UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN SISTEMAS COMPUTACIONALES

TRABAJO DE GRADO PREVIO LA OBTENCIÓN DEL TÍTULO DE

INGENIERO EN SISTEMAS COMPUTACIONALES

TEMA

ESTUDIO DE LA HERRAMIENTA DE DESARROLLO "JASPERREPORTS"

EN APLICACIONES WEB CON TECNOLOGÍA JSP

APLICATIVO

SISTEMA DE ADMINISTRACIÓN Y CONTROL DE RECURSOS TECNOLÓGICOS

PARA LA COOPERATIVA DE AHORRO Y CRÉDITO "ATUNTAQUI" LTDA.

AUTOR

WILSON ANÍBAL CÁRDENAS HERNÁNDEZ

DIRECTOR

ING. MARCELO JURADO

IBARRA – ECUADOR

2011

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLIZACIÓN
A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

La Universidad Técnica del Norte dentro del proyecto Repositorio Digital Institucional, determina la necesidad de disponer de textos completos en formato digital con la finalidad de apoyar los procesos de investigación, docencia y extensión de la Universidad.

Por medio del presente documento dejo sentada mi voluntad de participar en este proyecto, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	100309073-3		
APELLIDOS Y NOMBRES:	CÁRDENAS HERNÁNDEZ WILSON ANÍBAL		
DIRECCIÓN DOMICILIARIA:	IBARRA, PILANQUI BEV, PASAJE C CASA No. 1-72		
CORREO ELECTRÓNICO:	wcardenas1283@gmail.com		
TELÉFONO FIJO:	(06)2 612277	**TELÉFONO MÓVIL:**	093268628

DATOS DE LA OBRA		
TÍTULO:	ESTUDIO DE LA HERRAMIENTA DE DESARROLLO "JASPERREPORTS" EN APLICACIONES WEB CON TECNOLOGÍA JSP	
AUTOR:	WILSON ANÍBAL CÁRDENAS HERNÁNDEZ	
FECHA:	2011-11-01	
SOLO PARA TRABAJOS DE GRADO		
PROGRAMA:	[X] PREGRADO	[ ] POSGRADO
TÍTULO:	INGENIERO EN SISTEMAS COMPUTACIONALES	
DIRECTOR:	ING. MARCELO JURADO	

2. AUTORIZACIÓN DE USO A FAVOR DE LA UNIVERSIDAD

Yo, **Wilson Aníbal Cárdenas Hernández**, con cédula de identidad No. 100309073-3, en calidad de autor y titular de los derechos patrimoniales de la obra o trabajo de grado descrito anteriormente, hago entrega del ejemplar respectivo en formato digital y autorizo a la Universidad Técnica del Norte, la publicación de la obra en el Repositorio Digital Institucional y uso del archivo digital en la Biblioteca de la Universidad con fines académicos, para ampliar la disponibilidad del material y como apoyo la educación, investigación y extensión; en concordancia con la Ley de Educación Superior Artículo 143.

UNIVERSIDAD TÉCNICA DEL NORTE

CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE GRADO A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

Yo, **Wilson Aníbal Cárdenas Hernández**, con cédula de identidad No. 100309073-3, manifiesto mi voluntad de ceder a la Universidad Técnica del Norte los derechos patrimoniales consagrados en la Ley de Propiedad Intelectual del Ecuador, artículos 4, 5 y 6, en calidad de autor del trabajo de grado denominado *"Estudio de la herramienta de desarrollo JasperReports en aplicaciones web con tecnología JSP"*, que ha sido desarrollado para optar por el título de Ingeniero en Sistemas Computacionales en la Universidad Técnica del Norte, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente. En mi condición de autor me reservo los derechos morales de la obra antes citada. En concordancia suscribo este documento en el momento que hago entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Técnica del Norte.

Firma:

Nombre: Wilson Cárdenas

CI: 100309073-3

Ibarra, a los 8 días del mes de Noviembre de 2011

Ibarra, 03 de Octubre del 2011

Certificación

Por el presente certificamos, que el Señor WILSON ANÍBAL CÁRDENAS HERNÁNDEZ, egresado de la Universidad Técnica del Norte, Facultad de Ingeniería en Ciencias Aplicadas, Carrera de Ingeniería en Sistemas Computacionales, diseñó e implementó el SISTEMA DE ADMINISTRACIÓN Y CONTROL DE RECURSOS TECNOLÓGICOS, para la Cooperativa de Ahorro y Crédito "ATUNTAQUI" Ltda.; mismo que supero las pruebas de manera satisfactoria y se encuentra en producción a partir del 12 de septiembre del 2011 presentando un correcto funcionamiento. Además, cabe indicar que el sistema fue desarrollado respetando las normas, políticas y procesos establecidos en la institución.

Es todo cuanto puedo decir en honor a la verdad, pudiendo el interesado hacer uso del presente como a bien lo tuviere.

Ing. Roberto Peñafiel
JEFE DE SISTEMAS
COAC ATUNTAQUI Ltda.

Certificación

Certifico que el presente trabajo de grado fue desarrollado en su totalidad por el Señor Wilson Aníbal Cárdenas Hernández, egresado de la Carrera de Ingeniería en Sistemas Computacionales previo la obtención del Título de Ingeniero en Sistemas Computacionales bajo mi dirección, para lo cual firmo en constancia.

_____

Ing. Marcelo Jurado
DIRECTOR DE TESIS

Agradecimiento

Al culminar una etapa tan importante de mi vida, quiero expresar mi agradecimiento profundo y sincero a Dios por convertirse en la luz y guía en mi camino, a todos quienes de una u otra manera estuvieron junto a mí para brindarme su apoyo de manera muy especial a mi madre, quien ha sido el soporte y pilar fundamental en mi vida.

Mi gratitud sincera a todos mis maestros por compartir sus conocimientos y experiencia, en especial a mi director de tesis, el Ing. Marcelo Jurado por el apoyo y compromiso mostrado durante la realización de la presente tesis.

A la Cooperativa de Ahorro y Crédito "Atuntaqui" Ltda., por facilitar los equipos y herramientas necesarias, con una mención especial al Ing. Roberto Peñafiel por su apoyo y colaboración incondicionales, los que han permitido culminar con éxito el presente proyecto.

Agradecer a mi familia y amigos por estar a mi lado y no permitir que decaiga ante las dificultades, con sus muestras de cariño y aliento en los momentos que necesite de un sabio consejo que me diera el empuje para seguir adelante.

A todos y cada uno mi eterno agradecimiento por contribuir en la consecución de este objetivo.

Dedicatoria

La vida está llena de sueños, metas y objetivos por cumplir y que grato es saber que cuentas con el apoyo de personas que contribuyen a alcanzarlos. Este trabajo se lo dedico a mi Dios por ser la luz que ilumina mi vida, por haber permitido que conozca personas esenciales que han sido el eje fundamental de mi formación personal y profesional.

A mi mamita linda Faby, por todo el esfuerzo, dedicación y entrega con los que me formó, por el infinito amor y fortaleza que demostró al dedicar su vida a cuidarme y educarme, por sus palabras de aliento que me daban la fuerza necesaria para continuar en los momentos difíciles, por comprender mis errores y compartir mis penas y mis alegrías.

A mi abuelita Rosita, mi segunda madre, por ser mi guía para llegar a ser un hombre de bien, inculcando en mí valores y principios, base fundamental de mi vida, quien mientras estuvo a mi lado siempre me demostró su amor, cariño y aprecio.

A mis hermanos Susana y Wladimir, por marcar la pauta de esfuerzo y superación en mi vida constituyéndose en mi ejemplo a seguir y compartir conmigo momentos que hoy son valiosas experiencias.

A mi hijo Matías Ariel, por ser mi inspiración y fortaleza en los momentos difíciles, mi razón de ser, quien con una sonrisa me da el aliento necesario para seguir adelante.

A mi novia Pauly, por ser mi complemento y siempre estar a mí lado brindándome su amor y apoyo incondicional, te amo!

Wilson.

Resumen

El presente trabajo de investigación se fundamenta en el uso de la herramienta de desarrollo JasperReports para la generación de documentos listos para imprimir desde nuestras aplicaciones de escritorio o portales web; mismos que permitirán disponer de información clara, precisa y oportuna, necesaria para el cumplimiento de nuestras tareas cotidianas, como la planificación de actividades, la toma de decisiones, generación de informes, entre otras. Además, cabe señalar que los servicios de información y documentación accesibles a través de internet, más concretamente mediante servidores web, están aumentando considerablemente.

La evolución de la web desde hace algunos años ha provocado la sustitución de páginas y documentos estáticos por documentos generados dinámicamente, debido a la interacción de los usuarios con la lógica de procesos y flujo de trabajo definidos por los creadores del servicio y a la disponibilidad de cada vez mayores repositorios de información. Evidentemente, se ha ido pasado progresivamente de un concepto de publicación de páginas web, bastante simple en su origen, a esquemas más complejos y diferenciados, fundamentados en procedimientos y técnicas basados en la gestión de información. A todo esto se suma que durante los últimos años las aplicaciones web han tenido mejor acogida por parte de los usuarios, no solo por la interfaz a la que están familiarizados; sino también a la posibilidad que nos brindan de poder acceder a estas desde cualquier parte del mundo.

Por esta razón, se presenta una nueva alternativa a los desarrolladores de aplicaciones web para la generación de documentos, mediante el uso de la librería JasperReports.

JasperReports es una herramienta de creación de reportes que tiene la habilidad de entregar contenido enriquecido al monitor, a la impresora o a ficheros. Su propósito principal es ayudar a crear documentos de tipo páginas, preparados para imprimir en una forma simple y flexible. Forma parte de la iniciativa Open Source, encontrándose bajo licencia GNU, motivo por el cual es software libre. JasperReports se usa comúnmente con iReport, un front-end gráfico de código abierto para la edición de informes.

El uso de herramientas Open Source nos brinda muchas ventajas, entre las cuales podemos anotar: la disminución de costos, independecia tecnológica, libertad de uso y redistribución, utilización de formatos estándares, entre otras. La idea bajo el concepto de código abierto es sencilla, los programadores pueden leer, modificar y redistribuir el código fuente de un programa, éste evoluciona, desarrolla y mejora. Los usuarios lo adaptan a sus necesidades, corrigen sus errores a una velocidad impresionante, dando como resultado la producción de un mejor software.

