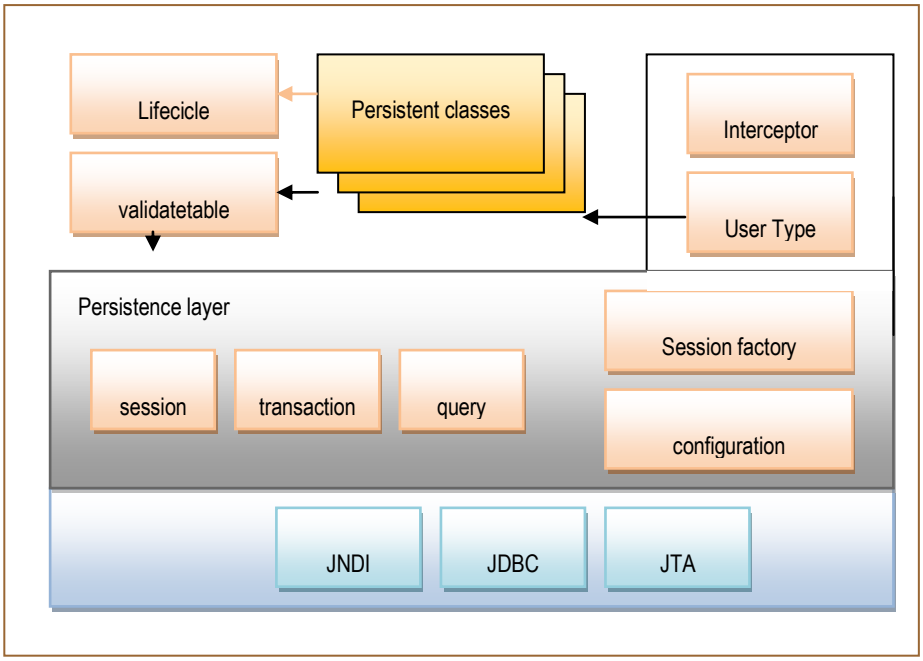


Figura 11 Organización de las capas.

3.2.6 Archivo hbm.xml

También conocido como archivo de mapeo, su función principal es especificar la relación entre una tabla y un objeto mediante una serie de reglas establecidas. Una de esas reglas establece el tipo de dato de cada atributo. La siguiente tabla muestra la correspondencia entre el tipo de dato que debe especificarse en el archivo de mapeo, su correspondiente tipo en Java y finalmente el tipo correspondiente en SQL estándar.6

Mapping type	Java type	Standard SQL built-in type
integer	int o java.lang.Integer	INTEGER
long	Long o java.lang.Long	BIGINT
short	short o java.lang.Short	SMALLINT
float	float o java.lang.Float	FLOAT
double	double o java.lang.Double	DOUBLE
character	java.lang.String	CHAR(1)
string	java.lang.String	VARCHAR
byte	byte o java.lang.Byte	TINYINT
boolean	boolean o java.lang.Boolean	BIT

Tabla 3-1 Tipos de datos

Los tipos de datos del tipo fecha.

Mapping type	Java type	Standard SQL built-in type
date	java.util.Date o java.sql.Date	DATE
time	java.util.Date o java.sql.Time	TIME
timestamp	java.util.Date o java.sql.Timestamp	TIMESTAMP

6 Las tablas de este capítulo fueron tomadas de http://www.hibernate.org/hib_docs/reference/en/html/

calendar	*java.util.Calendar*	*TIMESTAMP*

Tabla 3-2 Datos Binarios o de tipo largo

En esta tabla aparecen el resto de tipos de datos soportados por Hibernate.

Mapping type	*Java type*	*Standard SQL built-in type*
binary	*byte[] VARBINARY (or BLOB)*	
text	*java.lang.String CLOB*	
serializable	*Cualquier clase de Java que implemente java.io.Serializable.*	*VARBINARY (o BLOB)*
clob	*java.sql.Clob*	*CLOB*
blob	*java.sql.Blob*	*BLOB*

Tabla 3-3 Datosrelacionados a JDK

3.3 Configuración y Sintaxis básica.

El siguiente apartado tiene como finalidad una rápida comprensión acerca de la sintaxis y las posibilidades de Hibernate a manera de ejemplos. En este ejemplo se plantea el caso hipotético de un sistema de registro que requiere que diferentes usuarios se registren en el sistema y dependiendo de su perfil serán los recursos a los que tendrán acceso y las actividades que podrán realizar en el sistema. Se considera exclusivamente al registro omitiendo procesos específicos y procesos relacionados para no perder el objetivo de este ejemplo.

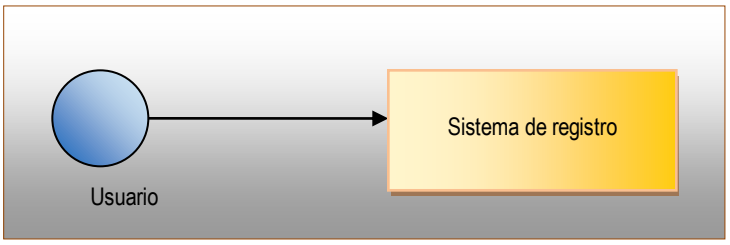


Figura 12. Manejo de transacciones en Hibernate

3.3.1El Archivo de Configuración de Hibernate

Se llama Hibernate.cfg.xml al archivo de configuración por defecto de Hibernate. Cada archivo de configuración (puede existir más de uno) puede representar una conexión a una base de datos distinta, y por supuesto cada archivo de configuración hará referencia a los archivos de mapeo correspondientes a las tablas de cada una de las bases de datos.

Atributo	Propósito
hibernate.connection.driver_class	Nombre del controlador jdbc de acuerdo a la base de datos seleccionada.
hibernate.connection.url	Nombre del controlador jdbc de acuerdo a la base de datos seleccionada.
hibernate.connection.username	Nombre del usuario de la base de datos.
hibernate.connection.password	Password del usuario de la base de datos.
hibernate.connection.pool_size	Indica el tamaño del pool de conexiones.

Tabla 3-4 Propiedades JDBC de Hibernate

Atributo	Propósito
hibernate.connection.datasource	nombre de la fuente de datos JNDI
hibernate.jndi.url	URL del proveedror JNDI (opcional)
hibernate.jndi.class	Clase del JNDI InitialContextFactory (opcional)
hibernate.connection.username	Usuario de la base de datos(opcional)
hibernate.connection.password	Password de la base de datos (opcional)

Tabla 3-5 Propiedades del Hibernate Datasource

Atributo	Propósito
hibernate.dialect	El nombre de la clase de un dialecto de Hibernate que permite que Hibernate genere el SQL optimó para una base de datos relacional en

Atributo	Propósito
	particular.
hibernate.show_sql	Escribir todas las sentencias SQL en la consola. Es un alternativa a establecer la categoría org.hibernate.SQL cómo debug.
hibernate.default_catalog	Calificar en el sql generado los nombres de tablas no calificados con el catálogo dado.
hibernate.session_factory_name	La SessionFactory será; ligada a este nombre en JNDI automáticamente después de ser creada. <emphasis role="strong">ej.</emphasis><literal>jndi/composite/name</literal> </para>
hibernate.generate_statistics	De habilitarse, Hibernate colectará estadísticas útiles para la afinación de rendimiento. E j true|false
hibernate.use_identifer_rollback	De habilitarse, las propiedades identificadoras generadas serán reseteadas a valores por defecto cuando los objetos sean borrados ej true |false
hibernate.use_sql_comments	If turned on, Hibernate will generate comments inside the SQL, for easier debugging, defaults to false. De activarse, Hibernate generará comentarios dentro del SQL, para una más fácil depuración, por defecto a <literal>false</literal>.

Tabla 3-6 Propiedades del origen de dato/para Hibernate: strutsconfig.cfg.xml

3.3.2Archivo de mapeo de Hibernate.

Los archivos de mapeo definen la correspondencia entre una tabla y una clase que hace referencia a esa tabla. Dentro del archivo se encuentran una serie etiquetas como las siguientes:

<hibernate-mapping>. Dentro de estas etiquetas se declaran las clases de cada objeto existente. Mientras <class> es el recurso mediante el cual se hace referencia a la clase persistente.

Atributo	Descripción
Name	Nombre completo de la clase o interface persistente. Deberemos incluir el nombre del paquete dentro del nombre (no es obligatorio pero es lo mas recomendable)

Table	*Nombre de la tabla en la BD referenciada. En esta tabla se realizará las operaciones de transacciones de datos. Se guardarán, modificarán o borrarán registros según la lógica de negocio de la aplicación.*
discriminator-value	*Permite diferenciar dos sub-clases. Utilizado para el polimorfismo.*
Proxy	*Nombre de la clase Proxy cuando esta sea requerida.*

Tabla 3-7 Atributos del elemento class

La etiqueta ***<id>*** Permite definir el identificador del objeto. Corresponde a la clave principal de la tabla en la base de datos. Es importante definir en este momento lo que será para la aplicación un ***OID*** (objectt Identifier / Identificador de Objeto) el cuál se requiere asignar identificadores únicos a cada objeto persistente.

Atributo	***Descripción***
Name	*Nombre lógico del identificador.*
Column	*Columna de la tabla asociada en la cual almacenaremos su valor.*
Type	*Tipo de dato.*
unsaved-value ("any|none|null|id_value")	*Valor que contendrá el identificador de la clase si el objeto persistente todavía no se ha guardado en la BD.*
generator	*Clase que utilizaremos para generar los OIDs. Si requiere de algún parámetro este se informa utilizando el elemento <paramater name="nombre del parámetro">.*

Tabla 3-8 Atributos del elemento id

Hibernate proporciona clases que generan automáticamente estos OID entre ellas caben destacar :

OID	***Descripción***
Native	
Identity	*Éste generador soporta identity columns en DB2, MySQL,. El tipo de retorno puede ser del tipo long, short o int.*
Sequence	*Utiliza una secuencia definida por un manejador de bases de datos específico*

Increment	*Lee el máximo valor de la columna que forma la llave primaria y la incrementa en 1.*
assigned	*Por si después de todo queremos que los identificadores los gestione la propia aplicación.*

Tabla 3-9 Generadores de OID's más importantes

La etiqueta <discriminator> sirve para cuando una clase requiere declarar un discriminador, es necesaria una columna en la tabla que contendrá el valor de la marca del discriminador. Los conjuntos de valores que puede tomar este campo son definidos en cada una de las clases o sub-clases a través de la propiedad <discriminator-value>. Los tipos permitidos son string, character, integer, byte, short, boolean, yes_no, true_false.

Atríbuto	*Descripción*
column	*El nombre de la columna del discriminador en la tabla.*
type	*El tipo del discriminador.*

Tabla 3-10 Atributos del elemento discriminator

La etiqueta <property> declara una propiedad persistente de la clase, que se corresponde con una columna.

Atríbuto	*Descripción*
name	*Nombre lógico de la propiedad.*
column	*Columna en la tabla.*
type	*Indica el tipo de los datos almacenados.*

Tabla 3-11 Atributos del elemento property

3.3.3Tipos de relaciones. (Componentes y Colecciones)

En todo diseño relacional los objetos se referencían unos a otros a través de relaciones, éstas son relación uno a uno 1-1, uno a muchos 1-n, muchos a muchos n-m, muchos a uno n-1. De los cuatro tipos, dos de ellas n-m y 1-n son colecciones que se tratan más a detalle, debido a que las demás relaciones 1-1 y n-1 son en realidad componentes de un objeto persistente cuyo tipo es otro objeto persistente.

3.3.4Relación many-to-one muchos a uno.

La relación n-1 necesita en la tabla un identificador de referencia, el ejemplo clásico es la relación entre padre e hijos. Un registro hijo necesita un identificador en su tabla que indique cual es su registro padre que a su vez pude ser hijo de otro registro. Pero en objetos en realidad no es un identificador sino el propio objeto padre, por lo tanto el componente n-1 es en realidad el propio objeto padre y no simplemente su identificador. (Aunque en la tabla sólo se guarde el identificador)

Atríbuto	*Descripción*
column	*Columna de la tabla donde se guardara el identificador del objeto asociado.*
class	*Nombre de la clase asociada. Hay que escribir todo el paquete.*
cascade	*Especifica que operaciones se realizaran en cascada desde el objeto padre.*

Tabla 3-12 Atributos de las relaciones n-1

3.3.5 Relación one-to-one

Asociacion entre dos clases persistentes, en la cual no es necesaria otra columna extra debido a que los OID's de las dos clases serán idénticos.

Atríbuto	Descripción
Class	La clase persistente del objeto asociado del objeto asociado.
Cascade ("all | none | save –update | delete")	Operaciones en cascada a partir de la asociación. paquete.
cascade	El nombre de la propiedad.

Tabla 3-13 Atributos de las relaciones 1-1

Atríbuto	Descripción
class	Guarda en un VARCHAR el nombre completo de la clase referenciada.
binary	Mapea un array de bytes al apropiado tipo de SQL.
serializable	Desde una clase que implementa el interface Serializable al tipo binario SQL.
clob, blob	Mapean clases del tipo java.sql.Clob y java.sql.Blob

Tabla 3-14 Objetos tipo blob

4 Optimización

4.1 Descripción

Este último capítulo está basado en la experiencia personal con proyectos que han integrado Hibernate como capa de persistencia, la cuál cubre 3 proyectos de desarrollo que lo han empleado en forma satisfactoria. Cada uno de estos proyectos es de considerable importancia debido a la misión para la que fueron creados pues implican una alta complejidad tanto en su estructura, como por la intervención de múltiples actores para su puesta en marcha, fase de operación, mantenimiento y otro gran grupo de usuarios finales para quienes estaba planeado cada proyecto.

El objetivo principal de este capítulo es detallar cada una de las dificultades encontradas en las distintas etapas de los proyectos teniendo como prioridad la capa de persistencia.

En primer término se describe el objetivo de cada módulo seguido de la propuesta de solución y los argumentos que sirvieron de base para haberla llevado a cabo, teniendo como base la experiencia con proyectos anteriores y la documentación y capacitación con Hibernate.

También se detallan los primeros resultados de las soluciones propuestas para desarrollar cada módulo, las anomalías detectadas en los procesos que estaban más lejos de ser la solución óptima y los procesos que los sustituyeron y el porqué de esa sustitución.

Finalmente, a lo largo de todo el capítulo se mencionan otras variables que afectaron el buen desarrollo de cada proyecto, conforme se presentaron en cada etapa, mencionando sus causas, consecuencias y el mecanismo para resolverlas y continuar con el ciclo de desarrollo del proyecto.

4.2 Hibernate en proyectos de desarrollo.

Generalmente los proyectos, empleen Hibernate o no, están constituidos de la siguiente manera:

- Obtención de requerimientos.
- Análisis de los requerimientos.
- Propuesta de solución.

- Generación de casos de uso.
- Desarrollo.
- Pruebas del sistema.
- Período de garantía.

La obtención de requerimientos Se trata de una serie de entrevistas, de manera verbal o empleando cuestionarios, para obtener las necesidades del cliente como usuario, administrador, por módulo y para el proyecto en su conjunto, así como los números referentes al volumen de trabajo, quienes van a administrar o a proveer de información al sistema, cuántos son los clientes potenciales, con qué cantidad de datos se espera abrir el proyecto y cuánto se espera crecer, en cuanto a clientes y datos de forma mensual o anualmente. Sobre este análisis se evalúan los actores, las clases y procesos que intervienen y se elabora una serie de documentos de solución, diagramas, casos de uso, propuestas de tiempos, alternativas de solución, infraestructura, análisis de servidores de aplicación, tipos de base de datos, sistema operativos y se compara con las requeridas por el cliente, si existen, para realizar lo ajustes necesarios.

El objetivo de esta etapa es asegurar que el documento de solución propuesto contemple todas las necesidades y aprovechando la experiencia del equipo de desarrollo entregar un sistema que cumpla las expectativas del cliente. El resto de las etapas se encadenaran una detrás de la otra cumpliendo en tiempos y productos necesarios para el comienzo de cada etapa para completar cada una de las fases del proyecto.

4.3 Objetivos específicos de cada sistema

Los objetivos específicos que cubre cada primer sistema, se pueden identificar mediante la descripción general de los módulos que lo componen, en el primer proyecto estos son:

- Autenticación. Permite el acceso de los usuarios al sistema mediante la verificación de la información de su usuario y contraseña.
- Administración de la información. Permite agregar y/o modificar la información relativa a los diferentes niveles del sistema.
- Reportes de consulta y control. Permite emitir los reportes relativos a la información de los módulos del sistema.
- Administración e integración de información. Permite integrar y administrar (dar de alta, modificar y eliminar) la información (registros) de los módulos que componen el sistema.
- Bitácora de movimientos. Permitir registrar y consultar los accesos al sistema, así como los movimientos de alta, baja, cambio, consulta y generación de reportes de cualquier tipo de información del sistema.

- Exportación e Importación de la información. Permite importar y exportar la información del sistema en archivos de información (txt).
- Administración de usuarios. Este caso de uso permite crear y modificar cuentas de usuario para controlar el acceso al sistema.
- Consulta pública. Permite emitir los reportes relativos a la información a los usuarios en general.

- Para el segundo proyecto se consideró lo siguiente.
  - Con el uso del sistema, los responsables de las instancias que participan en el análisis de las solicitudes, podrán verificar en línea su estatus y en su caso, dictaminar si procede o no su autorización; lo que permitirá reducir tiempos y costos de operación, logrando una mayor integridad y control de la información. Adicionalmente, por medio del portal se ofrecerá un servicio a los solicitantes que les permitirá conocer el estatus de su solicitud con respecto a la etapa en que se encuentra en ese momento.
  - Realizar el procedimiento para registro y consulta de la información, tanto del interesado como del negocio inscrito en el programa y el seguimiento de los trámites para la apertura de empresas de acuerdo a su especificación. Asimismo proporciona una serie de recomendaciones para la mejor comprensión y uso de cada pantalla y procesos que lo conforman.

4.3.1 Plataforma de desarrollo

Cada Sistema de información vía Web trabaja con un patrón de arquitectura denominado Modelo–Vista–Controlador. Este es un patrón que define la organización independiente del Modelo (Objetos de Negocio), la Vista (interfaz con el usuario u otro sistema) y el Controlador (controlador del flujo de trabajo de la aplicación).

El navegador genera una solicitud que es atendida por el Controlador (un Servlet especializado). Éste se encarga de analizar la solicitud, seguir la configuración que se le ha programado en un archivo XML (Lenguaje Extensible de Marcas) y llamar al Action (función encargada de la tarea solicitada) correspondiente pasándole los parámetros enviados. El Action instanciará y/o utilizará los objetos de negocio para concretar la tarea. Según el resultado que retorne el Action, el Controlador derivará la generación de interfaz a uno o más JSPs (JavaServer Pages), los cuales podrán consultar los objetos del Modelo a fin de realizar su tarea.

La implementación de la arquitectura según el patrón Modelo–Vista–Controlador se lleva a cabo a través de Struts (un marco de trabajo para construir aplicaciones Web Java), gracias a lo cual se simplifica notablemente. Struts separa muy bien lo que es la gestión del flujo de trabajo de la aplicación, del modelo de objetos de negocio y de la generación de la interfaz.

El controlador ya se encuentra implementado por Struts, aunque si fuera necesario se puede heredar y ampliar o modificar, y el flujo de trabajo de la aplicación se puede programar desde un archivo XML (struts-config.xml). Las acciones que se ejecutarán sobre el modelo de objetos de negocio se implementan basándose en clases predefinidas por el marco de trabajo (framework). La generación de la interfaz se soporta mediante un conjunto de etiquetas predefinido por Struts cuyo objetivo es evitar el uso de Scriplets (los trozos de código Java entre "<%" y "%>"), lo cual genera ventajas de mantenimiento y desempeño (pooling de etiquetas, almacenamiento en cache, entre otros).

Logísticamente, Struts separa claramente el desarrollo de la interfaz del flujo de trabajo y de la lógica de negocio permitiendo desarrollar ambas en paralelo o con personal especializado.

También es evidente que Struts potencía la reutilización, el soporte de múltiples interfaces de usuario (HTML [Lenguaje de Marcado de Hipertexto], DHTML [HTML dinámico], WML [Lenguaje de Marcado para Aplicaciones Inalámbricas], aplicaciones personales, entre otros) y de múltiples idiomas, configuraciones locales, etc.

4.4 Requerimientos Mínimos de Hardware y Software

4.4.1 Requerimientos de Hardware

Las características de hardware para la implantación del Sistema de consulta y control fueron determinadas por el cliente, por lo que su descripción queda fuera del alcance del equipo de desarrollo.

4.4.2 Requerimientos de Software - Manejador de base de datos.

- Primer proyecto. Oracle versión.9i. revisión 9.2.0.2. Enterprise Edition.
- Segundo proyecto. Informix versión 10
-

4.4.3Servidor de servicios Web.

- Primer proyecto. Tomcat-Apache versión 5.0.28 con kit de desarrollo Java2 Standard Edition (J2SE) 1.4.2
- Segundo proyecto. Web logic versión 10.

4.4.4Servidor de correo.

- Sendmail versión actualizada.

4.4.5Aplicación cliente.

- Primer proyecto. Java2 Standard Edition (J2SE) 1.4.2
- Segundo proyecto Java JDK 5.0.

Las siguientes características se refieren a los equipos sobre los cuales corre la aplicación, se desarrolló y se realizaron pruebas.