En conclusión, la presente investigación pretende sacarle el mayor provecho a las herramientas Open Source, como JasperReports, en el desarrollo de sistemas informáticos.

Summary

This research is based on using JasperReports development tool to generate ready to print documents from our desktop applications or web portals, which allow them to have clear, accurate and timely, necessary for the fulfillment of our daily tasks, such as business planning, decision making, reporting, among others.

Also, note that the information services and documentation accessible through the Internet, specifically through web servers are increasing significantly.

The evolution of the web in recent years has led to the replacement of static pages and documents by dynamically generated documents, due to user interaction with the logic and workflow processes defined by the creators of the service and the availability of each increasing information repositories. Obviously, it's gone past a concept gradually publishing web pages, rather simple in its origin, more complex and differentiated schemes, based on procedures and techniques based on information management. To all this the fact that in recent years web applications have had better reception from users, not just the interface they are familiar, but also give us the possibility of being able to access these from anywhere the world.

For this reason, presents a new alternative to web application developers to generate documents, using the JasperReports library.

JasperReports is a report creation tool that has the ability to deliver rich content to the monitor, printer or files. Its main purpose is to help create documents such as man pages, prepared for printing in a simple and flexible. Part of the Open Source Initiative, being under GNU license, because is free software. JasperReports is commonly used with iReport, a graphical front-end open source editing reports.

The use of Open Source tools gives us many advantages, among which we note: cost reduction, technological independence, freedom of use and redistribution, use of standard formats, among others. The idea under the open source concept is simple, programmers can read, modify and redistribute the source code of a program, this evolves, develops and improves. Users to suit their needs, correct their mistakes at an impressive speed, resulting in the production of better software.

In conclusion, this research aims to get the most out of Open Source tools such as JasperReports, the development of computer systems.

ÍNDICE DE CONTENIDOS

CAPÍTULO II

CAPÍTULO III

CAPÍTULO IV

CAPÍTULO V

CAPÍTULO VI

CAPÍTULO VII

ÍNDICE DE FIGURAS

CAPÍTULO I

CAPÍTULO II

CAPÍTULO III

CAPÍTULO IV

CAPÍTULO V

CAPÍTULO VI

ÍNDICE DE TABLAS

CAPÍTULO I

CAPÍTULO II

CAPÍTULO III

CAPÍTULO IV

CAPÍTULO V

CAPÍTULO VI

INTRODUCCIÓN

Un punto muy importante de los sistemas se presenta a la hora de mostrar la información resultante de los procesos de las aplicaciones y/o del día a día, principalmente cuando esto implica tomar decisiones comerciales y gerenciales. Dicha información, por lo general, esta almacenada en bases de datos o, en su defecto, en archivos planos.

Es posible realizar consultas a las bases de datos, leer estos archivos, diseñar y codificar ventanas o interfaces de usuarios, para interactuar con esa información. Pero este proceso se torna complicado cuando es necesario sumarle la funcionalidad de impresión, la personalización rápida de la información que estamos mostrando, o bien cuando la aplicación tiene múltiples interfaces, es decir, cuando el usuario puede utilizarla como un cliente Windows o desde un navegador.

A partir de estos puntos, toma importancia la generación de reportes, tanto para obtener dinamismo en las consultas, como para lograr múltiples vistas e incluso para facilitar el mantenimiento posterior y su extensibilidad.

Para Java, durante un tiempo, este fue uno de los puntos más débiles del lenguaje. Hoy en día existen diversas librerías y herramientas dedicadas (varias de ellas Open Source[1]) para la rápida generación de reportes. Veremos en esta ocasión la librería JasperReports, una de las más conocidas e interesantes, que combinada con herramientas para el diseño facilita y agiliza la generación, la previsualización y la impresión de los reportes.

JasperReports es precisamente, una poderosa herramienta para generar reportes en Java, con la habilidad de producir contenido complejo para la pantalla, directo para la impresora o en diferentes formatos de archivo (PDF, XLS, CSV y XML), entre otros. La librería es cien por ciento Java, y puede reutilizarse tanto en aplicaciones cliente y cliente/servidor, como en aplicaciones web, Java EE[2], etc.

JasperReports permite organizar la información obtenida desde una base de datos relacionada, a través de conectores JDBC[3], en diseños de reportes predefinidos en un formato XML[4].

Son varios los conceptos que deben conocerse, especialmente, su estructura de paquetes. La estructura de paquetes y las clases más importantes son:

- *net.sf.jasperreports.engine.JasperExportManager*
- *net.sf.jasperreports.engine.JasperPrintManager*
- *net.sf.jasperreports.engine.JasperFillManager*
- *net.sf.jasperreports.engine.JasperCompileManager*
- *net.sf.jasperreports.view.JasperViewer*

[1] Open Source es el término con el que se conoce al software distribuido y desarrollado libremente.
[2] Java Enterprise Edition, es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java con arquitectura de n capas distribuidas.
[3] Java Database Connectivity, es una API que permite la ejecución de operaciones sobre bases de datos desde Java.
[4] eXtensible Markup Language, es un metalenguaje extensible de etiquetas.

- *net.sf.jasperreports.view.JasperDesignViewer*

Estas clases presentan una abstracción para facilitar la vista del reporte, el diseño, la impresión, el llenado, la compilación, etc. Las primeras cuatro permiten manejar el motor de generación del reporte, y las últimas dos (las pertenecientes al paquete "*view*"), la visualización de los reportes.

Los reportes se generan basados en un diseño XML, armado y compilado antes de la puesta en marcha del motor generador del reporte. Una vez listo el diseño, el motor realiza la carga de datos a través de una conexión JDBC a una base de datos relacional, respetando los campos y las consultas predefinidas.

Cumplidos estos pasos, entran en juego las clases que permiten exportar, imprimir o visualizar el reporte, para darle fin a su ciclo de vida.

CAPÍTULO I

LA TECNOLOGÍA JAVA

INTRODUCCIÓN

CARACTERÍSTICAS

LA PLATAFORMA JAVA

EDICIONES DE LA PLATAFORMA JAVA

ARQUITECTURA JAVA ENTERPRISE EDITION

APPLETS

SERVLETS

JAVA DATABASE CONNECTIVITY

1.1. INTRODUCCIÓN

Java es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems[1] a principios de los años 90's. 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, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Java surgió en 1991 cuando un grupo de ingenieros de Sun Microsystems[1] trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos. La reducida potencia de cálculo y memoria de los electrodomésticos llevó a desarrollar un lenguaje sencillo capaz de generar código de tamaño reducido.

Debido a la existencia de diferentes tipos de CPU's y a los continuos cambios, era importante conseguir una herramienta independiente del tipo de CPU; desarrollaron un código "neutro" que no dependía del tipo de electrodoméstico, el cual se ejecutaba sobre una Máquina Virtual denominada Java Virtual Machine (JVM). Era la JVM quién interpretaba el código neutro convirtiéndolo a código particular de la CPU utilizada. Esto permitía lo que luego se ha convertido en el principal lema del lenguaje: *"Write Once, Run Anywhere"* (*"Escríbelo una vez, ejecútalo en cualquier lugar"*). A pesar de los esfuerzos realizados por sus creadores, ninguna empresa de electrodomésticos se interesó por el nuevo lenguaje.

Como lenguaje de programación para computadores, Java se introdujo a finales del año 1995. La clave fue la incorporación de un intérprete Java en la versión 2.0 del programa Netscape Navigator[2], produciendo una verdadera revolución en Internet. Java 1.1 apareció a principios del año 1997, mejorando sustancialmente la primera versión del lenguaje. Java 1.2, más tarde rebautizado como Java 2, nació a finales de 1998.

Entre diciembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL[3], de acuerdo con las especificaciones del *Java Community Process*[4], de tal forma que prácticamente todo el Java de Sun es ahora software libre, aunque la biblioteca de clases que se requiere para ejecutar programas Java aún no lo es.

Al programar en Java no se parte de cero. Cualquier aplicación que se desarrolle cuelga de un gran número de clases preexistentes. Algunas de ellas las ha podido hacer el propio usuario, otras pueden ser comerciales, pero siempre hay un número muy importante de clases que forman parte del propio lenguaje. Java incorpora en el propio lenguaje muchos aspectos que en cualquier otro lenguaje son extensiones propiedad de empresas de software o fabricantes

[1] Sun Microsystems fue una empresa informática recientemente (2009) adquirida por Oracle Corporation.
[2] Netscape Navigator fue un navegador web y el primer producto comercial de la compañía Netscape Communications.
[3] GNU General Public License, es una licencia creada por la Free Software Foundation en 1989 orientada a proteger la libre distribución, modificación y uso de software.
[4] Proceso formalizado que permite involucrarse en la definición de futuras versiones y características de la plataforma Java.

de ordenadores (threads, ejecución remota, componentes, seguridad, acceso a base de datos, etc.). Por eso muchos expertos opinan que Java es el lenguaje ideal para aprender la informática moderna, porque incorpora todos los conceptos de un modo estándar, mucho más sencillo y claro.

El 20 de Abril del 2009, Oracle[5] anuncia la compra de Sun Microsystems[1], la compañía creadora de Java, por la suma de $7,400 millones de dólares; tras la adquisición de Sun, Oracle se convierte en guardián absoluto de algunas de las joyas más importantes del código abierto, incluyendo la popular base de datos MySQL, el sistema operativo Solaris y la plataforma de desarrollo Java.

1.2. CARACTERÍSTICAS

Las características principales que nos ofrece la tecnología Java, respecto a cualquier otro lenguaje de programación son:

1.2.1. Simple

Java es un lenguaje de programación "simple" que puede ser utilizado sin extenso entrenamiento, mientras siga siendo adaptado a prácticas actuales de software. Los conceptos fundamentales de la tecnología Java son comprendidos rápidamente por los programadores. Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos.

El lenguaje de programación Java reduce notablemente los errores más comunes de programación de otros lenguajes, al eliminar muchas de sus debilidades.