Servidor

Segundo proyecto. Características del equipo:

- Modelo SUnFire880
- Procesador 4x UltraSparc III 1050 MHz
- Disco Duro 12x72GB
- RAM 8192 MB
- Sistema Operativo: Solaris 9

Equipo de desarrollo y pruebas

TIPO 1
Procesador Intel® Pentium® G3220 (3MB Caché, 3.00 GHz) *4 GB[1] Dos canales SDRAM* *4 GB[1] Dos canales SDRAM*

TIPO 2
Pentium(R) D CPU 3.20 GHz *1 GB en RAM* *120 GB de disco duro.*

HERRAMIENTAS DE ANÁLISIS Y DISEÑO
Control de Versiones. MicrosoftVisual Sourrce Safe 6.0 *Diseño. Erwin 4.0*

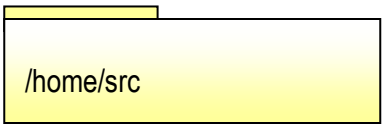
Requisitos de hardware para el cliente del sistema

COMPONENTES	CARACTERÍSTICAS
Microprocesador	*procesador Intel® Core™ i5-4440 (6MB Caché) o superior*
Memoria	*6GB de Memoria Doble Canal DDR3 o superior.*
Tarjeta de red	*10/100 Mbps*

4.4.6Estructura de las carpetas

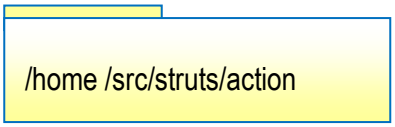
Un sistema se encuentra organizado por carpetas para la fácil ubicación de los archivos. Que simplifique el desarrollo y mantenimiento de código.

Para todos los archivos de código fuente el directorio base es:

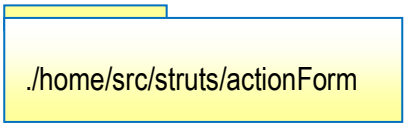


Archivos de métodos accesores (setters y getters) utilizados para generar objetos persistentes que interactúan con la base de datos (capa de Hibernate):

Los archivos de tipo "action" que contienen métodos manejadores, de inicialización y limpiado de valores, coordinación de métodos que accesan a la base de datos así como inicio y fin de transacciones, se encuentran en:



Los archivos de tipo "actionForm" que contienen métodos accesores, de actualización y obtención de valores así como validaciones de la Vista (Capa Cliente, browser), se encuentran en:



Los archivos de tipo JSP (Java Server Pages) se ubican en la carpeta webapps bajo la carpeta de instalación del tomcat

La función de Hibernate en estos proyectos es la de ser la capa de persistencia del sistema proporcionando a todos los módulos lo siguiente:

- Una clase especializada por tabla del tipo POJO (clase Abstract).
- Una clase para definición de las llaves compuestas.
- Una clase para administración del formato de los datos.
- Un archivo de definición por tabla (Archivo hbm.xml).
- Una clase DAO para transacciones con la base de datos.
- Una Consulta por llave primaria.
- Consultas especializadas.
- Método de Inserción.

- Método de Actualización.
- Método de Borrado.
- Una clase para administración del módulo (Action).

4.4.7 Criterios generales.

En cuanto al rendimiento de Hibernate se consideró que debido al volumen de información a manejar junto con el nivel de normalización del esquema de la base de datos, las prestaciones del servidor de aplicaciones y el servidor de bases de datos es una opción más viable manejar las consultas por llave primaria tal y cómo lo sugiere la documentación de Hibernate. En cuanto a las consultas por criterios también se optó por las consultas básicas, pues no se detectó que existieran inconvenientes.

Los archivos de configuración del sistema de la capa de persistencia, hibernate.cfg.xml, archivos DAO, Abstarct, clases, archivos hmb.xml. cambiaron constantemente, estos ajustes se debieron a cambios del cliente en cuanto a su estructura y a limitaciones de la versión de Hibernate para realizar el mapeo de ciertas estructuras de la base de datos, los tipos de datos blob y las llaves circulares por citar algunos de ellos, toda este constante cambio de estructura, aunque implicaba generar completamente todos los archivos de Hibernate, permitían continuar con el proceso de desarrollo.

Un factor determinante para esta exitosa re-estructuración fue el modelo Vista controlador y el manejo de Struts que permiten utilizar archivos dummie, que simulan el funcionamiento de la base de datos y permiten perfeccionar la capa de control y las vistas sin ningún retraso logrando incorporar la capa de persistencia de forma transparente.

El empleo de Struts también implica un contra si se utiliza por primera vez dentro de un proyecto de dimensiones mediano/grande. La curva de aprendizaje de Struts es muy elevada, los múltiples componentes y la estructura acompañada de documentación deficiente-- en la época en la cuál se realizó el desarrollo, actualmente buscar Struts en Internet proporciona cientos de páginas y manuales—no permiten que el aprendizaje sea suave ni intuitivo. Si se acompaña una Interfaz de desarrollo de la cual se conoce poco y se incorpora una capa de persistencia nueva en el mercado se está entrando a un terreno en el que los problemas no serán identificados fácilmente y las propuestas de solución en consecuencia, serán pocas.

El proyecto que forma la segunda etapa es una extensión del anterior. Su objetivo se centra en un incremento en las necesidades del sistema original, que no se habían contemplado al momento de definir los requerimientos de la aplicación. El otro factor determinante para cambiar el sistema en Web

es el hecho que fueron rebasados en cuanto a rendimiento, las soluciones propuestas en primer instancia, el aumento exponencial en el volumen de datos que se requería manejar; que fácilmente superaba el 500% con respecto a la primera versión. Los módulos, submódulos y procedimientos que fue necesario optimizar son:

- Manejo de reportes *
- Formato Impresión -Versión imprimible-
- Formato PDF –Versión portable-
- Formato Web –Consulta en línea-
- Manejo y validación masiva de Información. *
- Carga de datos.
- Descarga de datos.
- Componente de paginación de información. *
- Optimización de consultas a la base de datos por página.
- Vista de registros por página.
- Carga de información de acuerdo a la página seleccionada.
- Capa de persistencia. *
- Optimización de consultas
- Incorporación de nuevos criterios de búsqueda.
- Optimización del pool de conexiones. *

4.5 Ejemplos de Optimización con Hibernate.

En este apartado se detallan las problemáticas más importantes que se presentaron en el desarrollo de los 3 proyectos que se implementaron con Hibernate como ORM.

4.5.1 El proceso de Instalación.

Incorporar Hibernate al proceso de desarrollo es relativamente sencillo si se siguen todos los pasos requeridos para su implantación, ahora bien, el problema radica ahí, para un desarrollador inexperto puede resultar que su primer acercamiento no sea tan satisfactorio como pudiera esperarse pues es

muy probable cometer un error si no se tiene la precaución adecuada. Entre lo que se puede pasar por alto durante la instalación y configuración se destaca lo siguiente.7

Las diferentes versiones y revisiones disponibles de Hibernate. Las primeras versiones de Hibernate requerían librerías que no se encontraban en el paquete descargable de Hibernate, ahora conocido como *Core Hibernate*, todas estas librerías son necesarias para una administración mas completa de la capa persistente. En ese entonces era necesario descargar tanto el núcleo de Hibernate como las librerías que el desarrollador considerara necesarias para su aplicación, al ser software libre algunas de esas librerías eran desarrolladas con la colaboración de terceros y cambiaban constantemente pues se corregían bugs y se optimizaban procesos, hoy en día el software libre sigue evolucionando de esta misma forma y eso no es malo, incorporando mejoras en cuanto a velocidad y rendimiento manteniendo su compatibilidad --finalmente seguían siendo Jars de Java--. Esta actualización constante de librerías presentó problemas significativos cuando apareció la versión 3 de Hibernate. Hibernate 3.0 es una versión más completa que incorpora muchas de las últimas versiones de las librerías en el Core de Hibernate en un mismo paquete ***org.com*** en lugar de las antiguas ***net.sf***. Este sencillo cambio ha provocado una confusión terrible tanto a desarrolladores de las antiguas versiones de Hibernate como a los nuevos. Leer y entender la documentación disponible aseguraría no equivocarse en este respecto aún teniendo en cuenta que la información disponible en el sitio oficial de Hibernate aunque redactada en forma simple esta más orientada a usuarios de nivel medio y avanzado. Todo esto aunado a ejemplos insuficientes para iniciarse en Hibernate también hacen más lenta la comprensión de la herramienta. Cuando se pasa de los primeros ejemplos a la implantación de un sistema en forma no es tan sencillo inferir como solucionar las relaciones complejas entre tablas, la falta de documentación, tanto la oficial como la publicada en diversos foros en la red, puede confundir mas al equipo de desarrollo o incluso ejercer más presión al equipo de desarrollo que no encontrará respuestas y si experiencias de otros desarrolladores con el mismo problema y ninguna solución adecuada al mismo.

Existen herramientas especializadas que hacen el trabajo “sucio” de Hibernate, MyEclipse por ejemplo, pero debe tomarse en cuenta con mayor responsabilidad que las herramientas y/o scripts de generación de archivos cometen errores que deben ser corregidos por el desarrollador quién de ninguna manera estará exento de conocer todos los detalles del funcionamiento de Hibernate.

---

7 http://www.hibernate.org/250.html#A6

El log de transacciones. El log muestra información fundamental para seguir e identificar errores, warnings o casos de éxito, también requiere un elevado nivel de entendimiento para interpretar los diferentes mensajes que envía el log.

Hibernate evita codificar cientos de líneas de código pues es el quien se encarga de administrar el entendimiento entre objetos y registros de la base de datos y el desarrollador no necesita conocer que sucede dentro de esa especie de caja negra, únicamente se limita a indicar qué datos del sistema quiere recuperar o insertar. Pero ¿qué sucede si existe un problema, en el transcurso de ese proceso? Sucede que el log de transacciones captura la línea de código en donde se genero el error, la sección en la que se le solicitó intervenir a Hibernate como intermediario, también se muestran las líneas en las que se produjo internamente el error, en otras palabras el conjunto de estos mensajes no son otra cosa que la caja negra de Hibernate que el desarrollador "no" necesita conocer, líneas y líneas que no parecen tener ningún sentido y unas cuantas líneas que son comprensibles pero que no proporcionan ni una remota idea de cómo resolver el problema. Si se quiere entender qué significado tienen estas líneas hay que leer a profundidad la documentación y adquirir experiencia de cada nuevo proyecto en el que se utilice Hiberrnate. De ninguna manera se pretende excusar que la mayoría de los mensajes enviados por el log de transacciones de Hibernate son pobres y obscuros y en ocasiones le dejan al ensayo y error y, por qué no decirlo, a la suerte la solución del problema.