1.2.2. Orientado a Objetos

La programación en Java está diseñada para ser orientada a objetos desde la raíz. Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación[6], herencia[7] y polimorfismo[8]. Las plantillas de objetos son llamadas clases y sus copias instancias. Estas instancias necesitan ser construidas y destruidas en espacios de memoria.

1.2.3. Distribuido

Java se ha construido con extensas capacidades de interconexión TCP/IP[9]. Existen librerías de rutinas para acceder e interactuar con protocolos como http[10] y ftp[11]. Esto permite a los

[5] Oracle Corporation es una de las mayores compañías de software del mundo.
[1] Sun Microsystems fue una empresa informática recientemente (2009) adquirida por Oracle Corporation.
[6] Es el ocultamiento de los datos miembro de un objeto y sólo se pueden manipular a través de sus métodos definidos.
[7] Característica para heredar las propiedades y el comportamiento de todas las clases a las que pertenecen.
[8] Capacidad para que varias clases derivadas de una antecesora utilicen un mismo método de forma diferente.
[9] Es un conjunto de protocolos de red basados en Internet y permiten la transmisión de datos entre computadoras.
[10] Hypertext Transfer Protocol es el protocolo usado en cada transacción de la World Wide Web.
[11] File Transfer Protocol, es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP

programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales.

La verdad es que Java en sí no es distribuido, sino que proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que se corran en varias máquinas, interactuando.

1.2.4. Robusto

La programación en lenguaje Java está diseñada para crear software fiable. Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible; en el ciclo de desarrollo. Maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de memoria. También implementa los arrays auténticos, en vez de listas enlazadas de punteros, con comprobación de límites, para evitar la posibilidad de sobrescribir o corromper la memoria, resultado de punteros que señalan a zonas equivocadas. Estas características reducen drásticamente el tiempo de desarrollo de aplicaciones en Java.

1.2.5. Arquitectura neutral

Gracias a la Máquina Virtual de Java los programas escritos en Java pueden ser compilados en una máquina y ejecutados en cualquier tipo de máquina o hardware (Windows, Mac, Linux, tablets, móviles, etc.); simplemente se debe tener instalado el sistema de ejecución de Java.

El código fuente Java se compila a un código de bytes (*bytecode*) de alto nivel independiente de la máquina. Este código está diseñado para ejecutarse en una máquina virtual que es implementada por el sistema de ejecución de Java "run-time", que sí es dependiente de la máquina.

1.2.6. Seguro

La tecnología Java está diseñada para operar en entornos distribuidos, lo cual significa que la seguridad es de extrema importancia. Con características de seguridad diseñadas dentro del lenguaje y el "*runtime*", Java te permite construir aplicaciones que no pueden ser invadidas desde fuera. En un entorno de red, las aplicaciones escritas en lenguaje de programación Java son seguras de intrusiones no autorizadas, virus o sistemas de invasión de archivos.

El código Java pasa muchas pruebas antes de ejecutarse en una máquina. El código se pasa a través de un verificador de "*bytecode*" que comprueba el formato de los fragmentos de código y aplica un probador de teoremas para detectar fragmentos de código ilegal.

1.2.7. Portable

Los programas desarrollados en Java son los mismos en cada plataforma, no hay incompatibilidades de tipo de datos entre arquitecturas de hardware y software. Java implementa estándares de portabilidad para facilitar el desarrollo, los enteros son siempre

enteros. Además, Java construye sus interfaces de usuario a través de un sistema abstracto de ventanas de forma que las ventanas puedan ser implantadas en entornos Unix, Pc o Mac.

1.2.8. Interpretado

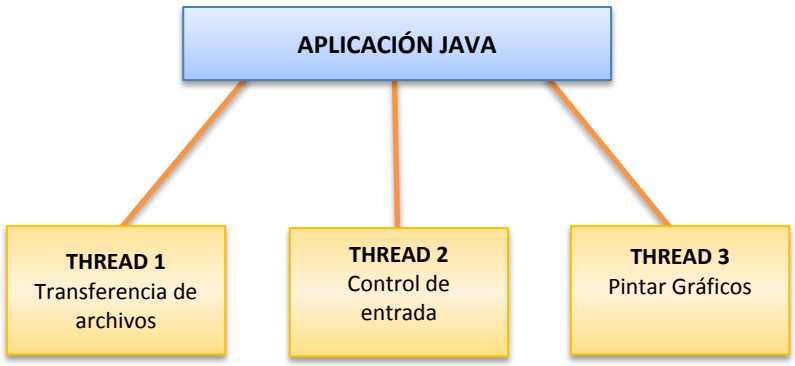
El intérprete de Java (run-time) puede ejecutar directamente el código objeto. Enlazar un programa, normalmente consume menos recursos que compilarlo; por lo que los desarrolladores con Java pasarán más tiempo desarrollando y menos esperando por el ordenador.

La verdad es que Java para conseguir ser un lenguaje independiente del sistema operativo y del procesador que incorpore la máquina utilizada, es tanto interpretado como compilado. Y esto no es ningún contrasentido; el código fuente escrito con cualquier editor se compila generando el "*bytecode*". Este código intermedio es de muy bajo nivel, pero sin alcanzar las instrucciones de máquina propias de cada plataforma. Ese mismo código es el que se puede ejecutar sobre cualquier plataforma. Para ello hace falta el "run-time", que sí es completamente dependiente de la máquina y del sistema operativo, que interpreta dinámicamente el "*bytecode*".

1.2.9. Multihilo

La capacidad multihilo de la tecnología Java proporciona el significado de construir aplicaciones con muchos hilos (procesos) concurrentes de actividad. El multiprocesamiento da como resultado un alto grado de interactividad para el usuario final.

Al ser multihilo, Java permite muchas actividades simultáneas en un programa. Los threads son básicamente pequeños procesos o piezas independientes de un gran proceso. El beneficio es un mejor rendimiento interactivo y mejor comportamiento en tiempo real.



Fuente: [Propia]

Figura 1.1. Aplicación Java multihilo.

Cualquiera que haya utilizado la tecnología de navegación concurrente, sabe lo frustrante que puede ser esperar por una gran imagen que se está trayendo. En Java, las imágenes se pueden

ir trayendo en un thread independiente, permitiendo que el usuario pueda acceder a la información en la página sin tener que esperar por el navegador.

1.2.10. Dinámico

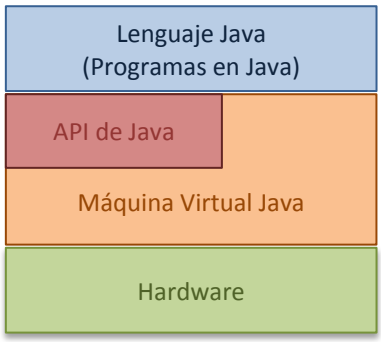
El lenguaje y el sistema "run-time" son dinámicos en sus etapas de conexión. Las clases son conectadas solamente si son necesarias. Nuevos módulos de código pueden ser conectados en una demanda desde una variedad de fuentes, incluso de fuentes a lo largo de la red. Java no intenta conectar todos los módulos que comprenden una aplicación hasta el tiempo de ejecución; las librerías nuevas o actualizadas no paralizarán las aplicaciones actuales.

Java para evitar que los módulos de *bytecode*, los objetos o nuevas clases sean traídos a través de la red cada vez que se necesiten, implementa las opciones de persistencia, para que no se eliminen cuando se limpie la caché de la máquina.

1.3. PLATAFORMA JAVA

Se entiende por plataforma al entorno de hardware y software en el cual se ejecutan los programas; así tenemos, que Java es una plataforma de software que se ejecuta sobre otras plataformas y puede ser usado sobre varios sistemas operativos y hardware.

La plataforma Java está constituida por tres componentes principales: la máquina virtual, el API de Java y el lenguaje Java en sí.



Fuente: [Propia]
Figura 1.2. Componentes de la plataforma Java.

1.3.1. Máquina Virtual Java (JVM)

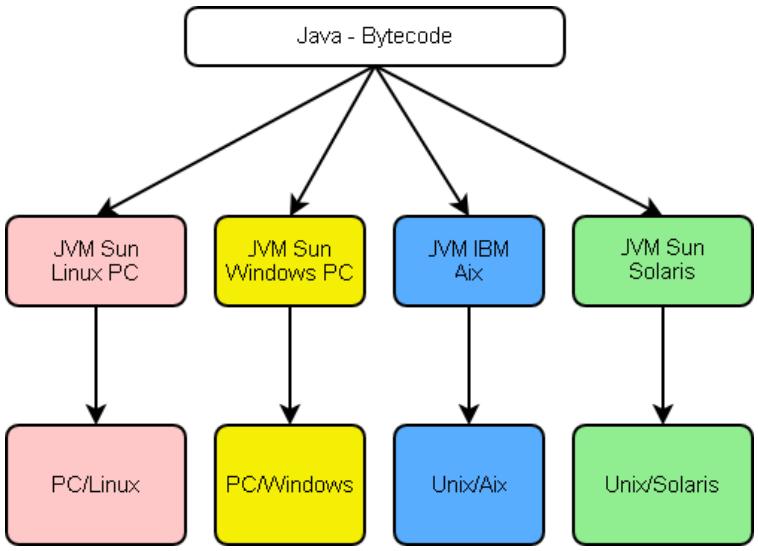
La Máquina Virtual Java (*Java Virtual Machine*) es un programa nativo; es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (*bytecode*), el cual es generado por el compilador del lenguaje Java.

El código binario de Java no es un lenguaje de alto nivel, sino un verdadero código máquina de bajo nivel, viable incluso como lenguaje de entrada para un microprocesador físico, fue desarrollado originalmente por Sun Microsystems.

La JVM es una de las piezas fundamentales de la plataforma Java. Básicamente se sitúa en un nivel superior al hardware del sistema sobre el que se pretende ejecutar la aplicación, y este

actúa como un puente que entiende tanto el *bytecode*, como el sistema sobre el que se pretende ejecutar. Así, cuando se escribe una aplicación Java, se hace pensando que será ejecutada en una Máquina Virtual Java en concreto, siendo ésta la que en última instancia convierte el código *bytecode* a código nativo del dispositivo final.