Seleccionar la infraestructura de desarrollo adecuada. Si se desea realizar el proceso de desarrollar un sistema que funcione vía Web, la primer dificultad que se presenta es que existen diferentes maneras de llevarlo a cabo, pero éste es sólo el primer obstáculo a lo largo del ciclo de vida del proyecto, conforme transcurra la planeación se presentarán preguntas como ¿con qué herramientas se va a realizar el desarrollo?, ¿la opción seleccionada es la que mejor se ajusta a los requerimientos del sistema?, ¿El software a emplear será software propietario o software libre?

Lo más recomendable es establecer un proceso de aprendizaje de aproximadamente 2 ½ a 3 semanas para leer la documentación y realizar los ejemplos disponibles en la página de Hibernate y otros que pueden encontrarse fácilmente. De igual forma realizar acercamientos relacionados con el servidor y base de datos seleccionados para poner en producción la aplicación. Es imperativo entender que Hibernate puede necesitar adecuaciones con respecto al manejador seleccionado y que no todas las arquitecturas de servidor reaccionan igual, por mucho que se quiera confiar en la llamada portabilidad de Java.

4.5.2 Validación de archivos.

Si se requiere explotar información contenida en un repositorio lo primero que se necesita es que esta sea accesible, de poco sirve que los datos se encuentren en forma impresa para un sistema que funciona en Web, aún si se contara con esa información almacenada en medios electrónicos el sistema requiere que esos registros formen parte de la base de datos dentro de una estructura comprensible para cargar la información, esto se logra de varias formas una de ellas al contar con la información antes de iniciar la operación formal del sistema (información precargada vía sentencias SQL), la otra manera de trabajar con esta información bien se puede dividir en los siguientes 2 casos:

Carga de registros como funcionalidad del sistema. Es cuando una vez que el sistema entra en operación se requieren ingresar nuevos datos, pero se trata de una cantidad considerable de los mismos y resulta poco útil emplear el módulo de carga individual que proporciona el sistema.

Descarga de información como funcionalidad. Sucede cuando se necesita transportar información ya almacenada en la base de datos, es la operación inversa a la carga de modo que es primordial que esta información sea reutilizable, permitiendo que pueda volver a cargarse, con el módulo de carga, en la misma base de datos o en otra.

Lo importante es que se definan las similitudes y dependencias entre estos tres diferentes procesos de forma que operen de forma complementaria y la realización de uno no dificulte el funcionamiento de algún otro. Obviamente la precarga de información es el primer paso para la aplicación y ésta es casi idéntica a la carga de registros salvo algunos detalles, la descarga de datos sería el módulo que probaría la funcionalidad de los otros 2 procesos y el propio. Lo primero es determinar los diversos factores que afectan el proceso de carga y descarga para así poder ponderar la importancia de cada uno y el algoritmo de solución más adecuado. El siguiente listado muestra cuáles son los más importantes y cómo se utilizan para generar la plantilla de validación que será el documento que contenga el orden y los procedimientos básicos para la carga y descarga de información.

4.5.3Estructura básica.

Son valores que definen la estructura de la plantilla de validación.

- El número de datos. Deben ser idéntico al número de atributos definidos en las tablas que se quiere cargar.
- El orden de los datos. lo más recomendable es que se mantenga el mismo orden en que están definidos los atributos en cada tabla.
- Definir un separador por atributo. Indica donde termina un dato y comienza el otro. Por ejemplo ‘@|’

- Definir un separador por renglón. Indica donde termina un registro completo y comienza el siguiente. Por ejemplo el símbolo de salto de línea, Java lo determina en forma automática.

Esta sería una primera aproximación de la plantilla de validación, este puede ser un documento de texto plano, un archivo XML o estar representado en registros de la base de datos. El primer elemento definido correspondería al número de datos en la tabla y los siguientes los tipos de datos por registro.

```
4@|Integer@|String@|Character@|Date@|
```

Una plantilla de carga adecuada podría ser esta:

```
100@|Alberto Gómez@|A@|18/05/1976@|
101@|Alfonso mora @|A@|15/01/1980@|
100@|Ana Beltrán@|B@|20/12/1969@|
```

Una vez dado este primer paso se integra la validación del contenido del archivo. Los criterios a tomar en cuenta, en un escenario ideal, deben ser:

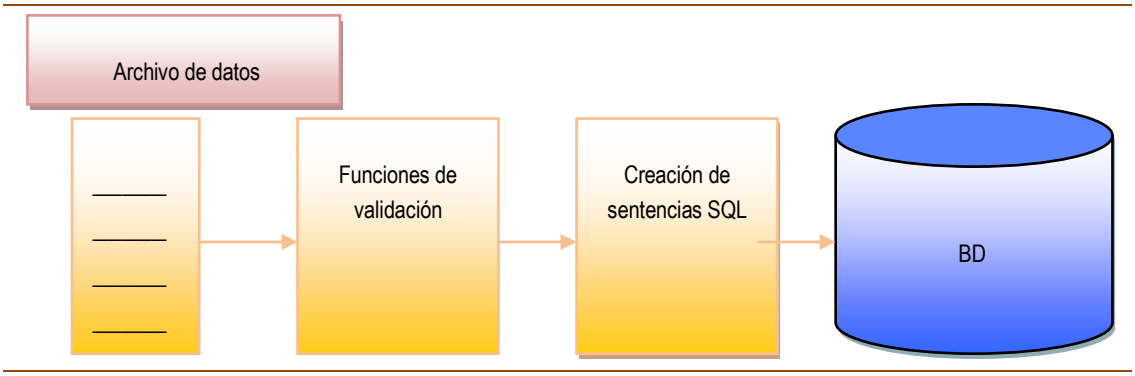


Figura 13 Secuencia del proceso de validación

4.5.4 Criterios generales.

Definen los criterios más importantes en cuanto a las tablas de la base de datos y su relación con tipos definidos de Java.

- La obligatoriedad del atributo. Puede ser o no nulo.
- El número máximo de caracteres que puede almacenar.
- Valores por default
- Listas con valores predefinidos. Por ejemplo en un campo que almacene niveles y solo acepte los valores A, B, C y D.

Identificación de llaves primarias

- Debe ser del tipo especificado. Integer y String son los más comunes.
- No debe estar repetida. Cuando ya se encuentra en la base de datos.
- No debe intentar ingresarse varias veces, sucede cuando una transacción posterior dentro del mismo archivo desea ingresar esa misma clave.
- Los nombres no pueden contener números o caracteres especiales.
- Las claves que pertenecen a otras tablas deben de ser válidas. Los valores existen en esa tabla a la que se hace referencia.
- También deben de respetar el formato especificado en esa tabla.
- Si es el caso no debe repetirse el valor a insertar.

Atributos numéricos

- No pueden contener caracteres que no sean números
- El valor debe ser coherente con el objetivo del atributo. Si se almacena la edad de una persona no pueden ser valores negativos o contener decimales.

Atributos alfanuméricos

- El valor debe ser coherente con el objetivo del atributo ya que este determina que caracteres son válidos y cuales no.

Atributos de tipo fecha

- No puede contener caracteres que no sean números o el carácter definido como separador de fecha. Es importante definir el formato de fecha que se utilizará en todos los procesos.

La plantilla quedaría de la siguiente forma.

```
4@|
Integer@|String@|Character@|Date@|
NO@|NO@|SI@|NO@|
4@|30@|1@|10@|
```

Ahora el primer renglón solo almacena el número de datos en la tabla, el segundo los tipos de datos ya conocido se ha añadido un tercer renglón que indica que registros no pueden ser nulos y cuales si; y un cuarto renglón con el máximo número de caracteres permitido por registro.

Hasta este momento queda claro que este archivo por lo menos debe recorrerse en 4 ocasiones y que el contenido debe almacenarse en arreglos de datos para validar todo el archivo. Ya se mencionó que debe validarse el contenido de acuerdo a cada tipo de dato pero aún no se especifica que proceso debe seguirse para garantizar cada valor. Una solución es utilizar métodos de Java definidos en las clases proporcionadas en la aplicación o generar nuevos métodos que cumplan con estos requerimientos cambiando la plantilla de datos así:

```
4@|
Integer@|String@|Character@|Date@|
NO@|NO@|SI@|NO@|
4@|30@|1@|10@|
validarNumeros@|ValidarCdenas@|validarCadenas@|validarFechas@|
```

Se ha agregado un nombre una función de Java definida por el usuario que validará el contenido de cada atributo en la plantilla de datos. Como último control se agrega un renglón para validar el rango de valores que un registro puede permitir mencionado anteriormente que en este caso hace referencia a un arreglo de datos llamado arrValoresNivel.

```
4@|
Integer@|String@|Character@|Date@|
NO@|NO@|SI@|NO@|
4@|30@|1@|10@|
validarNumeros@|ValidarCdenas@|validarCadenas@|validarFechas@|
@|@|arrValoresNivel@|@|
```

Criterios de recorrido. Corresponde ahora determinar los criterios para efectuar el recorrido de la plantilla de datos. Este proceso depende de las necesidades del usuario y a partir de estas varía el consumo de recursos totales del sistema. Se listan los criterios que pueden determinar el algoritmo de validación deseado.

- ***Criticidad del proceso.*** Se refiere a que tan imprescindible es la calidad de los datos antes que cualquier otra cosa, a mayor criticidad mayor número de controles para asegurar este objetivo.
- ***Validación de control e inserción de datos***. En control se refiere a verificar que cada atributo sea consistente con la plantilla de control mientras que en inserción se verifica que por un error en el manejador de bases de datos un registro no se pueda insertar. El orden de ejecución de ambos procesos puede ser primero la validación de la plantilla de datos y después la validación de la inserción. Otra opción es realizar las dos validaciones por cada registro, validar cada renglón y si es correcto verificar la inserción en la tabla.
- ***Interrupción del proceso de carga.*** Existen dos opciones en caso de que la plantilla de datos contenga errores. El primero es enviar un mensaje de error indicando el número de renglón y la inconsistencia encontrada por el sistema. La segunda opción es recorrer el archivo en su totalidad y al final mostrar al usuario el listado de renglones inconsistentes y sus respectivas causas. En el primer caso el tiempo es evidentemente menor pero no se garantiza que al corregir el archivo no existan mas inconsistencias posteriores a el punto donde se verificó mientras que en el segundo caso siempre tendrá que esperarse a que el sistema revise el archivo en su totalidad.
- ***Procedimiento en caso de error.*** De acuerdo a la criticidad del proceso al ocurrir un fallo en la plantilla de carga puede optarse por parar el proceso de carga y mantener el número de registros en la tabla hasta que ocurrió el error o no guardar ningún registro a menos que todos sean consistentes. Este es uno de los aspectos más importante en la carga de información y existen múltiples de combinaciones, por ejemplo guardar cada cierto número de registros y abortar todos los datos del bloque que contenga errores, el usuario debe estar consciente de que es realmente importante para que la implementación no se convierta en un cuello de botella para la aplicación.
- ***Validación de las reglas de negocio del sistema.*** Hasta este momento no se hecho mención de este tipo de validación. Se refiere a las reglas de la aplicación que van mas allá de la consistencia entre las tablas de la base de datos. Un ejemplo sería el nivel del usuario (A, B, C o D). “Los usuarios menores a 25 años no pueden ser nivel A”. En este caso la plantilla de datos no podría contener registros con usuarios de menos de esta edad y ser al mismo tiempo de nivel A.
- Algoritmo de validación
  - Validar toda la plantilla de datos.
  - En caso de éxito.
  - Validar todas las inserciones.
  - En caso de éxito
  - Terminar el proceso.
  - En caso de error.
  - Se realiza la inserción de cada registro, el registro erróneo no se registra quedando solo los registros consistentes en la tabla.
  - No se guarda ningún registro y se indica en que línea se presento el error.
  - En caso de error.
  - Mostrar él o los errores de la plantilla. Si se opto por Validar todos los renglones de la plantilla.