La gran ventaja de la Máquina Virtual Java es aportar portabilidad al lenguaje de manera que desde Sun Microsystems se han creado diferentes máquinas virtuales Java para diferentes arquitecturas y así un programa ".*class*" escrito en un Windows puede ser interpretado en un entorno Linux. Tan solo es necesario disponer de dicha máquina virtual para dichos entornos. De ahí el famoso axioma que sigue a Java, *"Write Once, Run Anywhere"* (*"Escríbelo una vez, ejecútalo en cualquier lugar"*).



Fuente: [W01]

Figura 1.3. Esquema general de un programa corriendo en un JVM.

Los intentos de la compañía propietaria de Java y productos derivados de construir microprocesadores que aceptaran el Java *bytecode* como su lenguaje de máquina fueron más bien infructuosos.

La Máquina Virtual de Java puede estar implementada en software, hardware, una herramienta de desarrollo o un web browser; lee y ejecuta código precompilado *bytecode* que es independiente de la plataforma "multiplataforma". La JVM provee definiciones para un conjunto de instrucciones, un conjunto de registros, un formato para archivos de clases, la pila y un área de memoria. Cualquier implementación de la JVM que sea aprobada por Sun debe ser capaz de ejecutar cualquier clase que cumpla con la especificación.

[W01] Wikipedia, La enciclopedia libre, Máquina Virtual Java, *http://es.wikipedia.org/wiki/JVM.*

1.3.2. API de Java

El conjunto de bibliotecas del lenguaje es conocido como la *Aplication Programming Interface* de Java o en siglas API de Java y es un conjunto de componentes que proporcionan diferentes herramientas para el desarrollo. La API de Java está formada por un conjunto de paquetes de clases que le proporcionan una gran funcionalidad. El núcleo de la API viene con cada una de las implementaciones de la Máquina Virtual de Java.

1.3.3. Lenguaje Java

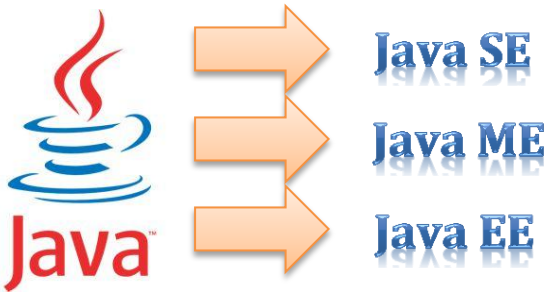
Bruce Eckel en la cuarta edición de su libro Piensa en Java, define: "*Java es un lenguaje de programación orientado a objetos puro, en el sentido de que no hay ninguna variable, función o constante que no esté dentro de una clase. Se accede a los miembros dato y las funciones miembro a través de los objetos y de las clases*"[L01].

Los tipos de programas más comunes que se pueden hacer con Java son los applets y las aplicaciones. Otro tipo especial de programa se denomina servlet que es similar a los applets pero se ejecutan en los servidores Java.

1.4. EDICIONES DE LA PLATAFORMA JAVA

Existen tres distribuciones básicas de la plataforma Java en un intento por cubrir distintos entornos de aplicación. Así, ha distribuido muchas de sus API's de forma que pertenezcan a cada una de las plataformas:

- Java Standard Edition (Java SE)
- Java Micro Edition (Java ME)
- Java Enterprise Edition (Java EE)



Fuente: [Propia]

Figura 1.4. Ediciones de la plataforma Java.

Las clases en las API's de Java se organizan en grupos disjuntos llamados paquetes. Cada paquete contiene un conjunto de interfaces, clases y excepciones relacionadas. La información acerca de los paquetes que ofrece cada plataforma puede encontrarse en la documentación que proporciona cada una.

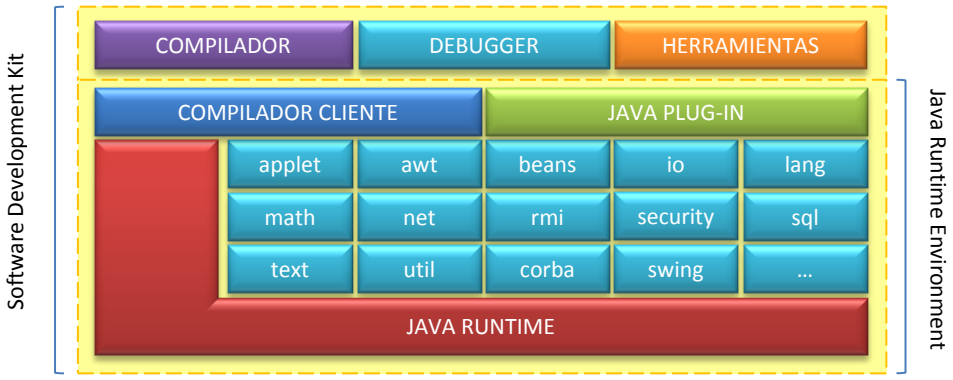
[L01] Bruce Eckel es autor de libros y artículos sobre programación. Sus obras más conocidas son *Piensa en Java* y Piensa en C++. Eckel fue miembro fundador del comité ANSI/ISO C++ standard.

El conjunto de las API's es controlado por Oracle Corporation[5] junto con otras entidades o personas a través del programa *Java Community Process*[4]. Las compañías o individuos participantes del *JCP* pueden influir de forma activa en el diseño y desarrollo de las API's.

1.4.1. Java Standard Edition

Java Standard Edition es la edición que se emplea en computadoras personales (desktops y laptops). Se le conoce también como Java Desktop y es la versión que tienes que instalar para poder programar en Java en tu computadora, aunque tus programas estén destinados para alguna de las otras ediciones. Java Platform Standard Edition o Java SE (conocido anteriormente hasta la versión 5.0 como plataforma *Java 2 Standard Edition* o *J2SE*), es una colección de API's del lenguaje de programación Java útiles para muchos programas de la plataforma Java.

Java SE, está compuesto de dos partes uno que es el SDK (*Software Development Kit*) que tiene todo el software necesario para el desarrollo de aplicaciones Java, y el JRE (*Java Runtime Environment*) que contiene solo el ambiente y las bibliotecas principales para ejecutar el software escrito en Java.



Fuente: [Propia]

Figura 1.5. Estructura de la plataforma Java Standard Edition (JSE).

1.4.2. Java Micro Edition

Java Micro Edition es la edición que se emplea en dispositivos móviles, tales como los teléfonos celulares. Es una versión recortada del Java SE con ciertas extensiones enfocadas a las necesidades particulares de esos tipos de dispositivos.

La plataforma Java Micro Edition, o Java ME (anteriormente J2ME), es una colección de API's de Java orientadas a productos de consumo como tablets, celulares o electrodomésticos.

Java ME se ha convertido en una buena opción para crear juegos en teléfonos móviles debido a que se puede emular en un PC durante la fase de desarrollo y luego subirlos fácilmente al teléfono. Al utilizar tecnologías Java el desarrollo de aplicaciones o videojuegos con estas API's resulta bastante económico de portar a otros dispositivos.

[5] Oracle Corporation es una de las mayores compañías de software del mundo.
[4] Proceso formalizado que permite involucrarse en la definición de futuras características de la plataforma Java.

1.4.3. Java Enterprise Edition

Java Enterprise Edition es la edición que se emplea para hacer aplicaciones. Incluye a toda la Standard Edition y muchas, muchas extensiones más. Java EE es un grupo de especificaciones diseñadas por Oracle Corporation que permiten la creación de aplicaciones empresariales, esto sería: acceso a base de datos (JDBC[12]), utilización de directorios distribuidos (JNDI[13]), acceso a métodos remotos (CORBA[14]), funciones de correo electrónico (JavaMail[15]), aplicaciones Web (JSP y Servlets), etc.

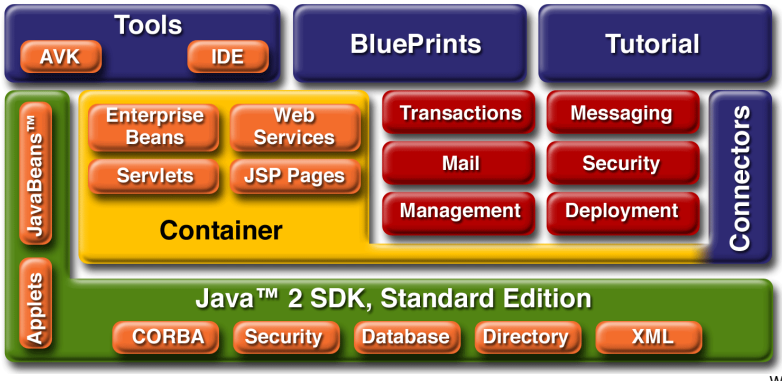
Aquí es importante notar que JEE es solo una especificación, esto permite que diversos productos sean diseñados alrededor de estas especificaciones; la especificación más reciente de Oracle[5] es Java EE 6, la cual está conformada por: JSP 2.2, Servlet 3.0, EJB 3.1, JavaMail 1.4, Connector 1.6 entre otros API's. El detalle de todas las especificaciones se encuentra en el sitio oficial "*http://www.oracle.com/technetwork/ java/javaee*".

Aunque varios productos Java están diseñados alrededor de estas especificaciones, no todos cumplen con el estándar completo, esto es, Tomcat solo cumple las especificaciones de JSP y Servlets; sin embargo, existen productos como Websphere y algunos otros "*Java Application Servers*" que son considerados "Fully Java EE Compliant", en otras palabras, cumplen con todas las especificaciones definidas por Oracle.

1.5. ARQUITECTURA JAVA ENTERPRISE EDITION

La plataforma Java EE es una plataforma para crear aplicaciones empresariales utilizando un modelo de multicapas; dividiendo la aplicación en diferentes niveles, cada uno basándose en una tarea en particular. Está basado en Java SE y un conjunto de sus API's a la cual JEE aporta la especificación de componentes.

Ahora, observemos los componentes que comprenden esta plataforma:



Fuente: [W02]

Figura 1.6. Arquitectura de la plataforma Java EE.

La parte externa de la plataforma se encuentra formada por JavaBeans, Applets, SQL, etc. Un

[12] Java Database Connectivity, es una API que permite la ejecución de operaciones sobre bases de datos desde Java.
[13] Java Naming and Directory Interface es una interfaz de programación de aplicaciones (API) para servicios de directorio.
[14] Common Object Request Broker Architecture es un estándar de una plataforma de desarrollo de sistemas distribuidos.
[15] JavaMail es una expansión de Java que facilita el envío y recepción de e-mail desde código java.
[5] Oracle Corporation es una de las mayores compañías de software del mundo.
[W02] Arquitecturas software, J2EE, *http://servidoresdeaplicaciones.wordpress.com/2009/03/09/arquitectura-j2ee/*.

nivel más interno tenemos el container o contenedor, que es el encargado de manejar los EJB, JSP, Servlets; y en un nivel más profundo tenemos las transacciones, mensajería y mail.

De todos estos elementos nos centraremos básicamente en el estudio de: EJBs, JSP, Servlets y SQL; de forma rápida podemos decir que:

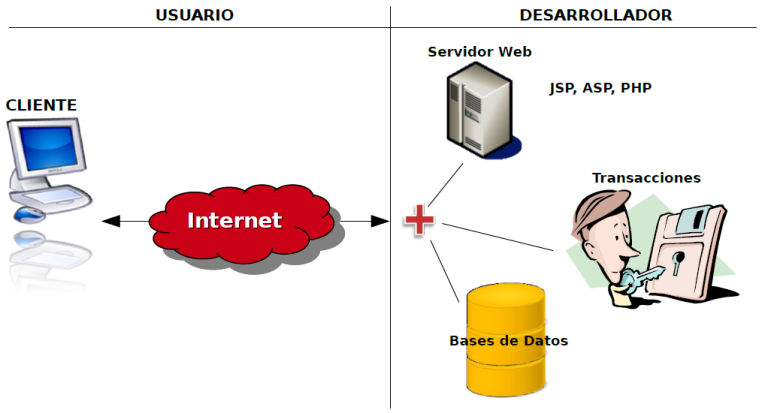
- **EJBs.-** Son módulos encargados de manejar toda la lógica de programación detrás de nuestra aplicación.
- **Servlets.-** Programas en Java que permiten la comunicación y la ejecución de programas Java a través de una aplicación web.
- **JSP.-** Quienes manejan interfaces de comunicación con el usuario o cliente.
- **SQL.-** Contiene toda la lógica de programación que comunica nuestra aplicación con el sistema gestor de base de datos.

Todos estos elementos nos permitirán desarrollar aplicaciones básicas utilizando Java EE como herramienta de trabajo.

Como se dijo anteriormente JEE utiliza una lógica de programación desarrollada en niveles o capas. Lo que nos permitirá encapsular o separar elementos de nuestras aplicaciones en partes claramente definidas; es decir, podemos dejar procesos en un lugar, datos en otros y mostrar interfaces en otro.

1.5.1. Modelo de capas

En la programación por capas básicamente la idea es buscar la forma de separar lo que ve el usuario con los procesos creados por el desarrollador. Así, tenemos diferentes lenguajes que nos permiten desarrollar aplicaciones por capa por ejemplo JSP, ASP, PHP. Por lo tanto debemos primero, ver de forma global donde se encuentra el usuario y donde se encuentra el desarrollador.



Fuente: [W03]

Figura 1.7. Modelo de programación en capas.

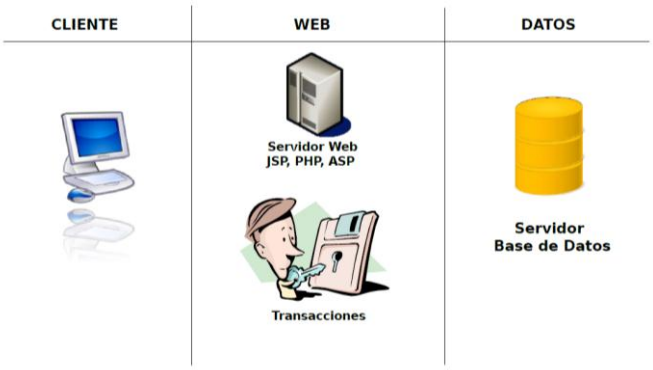
[W03] Aplicaciones en capa, *http://oness.sourceforge.net/proyecto/html/ch03s02.html.*

Como observamos entonces ya tenemos una capa claramente definida que es la del cliente en el cual se realizaran todas las peticiones a las capas superiores y estas a su vez enviaran las respuestas a esas solicitudes.

El servidor web es el encargado de realizar todos los procesos lógicos como crear consultas crear formularios, ejecutar procesos, etc. El servidor de bases de datos es donde, como su nombre lo dice, tenemos todas las tablas, vistas y consultas que necesitamos para nuestro sistema. Y la seguridad junto con las transacciones que contienen toda la lógica de negocio entre la aplicación y la base de datos.

1.5.1.1. Modelo de tres capas

Ahora que sucede en la arquitectura de tres capas, tenemos las transacciones con el servidor web realizándose en un mismo lugar; o en este caso una misma capa, y los datos se mantienen a su vez también en una capa aún más superior y alejada del cliente, para dar un nivel más de seguridad. Observemos la siguiente figura, en la cual se definen las tres capas básicas del desarrollo de aplicaciones web.

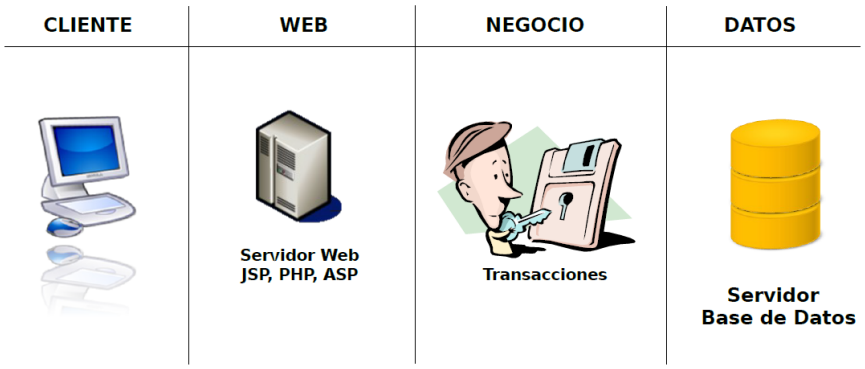


Fuente: [W03]

Figura 1.8. Modelo de programación de tres capas.

1.5.1.2. Modelo de cuatro capas

En esta arquitectura lo principal es que, separamos la segunda capa en dos: Capa Web que queda solamente con el Servidor Web y las Transacciones se realizan en la Capa de Negocio.



Fuente: [W03]

Figura 1.9. Modelo de programación de cuatro capas.

[W03] Aplicaciones en capa, *http://oness.sourceforge.net/proyecto/html/ch03s02.html.*

Esto nos permitirá un mejor control de todos los elementos que entran en juego a la hora de desarrollar aplicaciones web.

1.5.1.3. Modelo de n capas

A pesar de que el modelo de cuatro capas provee mayor flexibilidad, se puede desagregar cada capa, generando n componentes que implementen la misma funcionalidad. Esto permite mayor escalabilidad y flexibilidad en la construcción del sistema.

Ventajas del modelo

- Desarrollos paralelos en cada capa.
- Aplicaciones más robustas debido al encapsulamiento.
- Mantenimiento y soporte más sencillo, es más sencillo cambiar un componente que modificar una aplicación monolítica.
- Mayor flexibilidad, se pueden añadir nuevos módulos para dotar al sistema de nueva funcionalidad.
- Alta escalabilidad. La principal ventaja de una aplicación distribuida bien diseñada es su buen escalado, es decir, que puede manejar muchas peticiones con el mismo rendimiento simplemente añadiendo más hardware. El crecimiento es casi lineal y no es necesario añadir más código para conseguir esta escalabilidad.

Como tecnología, las arquitecturas de n capas proporcionan una gran cantidad de beneficios para las empresas que necesitan soluciones flexibles y fiables para resolver complejos problemas inmersos en cambios constantes.

1.5.2. Arquitectura aplicaciones JEE

Las aplicaciones JEE se basan completamente en el modelo de aplicaciones de cuatro niveles. Todas las aplicaciones constan de tres partes básicas, que son:

- **Módulo EJB.-** encargado de tener la lógica del negocio y transacciones. En otras palabras podemos decir, que es el encargado de ejecutar programas, consultas a la base de dato principalmente. (Capa de Negocio).
- **Módulo WAR.-** que es la encargada de tener todos los elementos de interfaz como páginas web, servlets, applets. (Capa Web).
- **Aplicación EAR.-** contiene en su interior toda la configuración de la aplicación J2EE, eso incluye el módulo WAR y EJB.

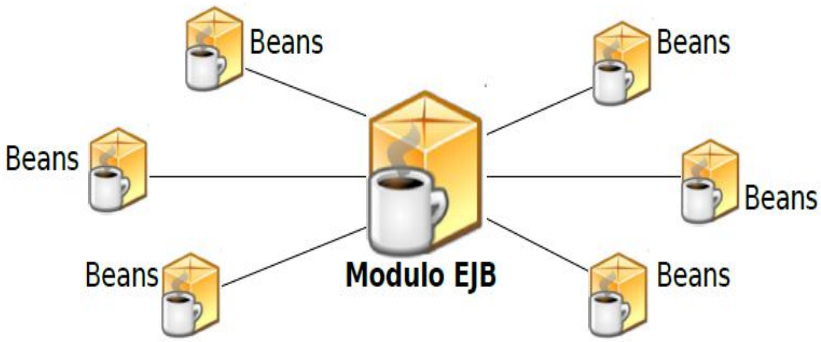
1.5.2.1. Módulo EJB

En este módulo entran en juego varias cosas, ya que es el encargado de administrar la mayor parte de la lógica de nuestra aplicación.

¿Qué son los Enterprise JavaBeans?

Los Enterprise JavaBeans (también conocidos por sus siglas EJB), proporcionan un modelo de componentes distribuido estándar del lado del servidor. El objetivo de los EJBs es dotar al programador de un modelo que le permita abstraerse de los problemas generales de una aplicación empresarial (concurrencia, transacciones, persistencia, seguridad, etc.) para centrarse en el desarrollo de la lógica de negocio en sí. El hecho de estar basado en componentes permite que éstos sean flexibles y sobre todo reutilizables.