4.5.5 Observaciones.

Se puede optar también por insertar por bloques de datos. Por ejemplo, si 100 registros son válidos para los dos filtros del algoritmo se insertan esos 100 registros. Si uno de esos registros produjo un error y no se llego a 100 no se guarda ningún registro y se indica en que línea del bloque no se llevó a cabo la inserción.

Si se validó toda la plantilla y hubo un error en la inserción es conveniente almacenar en la sesión de la aplicación que la plantilla no tiene errores para que no se valide nuevamente la información. Volver a validar 300,000 registrosya comprobados es absurdo para una aplicación decente.

- Validar un bloque de datos.
- En caso de éxito.
- Validar todas las inserciones.
- En caso de éxito
- Terminar el proceso.
- En caso de error.
- Se realiza la inserción de cada registro, el registro erróneo no se registra quedando solo los registros consistentes en la tabla.
- No se guarda ningún registro y se indica en que línea se presento el error.
- Si Existe otro bloque de información: repetir el proceso.
- En caso de error.
- Mostrar él o los errores de la plantilla e indicar el bloque en el que ocurrió el error.

Evidentemente existe un límite en la cantidad de registros que puede manejar un sistema. Depende mucho de los recursos del sistema y por ningún motivo debe tratarse el problema como una ecuación lineal. Si 2,000 datos tardan 5 minutos en subir 4,000 datos tardaran 10 minutos. Apegarse a esta conclusión es muy aventurado pues evidentemente es errónea ya que el comportamiento del servidor no es lineal conforme se incrementa la información sino que tiende a ser exponencial, en algunas pruebas el tiempo se incrementaba hasta un 400% en lugar del 100% estimado. Varios eran los factores que generaban este comportamiento por ejemplo, entre más tiempo transcurría la validación de la información se hacía mas tardada (los recursos de memoria no se liberaban de manera inmediata) y el proceso terminaba siendo visiblemente ineficiente.

En Java existe un mecanismo para eliminar los recursos que ya no se utilizan (Garbage Collector) que no puede forzarse, solo sugerir su ejecución, y solo contempla casos muy específicos para liberar memoria. Esta es una de las principales causas que dieron motivo a un segundo proyecto con

Hibernate para este sistema, que no era más que la optimización de procesos que eran ineficientes al no poder manejar el volumen de información necesario para cada uno de ellos.

En cuanto el número de registros empleados superó con creces todas las expectativas planteadas en la solución propuesta inicialmente, la cuál perdió toda efectividad. El hecho es que se incluyeron nuevos grupos de usuarios que no estaban contemplados en un principio así como procesos que requerían mucha más información que, obviamente, debía cargarse en la base de datos. Todos estos factores junto con un nuevo crecimiento de registros por año hacía inviable mantener esta solución. Si bien se realizaron cambios en la metodología y el tiempo de carga una vez que se probó con grandes volúmenes se redujo a la mitad se determinó utilizar PERL y validaciones vía procedimientos almacenados en la base de datos. Se consiguieron tiempos mucho más satisfactorios pero se agregó un nuevo elemento a la lista de herramientas por aprender a manejar por parte del cliente que además debía emplearse en modo consola y no se trata de una interfaz amigable e intuitiva.

En cuanto a Hibernate aún queda por probar un gestionador de archivos que divida el archivo original en pequeños archivos del tamaño que se comprobó puede ser controlado por la aplicación sin retrasos exponenciales y lanzarlos uno detrás de otro para su validación. Se corroboró que no se llegaba al punto en el que el sistema empezaba a consumir recursos en forma desmedida debido al descanso entre la carga de archivos pero no se aplicó en forma directa al sistema en producción. Una solución alternativa es cambiar las consultas por unas mas cercanas al lenguaje SQL nativo que son mucho más rápidas para procedimientos en los que no se necesita manipular objetos sino datos en forma directa.

4.5.6 Optimización del proceso.

Una alternativa muy recomendable después de trabajar con otros proyectos más que empleaban Hibernate o simplemente el proceso de carga y validación de datos permiten realizar las siguientes observaciones.

Ejecución de procesos en segundo plano. De acuerdo a la criticidad del proceso al ocurrir un fallo en la plantilla de carga puede optarse por parar el proceso de carga y mantener el número de registros en la tabla hasta que ocurrió el error o no guardar ningún registro a menos que

- ***Monitoreo.*** De acuerdo a la criticidad del proceso al ocurrir un fallo en la plantilla de carga puede optarse por parar el proceso de carga y mantener el número de registros en la tabla hasta que ocurrió el error o no guardar ningún registro a menos que

- ***Ejecución de Procesos simultáneos.*** De acuerdo a la criticidad del proceso al ocurrir un fallo en la plantilla de carga puede optarse por parar el proceso de carga y mantener el número de registros en la tabla hasta que ocurrió el error o no guardar ningún registro a menos que

4.6 Reportes

Un reporte es un documento que hace uso de registros de la base de datos y presenta la información en la forma más útil posible. Se deben tomar en cuenta varios aspectos como:

- El número de datos que utilizará el reporte.
- Las operaciones aritméticas que puede utilizar el reporte.
- La cantidad de registros a mostrar en pantalla.
- El número de tablas involucradas en el reporte.

Tomando como referencia al primer sistema se sabe que éste demoraba aproximadamente 10 minutos en el despliegue de 5,000 registros correspondientes a un reporte. Esta información contempla a 5 usuarios de forma concurrente. Tomando en cuenta que de acuerdo con una posterior proyección, el número de usuarios aumentaría en un 400% [8], se debieron tomar acciones encaminadas a buscar una forma más eficiente de realizar dichas operaciones. Estas acciones estaban relacionadas con la capa de datos, la capa de negocio y/o la capa de presentación.

Si una tabla contiene 100,000 registros y existe un reporte que necesite mostrar información acerca de todos esos registros es evidente que se exigirá en gran medida del manejador de la base de datos, pero un buen manejador debería controlar esta información sin ver afectado su desempeño en forma seria, aún si se requiere hacer uso de operaciones aritméticas por cada registro, renglón, columna y los valores totales el desempeño debe seguir siendo controlable. El problema radica más bien en el manejo de la información en pantalla.

Evidentemente no se requiere de objetos, complejos o sencillos, para realizar un reporte, basta con realizar las consultas necesarias así como establecer las relaciones entre tablas de manera directa en la sentencia, una vez hecho esto sólo hace falta acomodar los datos en el orden que está especificado en cada reporte.

8 Información proporcionada por el cliente.

Para utilizar Hibernate se puede recurrir a un tipo de consultas llamadas *"consultas directas"*, que no convierten los registros resultantes en objetos sino en arreglos. El proceso consiste en generar una colección con tantos registros como se haya especificado en la consulta, cada registro debe tener un tipo correspondiente en Java y ser manejado como tal Integer, String, Bolean, etc. Este tipo de consulta es mucho más rápida porque no genera los objetos normales por cada tabla, también llamados Pojo's, ni los objetos de todas las tablas a los que estén relacionadas. Son Querys más similares a una sentencia de SQL estándar, pueden identificarse porque existe código en la sentencia antes de la palabra clave FROM, un HQL normal una sentencia tiene que empezar con la palabra FROM seguida del resto de la sentencia y se intuye que regresa objetos del tipo relacionado con el nombre de la tabla, por ejemplo:

FROM registro;

Una sentencia directa es del tipo

Se recomienda que las sentencias sean lo más óptimas posibles para evitar que Hibernate haga un "trabajo extra" al implementar una sentencia cuya sintaxis sea demasiado obscura. Por ejemplo, si se emplean sentencias anidadas mal estructuradas Hibernate no realizará una sola consulta, lo que el desarrollador podría provocar es que se realizara una consulta principal y una sub-consulta por cada sentencia anidada dentro de esa sentencia, tantas consultas adicionales como registros produzca la primer consulta. Si es una sentencia HQL ya se ha mencionado la sobrecarga de objetos que esto puede generar.

4.6.1 Optimización del proceso.

No se puede descartar el no uso de Hibernate cuando el volumen de información es muy grande, si bien existen múltiples opciones con las cuales hacer consultas más eficientes, realizarlas requiere un nivel de conocimiento más elevado, por lo tanto si se tiene muy poco tiempo o se cuenta con un equipo de desarrollo inexperto bien valdría contemplar que secciones conviene tratar sin Hibernate para evitar retrasos.

4.6.2Generalidades de la capa de persistencia.

Si bien Hibernate permite resolver problemas empleando consultas directas elaboradas con la sintaxis adecuada existen ciertas precauciones que deben tomase si se presentan situaciones como las ocurridas a lo largo de los tres proyectos de desarrollo mencionados.

Llevar un seguimiento de la información contenida en los reportes. Es común que los reportes de la aplicación formen parte de la última etapa del proyecto, pues estos explotan la información generada por el resto de la aplicación, sin embargo al existir incontables cambios en la estructura de la base de datos, las reglas de negocio, volumen de información entre otros, es inevitable que los reportes serán afectados por estos cambios y al momento de llegar a trabajar con ellos representen una complejidad mayor de la que se planteó al inicio del proyecto.

Uno de os cambios que puede afectar el rendimiento del reporte es el número de operaciones que pueden realizarse una ves emitido el reporte. Por ejemplo. Algunos de los requerimientos del primer y segundo proyecto con Hibernate era recorrer la información presentada en el JSP, generar una versión imprimible con esa información y la generación de un archivo en formato PDF para almacenar en disco. La consecuencia de todos estos procesos que podían realizarse uno después de otro o bien solo efectuarse la generación del primer reporte era la necesidad de almacenar toda esa información en la sesión de sistema para poder utilizarla en cualquiera de las opciones que tenía el cliente sin tener que recurrir nuevamente a la base de datos. En cuestión de rendimiento no era lo más óptimo, pero tantas opciones con respecto a tanta información en un solo momento no dejaban muchas alternativas disponibles.