No hay que confundir los Enterprise JavaBeans con los JavaBeans. Los JavaBeans también son un modelo de componentes creado Sun Microsystems para la construcción de aplicaciones, pero no pueden utilizarse en entornos de objetos distribuidos al no soportar nativamente la invocación remota (RMI).



Fuente: [W04]

Figura 1.10. Modelo de un Enterprise JavaBeans.

Tipos de Enterprise JavaBeans

Existen tres tipos de EJBs:

- **EJBs de Entidad (Entity EJBs).-** su objetivo es encapsular los objetos del lado del servidor que almacena los datos. Los EJB de entidad presentan la característica fundamental de la persistencia:

  Persistencia gestionada por el contenedor (CMP).- el contenedor se encarga de almacenar y recuperar los datos del objeto de entidad mediante el mapeo o vinculación de las columnas de una tabla de la base de datos con los atributos del objeto.

  Persistencia gestionada por el bean (BMP).- el propio objeto entidad se encarga, mediante una base de datos u otro mecanismo, de almacenar y recuperar los datos a los que se refiere, por lo cual, la responsabilidad de implementar los mecanismos de persistencia es del programador.

[W04] Documentación Java, Tutorial Java EE6, *http://java.sun.com/javaee/6/docs/tutorial/doc/docinfo.html.*

- **EJBs de Sesión (Session EJBs).-** gestionan el flujo de la información en el servidor. Generalmente sirven a los clientes como una fachada de los servicios proporcionados por otros componentes disponibles en el servidor. Puede haber dos tipos:

  Con estado (stateful).- En un bean de sesión con estado, las variables de instancia del bean almacenan datos específicos obtenidos durante la conexión con el cliente. Cada bean de sesión con estado, por tanto, almacena el estado conversacional de un cliente que interactúa con el bean. Este estado conversacional se modifica conforme el cliente va realizando llamadas a los métodos de negocio del bean. El estado conversacional no se guarda cuando el cliente termina la sesión.

  Sin estado (stateless).- Los beans de sesión sin estado son objetos distribuidos que carecen de estado asociado permitiendo por tanto que se los acceda concurrentemente. No se garantiza que los contenidos de las variables de instancia se conserven entre llamadas al método.

- **EJBs dirigidos por mensajes (Message-driven EJBs).-** son los únicos beans con funcionamiento asíncrono. Usando el *Java Messaging System* (JMS), se suscriben a un tema (topic) o a una cola (*queue*) y se activan al recibir un mensaje dirigido a dicho tema o cola. No requieren de su instanciación por parte del cliente.

A partir de la versión JEE 5.0 de Java, los Beans de Entidad desaparecen ya que son remplazados por JPA (*Java Persistence API*).

Java Persistence API, más conocida por sus siglas *JPA*, es la API de persistencia desarrollada para la plataforma Java EE. La *Java Persistence API*, es un framework del lenguaje de programación Java que maneja datos relacionales en aplicaciones usando la Plataforma Java en sus ediciones Standard (*Java SE*) y Enterprise (*Java EE*).

La JPA fue originada a partir del trabajo del JSR 220 Expert Group. Ha sido incluida en el estándar EJB3. La persistencia en este contexto cubre tres áreas:

- La API en sí misma, definida en *javax.persistence.package*
- La Java Persistence Query Language (JPQL)
- Metadatos objeto/relacional

El objetivo que persigue el diseño de esta API es no perder las ventajas de la orientación a objetos al interactuar con una base de datos, siguiendo el patrón de mapeo objeto-relacional y permitir usar objetos regulares.

Funcionamiento de un Enterprise JavaBean

Los EJBs se disponen en un contenedor EJB dentro del servidor de aplicaciones. La especificación describe cómo el EJB interactúa con su contenedor y cómo el código cliente interactúa con la combinación del EJB y el contenedor.

Cada EJB debe facilitar una clase de implementación Java y dos interfaces Java. El contenedor EJB creará instancias de la clase de implementación Java para facilitar la implementación EJB. Las interfaces Java son utilizadas por el código cliente del EJB. Las dos interfaces, conocidas como Interfaz Home e Interfaz Remota, especifican las firmas de los métodos remotos del EJB.

Los métodos remotos se dividen en dos grupos:

- Métodos que no están ligados a una instancia específica, por ejemplo aquellos utilizados para crear una instancia EJB o para encontrar una entidad EJB existente. Estos métodos se declaran en la interfaz home.
- Métodos ligados a una instancia específica. Se ubican en la interfaz remota.

Dado que se trata simplemente de interfaces Java y no de clases concretas, el contenedor EJB genera clases para esas interfaces que actuarán como un proxy en el cliente. El cliente invoca un método en los proxies generados que a su vez sitúa los argumentos método en un mensaje y envía dicho mensaje al servidor EJB. Los proxies usan RMI-IIOP[16] para comunicarse con el servidor EJB.

El servidor llamará a un método correspondiente a una instancia de la clase de implementación Java para manejar la llamada del método remoto.

Interfaz Home

La Interfaz Home permite al código cliente manipular métodos de clase del EJB que no están asociados a ninguna instancia particular. La Interface Home permite crear las instancias de EJB de entidad o sesión a través del método *create* que puede ser sobrecargado.

La especificación EJB 1.1 establece el tipo de métodos de clase que se pueden definir como métodos que crean un EJB o para encontrar un EJB existente si es un "bean" de entidad. La especificación EJB 2.0 permite a los desarrolladores de aplicaciones definir nuevos métodos de clase sin limitarse a su sola creación o borrado.

Interfaz Remota

La interfaz remota especifica los métodos de instancia públicos encargados de realizar las operaciones.

Una sesión bean puede implementar múltiples interfaces, con cada interfaz apuntada por un tipo de cliente diferente. La interfaz local es para aquellos clientes que corren en la misma máquina virtual que el contenedor EJB. La interfaz remota es para clientes remotos. Frente a una consulta del cliente, el contenedor retorna un stub serializado del objeto que implementa la interfaz remota. El stub conoce cómo pasar llamadas a procedimientos remotos (RPCs) al servidor.

[16] Este estándar fue creado intentando simplificar el desarrollo de las aplicaciones CORBA, mientras preservaba todos los beneficios principales.

1.5.2.2. Módulo WAR

Este módulo es el encargado de la comunicación Cliente - Servidor Web, en el realizan las peticiones a las capas superiores, en el tenemos tres elementos básicos que son los:

- **JSP.-** Java Server Pages.
- **Servlets.-** Componente web desarrollada con el objetivo de procesar requerimientos de un cliente o request y genera respuestas con contenido web dinámico. Para ser ejecutados es necesaria la utilización de un servidor que soporte Servlets y su container.
- **Aplicaciones Java.-** códigos Java en su forma pura, que tienen métodos y atributos y que pueden ser utilizados en aplicaciones JSP y Servlets.

Muchas veces se tiende a utilizar los JSP de la misma forma que páginas PHP, lo cual no es recomendable debido a la gran cantidad de código que se genera y lo cual produce mucho desorden por esto se hará gran énfasis a respetar la arquitectura J2EE, en donde debemos tener cada parte donde corresponde, vale decir, Interfaces en la capa web, peticiones y respuestas en la capa de negocio y las conexiones de la base de datos en la capa de datos.

Comunicación Capa Web y Capa de Negocio

Cuando recién se comienza el desarrollo de estas aplicaciones, cuesta un poco saber dónde realizar la comunicación de cada una de las capas; uno tiende a escribir todo el código en las aplicaciones JSP, lo que no era recomendable, por lo que debemos hacerlo de la siguiente forma:

Fuente: [W04]

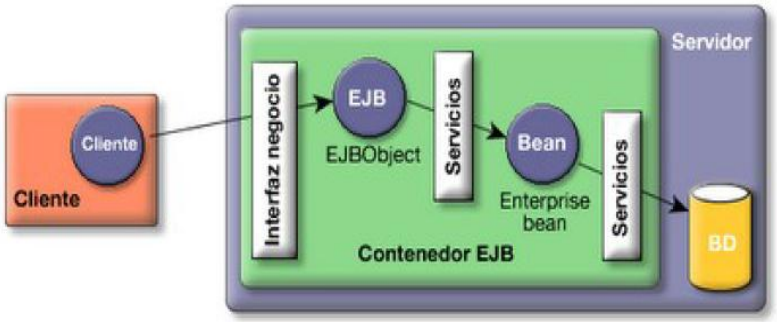
Figura 1.12. Comunicación entre la Capa Web y la Capa Negocio.

Se deben utilizar los Servlets como ente comunicador de ambas capas, para poder mantener la arquitectura de la aplicación y el orden.

1.5.2.3. El Contenedor o Container

Los contenedores son un servicio que proporciona la infraestructura necesaria a un componente para ser ejecutado, para proveer sus servicios a un cliente y dar comunicación con otras componentes. Un producto típico proveerá un contenedor para cada tipo de componente de la aplicación: contenedor de la aplicación cliente, para applets y para componentes de EJB.

[W04] Documentación Java, Tutorial Java EE6, *http://java.sun.com/javaee/6/docs/tutorial/doc/docinfo.html.*



Fuente: [W04]

Figura 1.13. Funcionamiento del contenedor EJB

¿De qué cosas se preocupa el contenedor?

- ¿Tiene permisos el cliente para llamar al método?
- ¿Hay que abrir la transacción al comienzo de la llamada y cerrarla al terminar?
- ¿Es necesario refrescar el bean con los datos de la base de datos?

1.6. APPLETS

Un applet Java es un applet escrito en el lenguaje de programación Java. Los applets de Java pueden correr en un navegador web utilizando la Java Virtual Machine, o en el AppletViewer.

Entre sus características podemos mencionar un esquema de seguridad que permite que los applets que se ejecutan en el equipo no tengan acceso a partes sensibles; por ejemplo, no pueden escribir archivos, a menos que uno mismo le dé los permisos necesarios en el sistema. La desventaja de este enfoque es que la entrega de permisos es engorrosa para el usuario común, lo cual juega en contra de uno de los objetivos de los Java applets: proporcionar una forma fácil de ejecutar aplicaciones desde el navegador web.