El servidor. El tipo de Software y Hardware se determina incluso antes de empezar el desarrollo. Se deben conocer las prestaciones del mismo. Estas prestaciones ya se han mencionado pero existen otros aspectos que deben considerarse pues su impacto afecta el rendimiento del servidor con lña palicación.

El servidor compartido. Algunas empresas utilizan un mismo servidor para cargar varias aplicaciones, esta manera de trabajar no se puede pasar por alto, muchas veces se estima el rendimiento.

4.6.3 Generación de documentos PDF.

El objetivo del módulo consiste en elaborar documentos en formato .pdf con la información contenida en la base de datos a través de un reporte específico o una consulta compuesta con diversos valores.

Se busca contar con un respaldo organizado y portable de la información ya sea en formato electrónico o en forma impresa sin tener que ingresar al sistema y repetir el proceso que elabora el mismo reporte.

Estos documentos requieren generalmente toda la información de un número variable de tablas que conforman uno o varios módulos del sistema, la menor de las veces solo unos cuantos datos. En ambos casos emplear Hibernate es una muy buena opción que agiliza el tiempo de desarrollo en este proceso compensando el retraso que podría presentarse si se desconoce como funcionan las librerías que se encargan de generar, estructurar y escribir información de Java en un documento pdf. Si el requerimiento especifica generar documentos con poca información pueden ser solo títulos, links y direcciones el proceso podría ser el siguiente.

- Generar objeto(s) que contenga(n) los tipos de datos Java que requiere el documento.
- Realizar las consultas necesarias y almacenar e resultado en los objetos ya generados.
- Codificar los manejadores adecuados a la estructura del documento que se requiere. Uno que gestione la apertura, escritura, configuración y cierre del documento que sería general para todos los documentos, Flujo del archivo pdf. Y otro que ordene las consultas según las especificaciones del documento con ayuda de la clase de acceso al documento .PDF.

El proceso de documentos pdf es muy similar, seguramente se hará uso de las clases ya definidas por Hibernate para cada tabla y se requiera generar algunos nuevos objetos para completar el documento pero básicamente se mantiene el mismo proceso de creación de archivos pdf.

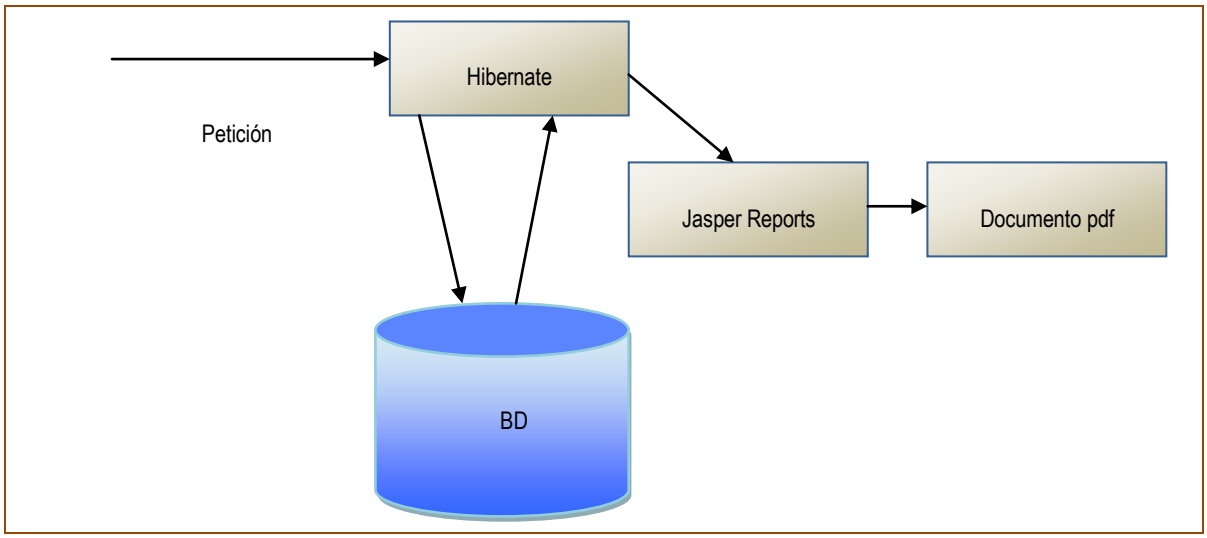


Figura 14. El proceso de validación

4.6.4 Paginación de múltiples registros.

Cuando se realiza una búsqueda o se pretende realizar un reporte específico del sistema el resultado final puede involucrar una cantidad de registros muy difíciles de manejar, tal cantidad de información no puede mostrarse al mismo tiempo en la pantalla, debido a que no es razonable recorrer los cientos de páginas para encontrar un dato y es prácticamente imposible realizar una interpretación útil en esas condiciones.

Implementar un paginador requiere que se identifiquen los elementos que provocan que el resultado de las búsquedas y reportes sea lo suficientemente grande para requerir uno. Las causas más comunes son las consultas elaboradas con muy pocos criterios de búsqueda. Éstas ocurren cuando la información disponible para la búsqueda es reducida o cuando la búsqueda se hace con información muy general (buscar un dato de acuerdo a una fecha determinada).

El siguiente paso es convertir la información para mostrarla en pantalla, los registros son formateados, se transforman en código HTML o XML usando tablas, frames, identaciones,etc.

Realizar operaciones como sumatorias y cálculos más complejos según se haya especificado en el reporte. Estas operaciones se ejecutan en el servidor para mostrar un archivo XML final que debe ser mandado a través de la red a la máquina cliente que solicitó la consulta y generarse en el navegador cliente. El tiempo sumado de estos procesos conforman el rendimiento de la aplicación que debe ser el más óptimo posible. Las características que un buen paginador como solución debe tener son:

- Que las pantallas no sean muy largas y contengan demasiados registros evitando recorrer la pantalla para buscar un dato específico y el usuario no lo distinga entre el resto de los datos.
- El tiempo de espera no puede ser excesivo pues desespera y/o distrae al usuario, en cuanto los registros aparecen en pantalla este ya no esta dispuesto para encontrar dicha información.
- El sistema debe proporcionar una versión imprimible de la consulta, esta versión necesita la información de la primer consulta (o es la misma consulta) para generar el archivo imprimible.
- Evitar cargar información sin la certeza de que esta será empleada.
- El tiempo de impresión de este reporte depende de la velocidad de la impresora no del reporte una vez impreso no se garantiza que sea más fácil encontrar un registro en específico.

4.6.5 Algoritmo de paginación.

Se extraen bloques de información de la base de datos, si éstos son los que requiere el usuario ya no se requiere explotar más información si no son útiles el usuario invoca a un nuevo bloque de datos (mediante una liga)repitiendo el proceso de verificación.

- El reporte se limita al grupo de páginas que aparecen en la pantalla. Por ejemplo cada pantalla muestra 20 bloques con 80 registros cada uno.
- Al pie de la página también se muestran las siguientes ligas
- Anterior si existen registros antes de la pantalla actual
- Siguiente Si existen registros depués de la pantalla actual.

Se realiza la consulta a la base de datos, para este momento ya están predefinidos el número de bloques por pantalla y el número de registros por bloque. La consulta trae el número de registros correspondientes, esta información se almacena en la base de datos, en pantalla se muestran tantos links como número de bloques se específico. Ese número representa el índice a partir del cual se buscará en la base de datos si se requiere esa consulta que estará compuesto por n registros siendo

n= número de bloques * número de registros por bloques.

No se realizará otra consulta a la base de datos mientras no se haga uso de los botones de anterior y siguiente.

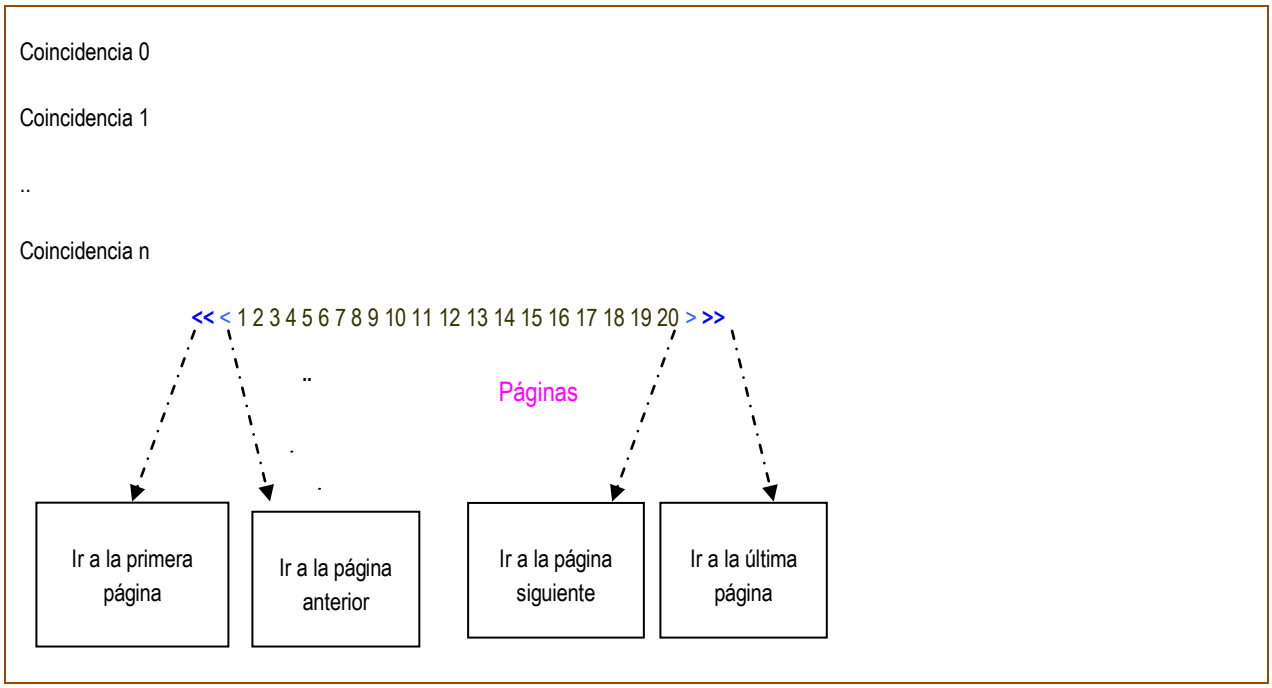


Figura 15. Estructura del paginador.

La generación de reportes en PDF tiene una característica similar a la de las consultas utilizadas en el CGC y expedientes, ya que cuando se genera uno de éstos, se efectúa una consulta en la que se extraen todos los elementos necesarios para la generación del PDF, aquí uno de los problemas es la forma en que se está construyendo, ya que utilizando una herramienta de bajo nivel como los es iText, consume recursos que se traduce en un tiempo de espera del usuario.

Para resolver esta situación se realizó una investigación de otras herramientas que ayudaran a generar este tipo de reportes sin que se consumieran muchos recursos. A continuación se muestra una tabla con distintas herramientas evaluadas para dicha tarea:

Nombre de la Herramienta	Costo		Archivos de Salida
	Libre	Propietaria	
JasperReports	x		HTML, PDF, XLS, XML y CSV
DataVision	x		HTML, XML, PDF, LaTeX2e, DocBook
Formula One e.Report Engine		x	PDF, XML, DHTML, CSV, HTML
Big Faceless PDF Library		x	PDF
JFreeReport	x		PDF, HTML, CSV, Excel, plain text
Crystal Reports (developer)		x	PDF, Excel
RReport Visual Builder		x	HTML , PDF , XML, Text
espressreport		x	PDF, HTML, CSV, Excel, plain text
ordesigner (OpenReports Designer)	x		PDF, HTML, XLS, CSV
ElegantJ DataReports		x	CSV, HTML, XML,PDF
VisualSoft FlexiReports		x	HTML, DHTML,CSV
iReporter	x		

Tabla 3-1 Comparativa de herramientas PDF 2001

Dadas las características anteriores, y ponderando entre la usabilidad, costo y tiempo de desarrollo, JasperReports fue la herramienta más adecuada para la generación de reportes PDF, destacando que es una herramienta libre, se pueden generar reportes no sólo en PDF, sino en otros formatos, lo cual le da un valor agregado a futuro porque da las posibilidades de cambiar el formato de los reportes. Adicionalmente, cabe mencionar, que muchas herramientas lo utilizan como su motor (core) para la creación de reportes.

En las pruebas realizadas en cuestión de tiempos, se tiene la siguiente tabla:

Registros	*SQLPlus*	*iReport-JasperReport*
1,000	*0.235*	*0.095*
2,000	*0.268*	*0.156*
5,000	*1.135*	*0.356*
7,500	*0.550*	*0.498*
10,000	*0.718*	*0.643*
15,000	*5.818*	*0.938*
20,000	*5.083*	*1.256*
30,000	*5.152*	*1.858*
50,000	*4.201*	*3.023*
100,000	*7.151*	*6.631*
167,291	*13.617*	*10.356*

Tabla 3-2 Tiempo de Respuesta en minutos

Estas pruebas fueron realizadas bajo las siguientes condiciones:

La prueba comprende el tiempo de obtener y mostrar los datos.

Consulta sencilla con un agrupamiento

Máximo de 100 objetos en memoria.

Nota: La tabla anterior no incluye los tiempos de latencia por uso de red.

4.6.6 Hibern8

Esta herramienta es muy útil para realizar sentencias tanto HQL como SQL estándar, entre sus características más importantes se pueden mencionar.

Para configurar Hibern8 la aplicación solicita el archivo de configuración, si llegara a existir un error en dicho archivo Hibern8 manda un mensaje indicando el error detectado. Si no se contara con esta herramienta el framework y las librerías de Hibernate no son tan espefícficas con respecto a estos mensajes de error.

Nombres de clases, tablas, identificadores y campos. Hibern8 muestra todos estos atributos en una zona ubicada del lado izquierdo de la aplicación con los nombres tal como deben ser utilizados para armar las sentencias de la base de datos. Los usuarios inexpertos se pueden familiarizar de manera sencilla de estructurar sentencias, además todas aquellas tablas, que debido a la direccionalidad no se puedan acceder de manera directa, no se muestran como atributos de las tablas, de esta manera el usuario identifica que hay una restricción de acceso y ya no pasa horas tratando de entender por qué no funciona su sentencia y busque otras alternativas para realizar esas sentencias.

Basta con hacer clic sobre cada uno de los elementos del árbol de tablas de Hibern8 para que se genere automáticamente la sentencia con la que se puede acceder a esos datos.

Las sentencias resultantes no son sólo tablas con la información del query ejecutado, es una estructura anidada que nos permite seleccionar cualquier registro, automáticamente se abre una ventana con la información del mismo y si se selecciona una id o llave foránea se muestra en esa misma pantalla como está compuesta la id o el registro de la tabla foránea correspondiente a esa clave, y si existen más relaciones se muestran todas de esta manera.

Es la mejor forma de probar sentencias SQL para después agregarla a la aplicación Java.

5 Conclusiones

El desarrollo de aplicaciones tiene intrínsecamente ventajas y desventajas desde la selección de la plataforma de desarrollo, el análisis, el proceso de desarrollo y los tiempos establecidos. Aplicar como mecanismo de solución nuevas tecnologías implica una disminución en los tiempos y cargas de trabajo en alguna de las etapas del ciclo de vida del proyecto, pero evidentemente, este beneficio tendrá un costo en cuanto al aprendizaje y experimentación que se verá reflejado en otra etapa del proyecto. Este último capítulo resume las ventajas y desventajas más importantes de emplear Hibernate como capa de persistencia de una aplicación Web.

Hibernate no es software "Instálese y úsese". Requiere tiempo y esfuerzo conocer cómo funciona el núcleo de Hibernate y todavía más para realizar cambios y optimizaciones verdaderamente funcionales. Antes de instalar Hibernate es recomendable:

Establecer tiempos razonables de capacitación para el equipo de desarrollo para atacar de manera eficaz el desarrollo de la aplicación. Se requiere tiempo, espacio y mantener al equipo de desarrollo ocupado en ese periodo de tiempo pero es una decisión que ofrecerá muchas ventajas en ese y otros proyectos posteriores.

Conocer por lo menos a nivel intermedio las herramientas con las que trabaja Hibernate como XML y Ant. El primero representa cómo la base de datos se comunicará con los objetos mientras que Ant puede crear la estructura de archivos con todos los objetos, mapeos, relaciones y métodos para interactuar con la base de datos. Pero ninguno asegura que el resultado final sea el más adecuado, eso depende de los requerimientos del sistema y esos requerimientos no pueden ser interpretados ni por XML ni por Ant o cualquier otro software que se utilice para reducir tiempo escribiendo líneas de código. Se necesita conocer como trabaja cada herramienta para adecuarla a cada requerimiento, agregar o quitar líneas de código o incluso, modificar o agregar archivos completos. Desconocer como funciona este software acercará más a la aplicación a una instalación "por default" que ser un producto adecuado a lo que el cliente solicitó.

Para los procesos de validación de archivos Hibernate resulta poco eficiente debido a que la estructura de los datos se encuentra formada por caracteres especiales e información almacenados en un archivo de texto plano. No son registros de la base de datos ni objetos de Java quedando fuera

del alcance de Hibernate hasta que la aplicación los convierta en registros u objetos. Para ello se requiere repetir funcionalidad de Hibernate que sirva de puente entre el archivo y la capa de persistencia. Haciendo inviable esta solución a menos que los archivos sean de tamaño reducido y la cantidad de datos por renglón fáciles de convertir a objetos.

Los reportes de aplicación son viables si se utilizan consultas naturales en lugar de HQL, se obtiene la información precisa en forma rápida y sin tener que salir de Hibernate para ello.

La conversión a archivos PDF es uno de los procesos más exitosos en cuanto el empleo de Hibernate, al extraer los objetos directamente de la base de datos agrupados de manera que pueda colocarse la información fácilmente y poder acceder a objetos relacionados por medio de la navegación entre objetos de Hibernate son las ventajas principales de esta solución. La limitante más destacada es el volumen de información excesivo para generar el archivo PDF y los casos en los que el número de registros a utilizar sea muy reducido y disperso para extraer objetos completos con valores que no serán útiles.

Finalmente está el factor humano. Decisiones apresuradas o sin el sustento adecuado pueden convertir una herramienta "mágica" en un lastre durante todo el proyecto aunque no todo lo negativo es producto de simple acto de negligencia. Aún actuando con precaución y realizando procesos bien definidos se corre el riesgo de optar por una solución que a corto o mediano plazo represente un cuello de botella para la aplicación. La experiencia con varios productos de desarrollo será entonces la mejor guía para tomar decisiones adecuadas en cuanto a cada problema que se presente.

6 Anexos

6.1 Listado de archivos disponibles en una aplicación convencional.

6.1.1Archivos tipo abstract

- AbstractEntidad.java
- AbstractMunicipio.java
- AbstractPermiso.java
- AbstractPermisoId.java –Para llave comnpuesta -
- AbstractRol
- AbstractSucursal
- AbstractUsuario
- AbstractUsuario

6.1.2Archivos tipo entity

- Entidad.java
- Municipio.java
- Permiso.java
- PermisoUsuarioId.java
- Rol.java
- Sucursal.java
- Usuario.java

6.1.3Archivos de mapeo (hbm)

- Entidad.hbm.xml
- Municipio.hbm.xml
- Permiso.hbm.xml
- PermisoUsuarioId.hbm.xml
- Rol.hbm.xml
- Sucursal.hbm.xml
- Usuario.hbm.xml

6.1.4Archivos DAO

- EntidadDAO
- MunicipioDAO
- PermisoDAO
- PermisoUsuarioDAO
- RolDAO
- SucursalDAO
- UsuarioDAO

6.1.5 Archivos de configuración

- IBaseHibernateDAO
- hibernate.cfg

6.1.6 Hibernate.cfg.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<!-- Generated by MyEclipse Hibernate Tools.                   -->
<hibernate-configuration>

    <session-factory>
        <property name="connection.username">sare</property>
        <property                   name="connection.url">jdbc:informix-sqli://ruta
/tesis:informixserver=informix1_tcp</property>

        <property
name="connection.driver_class">com.informix.jdbc.IfxDriver</property>
      <property name="connection.password">password</property>

<!-- SQL dialect -->
      <property name="dialect">org.hibernate.dialect.HSQLDialect</property>
<!-- Enable Hibernate's automatic session context management -->
      <property name="current_session_context_class">thread</property>
<!-- Disable the second-level cache -->
      <property
name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
<!-- Echo all executed SQL to stdout -->
       <property name="show_sql">true</property>
       <!-- Drop and re-create the database schema on startup -->
        <!-- Sección I -->
      <property name="connection.autocommit">false</property>
```

```
        <mapping resource="mx/cofemer/sare/entity/EntidadFederativa.hbm.xml" />

        <mapping resource="mx/cofemer/sare/entity/Municipio.hbm.xml" />
        <mapping resource="mx/cofemer/sare/entity/Entidad.hbm.xml" />
        <mapping resource="mx/cofemer/sare/entity/Permiso.hbm.xml" />
        <mapping resource="mx/cofemer/sare/entity/PermisoUsuario.hbm.xml" />
        <mapping resource="mx/cofemer/sare/entity/Rol.hbm.xml" />
        <mapping resource="mx/cofemer/sare/entity/Sucursal.hbm.xml" />
        <mapping resource="mx/cofemer/sare/entity/Usuario.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