En Java un applet, es un programa que puede incrustarse en un documento HTML[17], es decir, en una página web. Cuando un navegador carga una página web que contiene un applet, éste se descarga en el navegador web y comienza a ejecutarse. Esto nos permite crear programas que cualquier usuario puede ejecutar con tan solo cargar la página web en su navegador.

El navegador que carga y ejecuta el applet se conoce en términos genéricos como el contenedor de applets. El kit de desarrollo de software para Java 2 (J2SDK) incluye el contenedor de applets, llamado AppletViewer, para probar los applets antes de incrustarlos en una página web.

Ventajas

Los applets de Java suelen tener las siguientes ventajas:

[W04] Documentación Java, Tutorial Java EE6, *http://java.sun.com/javaee/6/docs/tutorial/doc/docinfo.html.*

[17] HyperText Markup Language, es el lenguaje de marcado predominante para la elaboración de páginas web.

- Son multiplataforma (funcionan en Linux, Windows, Mac OS, y en cualquier sistema operativo para el cual exista una JVM).
- Es soportado por la mayoría de los navegadores web.
- Puede ser almacenado en la memoria cache de la mayoría de los navegadores web, de modo que se cargará rápidamente cuando se vuelva a cargar la página web, aunque puede quedar atascado en la caché, causando problemas cuando se liberan nuevas versiones.
- Puede tener acceso completo a la máquina en la que se está ejecutando, si el usuario lo permite.
- Puede ejecutarse con velocidades comparables a la de otros lenguajes compilados, como C + +, pero muchas veces más rápida que la de JavaScript.
- Puede trasladar el trabajo del servidor al cliente, haciendo una solución web más escalable tomando en cuenta el número de usuarios / clientes.

Desventajas

Los applets de Java suelen tener las siguientes desventajas:

- Requiere el plug-in de Java, que no está disponible por defecto en todos los navegadores web.
- No puede iniciar la ejecución hasta que la JVM esté en funcionamiento, y esto puede tomar tiempo la primera vez que se ejecuta un applet.
- Si no está firmado como confiable, tiene un acceso limitado al sistema del usuario, en particular no tiene acceso directo al disco duro del cliente o al portapapeles.
- Algunas organizaciones sólo permiten la instalación de software a los administradores. Como resultado, muchos usuarios sin privilegios para instalar el plug-in en su navegador no pueden ver los applets.
- Un applet podría exigir una versión específica del JRE, y el cliente se verá obligado a esperar durante la descarga de la nueva JRE.

Diferencias entre una aplicación autónoma y un applet

Existen diferencias entre un programa autónomo y un applet:

- Restricciones de seguridad: los applets son considerados código de poca confianza (a excepción de que lleven una firma digital) ya que son compartidos por todos los usuarios de internet. Por ejemplo, no se permite el acceso a ficheros locales ni conectarse a un servidor distinto al que está alojado el applet.
- Necesitan un navegador para ser visualizados, o un visor de applets como AppletViewer.
- No tienen un método principal.

Ciclo de vida

Cuando un applet se inicia, se llaman en este orden a los siguientes métodos:

1. ***init*.-** Suele contener instrucciones para inicializar el applet.
2. ***start*.-** Como init, se suele usar para inicializar, pero con la diferencia de que este método también se llama cuando se reinicia un applet.
3. ***paint*.-** Se encarga de mostrar el contenido del applet. Se ejecuta cada vez que se tenga que redibujar.

Para terminar o pausar la ejecución se llama a los siguientes métodos:

- ***stop*.-** suspende la ejecución del programa. Se llama cuando el applet se vuelve temporalmente invisible.
- ***destroy*.-** cuando no se va a necesitar más el applet. Se usa para liberar recursos.

1.7. SERVLETS

Los servlets, son objetos que corren dentro del contexto de un contenedor de servlets y extienden su funcionalidad. Se usan particularmente para generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web. La palabra servlet deriva de otra anterior applet, que se refiere a pequeños programas que se ejecutan en el contexto de un navegador web. Por contraposición, un servlet es un programa que se ejecuta en un servidor.

La especificación original de servlets fue creada por Sun Microsystems[1] (la versión 1.0 fue terminada en junio de 1997). Comenzando con la versión 2.3, la especificación de servlet fue desarrollada siguiendo el Proceso de la Comunidad Java (*Java Community Process*[4]).

Un servlet es un objeto que se ejecuta en un servidor o contenedor JEE, especialmente diseñado para ofrecer contenido dinámico desde un servidor web, generalmente HTML[18]. Otras opciones que permiten generar contenido dinámico son los lenguajes ASP, PHP, Ruby, Python y JSP un caso especial de servlet. Forman parte de Java Enterprise Edition, que es una ampliación de Java Standard Edition.

Un servlet implementa la interfaz *javax.servlet.Servlet* o hereda alguna de las clases más convenientes para un protocolo específico, por ejemplo: *javax.servlet.HttpServlet*. Al implementar esta interfaz el servlet es capaz de interpretar los objetos de tipo *HttpServletRequest* y *HttpServletResponse* quienes contienen la información de la página que invocó al servlet.

Entre el servidor de aplicaciones y el servlet existe un contrato que determina cómo han de interactuar. La especificación de éste se encuentra en los JSR (*Java Specification Requests*[18]) del JCP (*Java Community Process*).

[1] Sun Microsystems fue una empresa informática recientemente (2009) adquirida por Oracle Corporation.

[4] Proceso formalizado que permite involucrarse en la definición de futuras características de la plataforma Java.

Ciclo de vida

El ciclo de vida de un servlet se divide en los siguientes puntos:

1. El cliente solicita una petición a un servidor vía URL.
2. El servidor recibe la petición.
   - Si es la primera, se utiliza el motor de servlets para cargarlo y se llama al método *init().*
   - Si ya está iniciado, cualquier petición se convierte en un nuevo hilo. Un servlet puede manejar múltiples peticiones de clientes.
3. Se llama al método *service()* para procesar la petición devolviendo el resultado al cliente.
4. Cuando se apaga el motor de un servlet se llama al método *destroy()*, que lo destruye y libera los recursos abiertos.

1.8. JAVA DATABASE CONNECTIVITY

Java Database Connectivity, más conocida por sus siglas JDBC, es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo 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[15]) que pueden manejar.

Para utilizar una base de datos particular, el usuario ejecuta su programa junto con la biblioteca de conexión apropiada al modelo de su base de datos y accede a ella estableciendo una conexión; para ello provee el 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: consulta, actualización, creación, modificación y borrado de tablas, ejecución de procedimientos almacenados en la base de datos, etc.

JDBC ofrece el paquete "*java.sql*", en el que existen clases muy útiles para trabajar con bases de datos.

[18] Java Specification Requests, son documentos formales que describen las especificaciones y tecnologías propuestas para ser añadidas a la plataforma Java.

[15] Uniform Resource Locator, es una secuencia de caracteres, de acuerdo a un formato modélico y estándar, que se usa para nombrar recursos en la red para su localización o identificación.

CLASE	DESCRIPCIÓN
DriverManager	Para cargar un driver.
Connection	Para establecer conexiones con las bases de datos.
Statement	Para crear consultas SQL y enviarlas a las BDD.
ResultSet	Para almacenar el resultado de la consulta.

Fuente: [W04]

Tabla 1.1. Clases principales de la librería JDBC.

[W04] Documentación Java, Tutorial Java EE6, *http://java.sun.com/javaee/6/docs/tutorial/doc/docinfo.html.*

CAPÍTULO II

JSP

Java Server Pages

INTRODUCCIÓN A JSP

FUNCIONAMIENTO DE JSP

COMPONENTES JSP

OBJETOS IMPLÍCITOS

BIBLIOTECAS DE ETIQUETAS

2.1. INTRODUCCIÓN A JSP

JavaServer Pages (JSP) es una tecnología Java desarrollada por la compañía Sun Microsystems[1], que permite generar contenido dinámico para la web, en forma de documentos HTML[2]. Los archivos JSP combinan HTML[2] con etiquetas especiales y fragmentos de código Java. La Especificación JSP 1.2 fue la primera que se liberó y en la actualidad está disponible la Especificación JSP 2.2.

Las JSPs permiten la utilización de código Java mediante scripts. Además es posible utilizar algunas acciones JSP predefinidas mediante etiquetas. Estas etiquetas pueden ser enriquecidas mediante la utilización de Librerías de Etiquetas (*Tag Libs o Tag Libraries*) externas e incluso personalizadas.

JSP puede considerarse como una manera alternativa, y simplificada, de construir servlets. Es por esto que una página puede hacer todo lo que un servlet puede hacer, y viceversa. Cada versión de la especificación de JSP está fuertemente vinculada a una versión en particular de la especificación de servlets.

La principal ventaja de JSP frente a otros lenguajes es que el lenguaje Java es un lenguaje de propósito general que excede el mundo web y que es apto para crear clases que manejen lógica de negocio y acceso a datos de una manera prolija. Esto permite separar en niveles las aplicaciones web, dejando la parte encargada de generar el documento HTML[2] en el archivo JSP.

Otra ventaja, es que JSP hereda la portabilidad de Java, y es posible ejecutar las aplicaciones en múltiples plataformas sin cambios. Es común incluso que los desarrolladores trabajen en una plataforma y que la aplicación termine siendo ejecutada en otra.