6.1.7 Script de creación de la base de datos (Postgres)

```
CREATE TABLE entidad (
        entClave              int NOT NULL,
        entNombre             varchar(50) NOT NULL,
        PRIMARY KEY (entClave)
);

CREATE TABLE municipio (
        munClave              int NOT NULL,
        munNombre             varchar(150) NOT NULL,
        entClave              int NOT NULL,
        PRIMARY KEY (munClave),
        FOREIGN KEY (entClave)
                               REFERENCES entidad
);

CREATE TABLE rol (
        rolClave              int NOT NULL,
        rolNombre             varchar(100) NOT NULL,
        PRIMARY KEY (rolClave)
);

CREATE TABLE sucursal (
        sucClave              int NOT NULL,
        sucDireccion          varchar(200) NOT NULL,
        sucNombre             varchar(100) NOT NULL,
        sucTelefono           varchar(50) NOT NULL,
        munClave              int NOT NULL,
        PRIMARY KEY (sucClave),
        FOREIGN KEY (munClave)
                               REFERENCES municipio
```

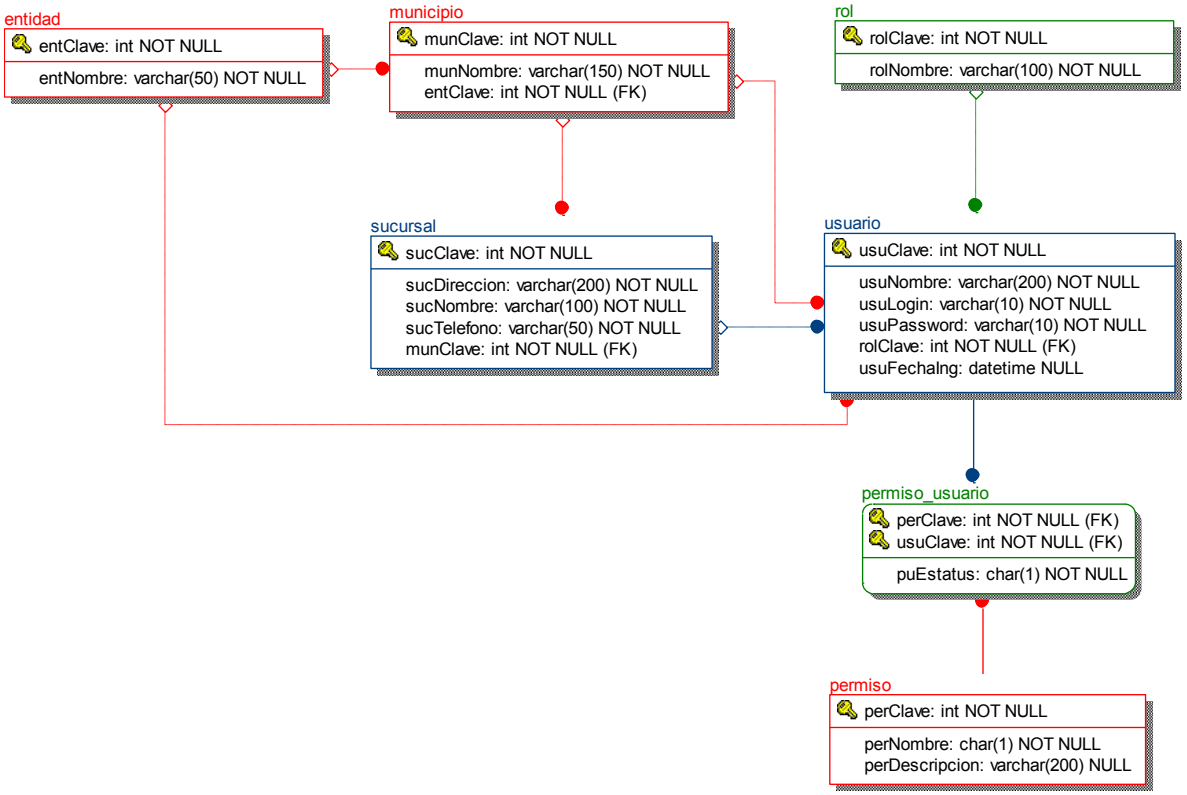
```
);

CREATE TABLE usuario (
       usuClave             int NOT NULL,
       usuNombre            varchar(200) NOT NULL,
       usuLogin             varchar(10) NOT NULL,
       usuPassword          varchar(10) NOT NULL,
       rolClave             int NOT NULL,
       usuFechaIng          timestamp NULL,
       sucClave             int NOT NULL,
       munClave             int NULL,
       entClave             int NULL,
       PRIMARY KEY (usuClave),
       FOREIGN KEY (entClave)
                            REFERENCES entidad,
       FOREIGN KEY (munClave)
                            REFERENCES municipio,
       FOREIGN KEY (rolClave)
                            REFERENCES rol,
       FOREIGN KEY (sucClave)
                            REFERENCES sucursal
);

CREATE TABLE permiso (
       perClave             int NOT NULL,
       perNombre            char(1) NOT NULL,
       perDescripcion       varchar(200) NULL,
       PRIMARY KEY (perClave)
);

CREATE TABLE permiso_usuario (
       perClave             int NOT NULL,
       usuClave             int NOT NULL,
       puEstatus            char(1) NOT NULL,
       PRIMARY KEY (perClave, usuClave),
       FOREIGN KEY (usuClave)
                            REFERENCES usuario,
       FOREIGN KEY (perClave)
                            REFERENCES permiso
);
```

6.1.8Diagrama Entidad Relación.



6.2 Glosario de términos

DTD *Document Type Definition*. Es una definición en un documento SGML ó XML que específica restricciones en la estructura del mismo. El DTD puede ser incluido dentro del archivo del documento, pero normalmente se almacena en un archivo ASCII de texto separado. La sintaxis de los DTDs para SGML y XML es similar pero no idéntica.

La definición de un DTD especifica la sintaxis de una aplicación de SGML o XML, que puede ser un estándar ampliamente utilizado como XHTML o una aplicación local.

Los DTDs son generalmente empleados para determinar la estructura de un documento XML o SGML. Un DTD describirá típicamente cada elemento admisible dentro del documento, los atributos posibles y (opcionalmente) los valores de atributo permitidos para cada elemento. Es más, describirá los anidamientos y ocurrencias de elementos. La mayoría de los DTDs se componen generalmente de definiciones de ELEMENT y definiciones de ATTLIST.

SQL. El Lenguaje de Consulta Estructurado (Structured Query Language) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Aún a características del álgebra y el cálculo relacional permitiendo lanzar consultas con el fin de recuperar información de interés de una base de datos, de forma sencilla.

JDBC. Es el acrónimo de Java Database Connectivity, un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema de operación donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice.

El API JDBC se presenta como una colección de interfaces Java y métodos de gestión de manejadores de conexión hacia cada modelo específico de base de datos. Un manejador de conexiones hacia un modelo de base de datos en particular es un conjunto de clases que implementan las interfaces Java y que utilizan los métodos de registro para declarar los tipos de localizadores a base de datos (URL) que pueden manejar. Para utilizar una base de datos particular, el usuario ejecuta su programa junto con la librería de conexión apropiada al modelo de su base de datos, y accede a ella estableciendo una conexión, para ello provee en localizador a la base de datos

y los parámetros de conexión específicos. A partir de allí puede realizar con cualquier tipo de tareas con la base de datos a las que tenga permiso: consultas, actualizaciones, creado modificado y borrado de tablas, ejecución de procedimientos almacenados en la base de datos, etc.

DAO. Emplear un Data Access Object DAO para abstraer y encapsular todos los accesos al almacenamiento persistente. El Data Access Object maneja las conexiones para guardar y recuperar datos.

ADO (ActiveX Data Objects) es uno de los mecanismos que usan los programas de computadoras para comunicarse con las bases de datos, darles órdenes y obtener resultados de ellas. Con ADO, un programa puede leer, insertar, editar, o borrar, la información contenida en diferentes áreas de almacenamiento dentro de la base de datos llamadas tablas. Además, se puede manipular la propia base de datos para crear nuevas áreas para el almacenamiento de información (tablas), como también alterar o eliminar las ya existentes, entre otras cosas.

El Número de identificación fiscal (NIF) es el sistema de identificación tributaria utilizada en España para las personas físicas (con documento nacional de identidad (DNI) o número de identificación de extranjero (NIE) asignados por el Ministerio del Interior).

OID. Es un acrónimo inglés que significa Object IDentifier (identificador de objeto).

HSQLDB es un sistema gestor de bases de datos libre escrito en Java. La suite ofimática OpenOffice.org lo incluye desde su versión 2.0 para dar soporte a la aplicación Base.

SGML. Son las siglas de "Standard Generalized Markup Language" o "Lenguaje de Marcación Generalizado". Consiste en un sistema para la organización y etiquetado de documentos. La Organización Internacional de Estándares (ISO) ha normalizado este lenguaje en 1986. El lenguaje SGML sirve para especificar las reglas de etiquetado de documentos y no impone en sí ningún conjunto de etiquetas en especial. El lenguaje HTML está definido en términos del SGML. XML es un nuevo estándar con una funcionalidad similar a la del SGML, aunque más sencillo y de creación posterior.

XHTML. Acrónimo inglés de eXtensible Hypertext Markup Language (lenguaje extensible de marcado de hipertexto), es el lenguaje de marcado pensado para sustituir a HTML como estándar para las páginas web. XHTML es la versión XML de HTML, por lo que tiene, básicamente, las mismas

funcionalidades, pero cumple las especificaciones, más estrictas, de XML. Su objetivo es avanzar en el proyecto del World Wide Web Consortium(W3C) de lograr una web semántica, donde la información, y la forma de presentarla estén claramente separadas. En este sentido, XHTML serviría únicamente para transmitir la información que contiene un documento, dejando para hojas de estilo (como las hojas de estilo en cascada) y JavaScript su aspecto y diseño en distintos medios (ordenadores, PDAs, teléfonos móviles, impresoras..etc.).

6.3 Índice de figuras

6.4 Índice de tablas

6.5 Referencias

http://tomcat.apache.org/

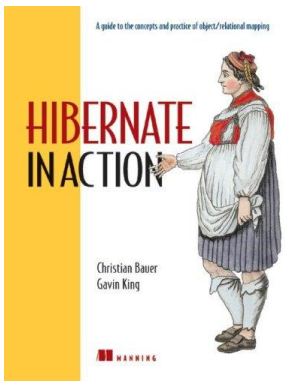
http://www.xam.dk/hibern8ide/

http://www.hibernate.org/

Java Persistence with Hibernate

Second Edition of Hibernate in Action

Christian Bauer and Gavin King



Hibernate in Action

Christian Bauer and Gavin King