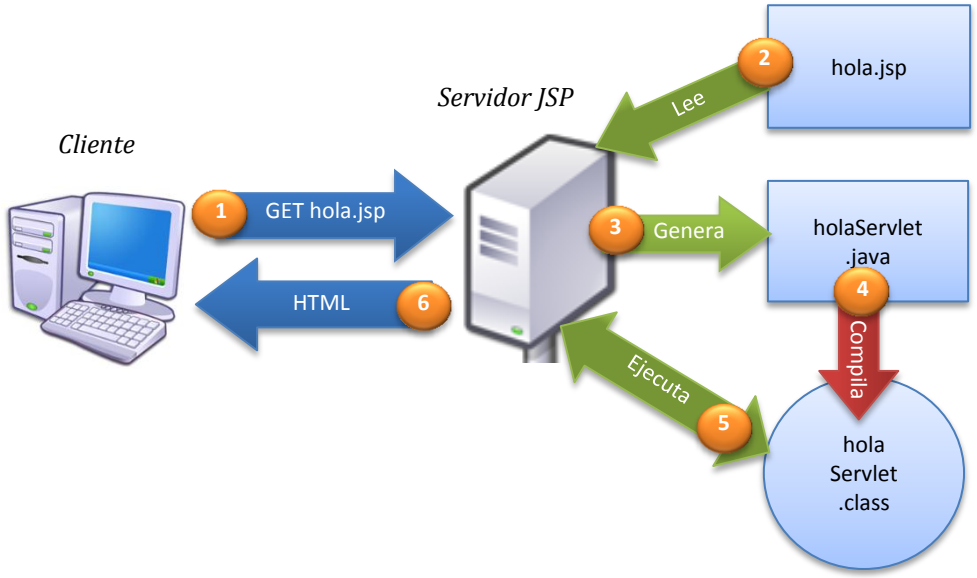
Los servlets y JavaServer Pages son dos métodos de creación de páginas web dinámicas en servidor usando el lenguaje Java. En ese sentido son similares a otros métodos o lenguajes tales como el PHP, ASP o los CGIs (*Common Gateway Interface*[3]), programas que generan páginas web en el servidor. Sin embargo, se diferencian de ellos en otras cosas.

Para empezar, los JSP's y servlets se ejecutan en una máquina virtual Java, lo cual permite que, en principio, se puedan usar en cualquier tipo de ordenador, siempre que exista una máquina virtual Java para él. Cada servlet o JSP se ejecuta en su propio contexto; pero no se comienza a ejecutar cada vez que recibe una petición, sino que persiste de una petición a la siguiente, de forma que no se pierde tiempo en invocarlo. Su persistencia le permite también hacer una serie de cosas de forma más eficiente, como por ejemplo: la conexión a bases de datos y la administración de sesiones.

[1] Sun Microsystems fue una empresa informática recientemente (2009) adquirida por Oracle Corporation.
[2] HyperText Markup Language, es el lenguaje de marcado predominante para la elaboración de páginas web.
[3] Es una importante tecnología que permite a un cliente solicitar datos de un programa ejecutado en un servidor web.

2.2. FUNCIONAMIENTO DE JSP

El funcionamiento general de la tecnología JSP es que el servidor de aplicaciones interpreta el código contenido en la página JSP para construir el código Java del servlet a generar. Este servlet será el que genere el documento (típicamente HTML) que se presentará en la pantalla del navegador del usuario.



Fuente: [Propia]

Figura 2.1. Funcionamiento de JSP.

Las JSP's son en realidad servlets, un JSP se compila a un programa en Java la primera vez que se invoca y del programa en Java se crea una clase que se empieza a ejecutar en el servidor como un servlet. La principal diferencia entre los servlets y las JSP's es el enfoque de la programación. Un JSP es una página web con etiquetas especiales y código Java incrustado, mientras que un servlet es un programa que recibe peticiones y genera a partir de ellas una página web.

El rendimiento de una página JSP es el mismo que tendría el servidor equivalente, ya que el código es compilado como cualquier otra clase Java. A su vez, la máquina virtual compilará dinámicamente a código de máquina las partes de la aplicación que lo requieran. Esto hace que JSP tenga un buen desempeño y sea más eficiente que otras tecnologías web que ejecutan el código de una manera puramente interpretada.

2.3. COMPONENTES JSP

Los componentes que podemos utilizar en una página JSP para hacerla dinámica son:

2.3.1. Expresiones

Son fragmentos de código Java, con la forma *<%= expresión %>* que se evalúan y se muestran en la salida del navegador. En general, dentro de una expresión podemos usar cualquier cosa que usaríamos dentro de un *System.out.print()*. Ejemplos:

```
<%= "Tamaño de cadena: "+cadena.length() %>
<%= new java.util.Date() %>
<%= Math.PI*2 %>
```

2.3.2. Declaraciones

Contienen declaraciones de variables o métodos, con la forma *<%! declaración %>*. Estas declaraciones serán accesibles desde cualquier lugar de la página JSP. Hay que tener en cuenta que el servidor transforma la página JSP en un servlet, y éste es usado por múltiples peticiones, lo que provoca que las variables conserven su valor entre sucesivas ejecuciones. Ejemplos:

```
<%! int numeroAccesos=0; %>
<html>
   <body>
      <%= "La página ha sido accedida " + (++numeroAccesos) + " veces desde el
      arranque del servidor"%>
   </body>
</html>
```

```
<%! java.util.Date primerAcceso=new java.util.Date(); %>
<html>
   <body>
      El primer acceso a la página se realizó en: <%= primerAcceso %>
   </body>
</html>
```

```
<%! private String ahora(){
   return ""+new java.util.Date();
}%>
<html>
   <body>
      <%= ahora() %>
   </body>
</html>
```

2.3.3. Scriplets

Son fragmentos de código Java con la forma *<% código %>*, en general podemos insertar cualquier código que pudiéramos usar dentro de una función Java. Para acceder a la salida del navegador, usamos el objeto implícito *out*. Ejemplos:

```
<%
out.println ("<table>");
for (int i=0;i<10;i++)
   out.println("<tr><td>"+i+"</td></tr>");
out.println("</table>");
%>
```

```
<table>
   <% for (int i=0; i<10; i++){ %>
      <tr><td> <%=i%> </td></tr>
   <%}%>
</table>
```

Si observamos los dos ejemplos anteriores hacen lo mismo, podría parecer que la segunda opción es más deseable, pero en general hay que evitar el uso de *out.println()* para elementos HTML. En un proyecto en el que trabajen programadores y diseñadores conjuntamente, hay que separar presentación y código tanto como sea posible.

Dentro de un scriptlet podemos usar cualquier librería de Java, lo cual hace que resulte muy sencillo construir interfaces web de entrada y salida para nuestras clases. Ejemplo:

```
<%
String parametro1=request.getParameter("parametro1");
String parametro2=request.getParameter("parametro2");
MiClase miClase=new MiClase();
String salida=miClase.procesa(parametro1, parametro2);
%>
<%= salida %>
```

2.3.4. Directivas

Las directivas son elementos que proporcionan información al motor JSP e influirán en la estructura del servlet generado. Hay tres tipos de directivas: *page*, *taglib* e *include*.

- ***page.*** Se indica con forma *<%@ page atributo="valor">*. Tiene diversos usos, entre los cuales destacaremos:

  Importar código, de la misma forma como se realiza en un programa Java, se indica con el atributo *import*. Ejemplo:

  ```
  <%@page import="java.io.*, miPackage.miClase"%>
  ```

  Gestionar errores permitiendo redireccionar a una página cuando se produzca un error, se indica con los atributos *errorPage* y *isErrorPage*. Ejemplo:

  ```
  <%@page errorPage="error.jsp">
  [...]
  <%@page isErrorPage="yes">
  <html>
     <body>
        Error, contacte con el administrador [...]
     </body>
  </html>
  ```

Indicar si la página tendrá acceso a la sesión. Se especifica con el atributo *session*. Ejemplo:

```
<%@page session="true" import="java.util.ArrayList"%>
```

- ***include.*** Permite incluir un archivo en el lugar donde se especifique, la directiva *include* copia el contenido del archivo byte a byte, siendo el resultado similar a si copiáramos el texto del archivo incluido y lo pegáramos en el JSP. Ejemplo:

```
<html>
   <head>
      <%@ include file="titulo.txt"%>
   </head>
   <body>
      <%@ include file="cuerpoPagina.jsp"%>
   </body>
</html>
```

- ***taglib.*** Se emplea para indicar que se van a emplear librerías de etiquetas. Ejemplo:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

2.3.5. Etiquetas

Las etiquetas tienen la forma *<jsp:etiqueta atributos />*, y tienen diversos usos, entre los que destacan la inclusión de páginas, transferencia de control y comunicarse con los JavaBeans de forma que el código de la página JSP sea lo más parecido al html. Ejemplos:

2.3.6. Comentarios

Para introducir comentarios en JSP, usaremos las marcas *<%-- comentario --%>*, dentro de un scriptlet o declaración podemos usar comentarios siguiendo la sintaxis de Java.

```
<%-- Comentario JSP --%>
<!-- Comentario HTML -->
<%
// Comentario
/* Comentario */
%>
```

2.4. OBJETOS IMPLÍCITOS

En JSP disponemos de algunos objetos implícitos, que nos permitirán acceder a diferente información y realizar diversas acciones. En JSP tenemos los siguientes objetos implícitos: *request*, *response*, *out*, *session*, *application*, *config*, *pageContext* y *page*.

2.4.1. request

Es un objeto de la clase *HttpServletRequest*, su uso principal es el acceso a los parámetros de la petición. Destacaremos las siguientes funciones:

- ***String getParameter(String name).-*** Devuelve el valor de un parámetro.
- ***Enumeration getParameterNames().-*** Devuelve una enumeración con los nombres de todos los parámetros de la petición.
- ***String[] getParameterValues(String name).-*** Los parámetros pueden tener valor múltiple, con esta función recuperamos un array con todos los valores para un nombre.
- ***String getRemoteAddr().-*** Devuelve la IP del host desde el que se realiza la petición.
- ***String getRemoteHost().-*** Devuelve el nombre del host desde el que se realiza la petición.

Ejemplo:

```
<html>
   <body>
      <form>
         <input type="text" name="parametro"/>
         <input type="submit"/>
      </form>
      <br>
      <br>
      Su IP: <%=request.getRemoteAddr()%>
      <br>
      Su nombre de host: <%= request.getRemoteHost() %>
      <br>
      Valor del parámetro: <%= request.getParameter("parametro") %>
   </body>
</html>
```

2.4.2. reponse

Es un objeto de la clase *HttpServletResponse*, que asiste al servlet en su generación de la respuesta para el cliente, contiene funciones para manejo de cabeceras, códigos de estado, cookies y transferencia de control.

2.4.3. out

Es un objeto de la clase *JspWriter*, es el que nos permite acceder a la salida del navegador desde los scriplet. Ejemplo:

```
<%
out.print("cadena");
out.println("cadena");
%>
```

2.4.4. session

Es un objeto de la clase *HttpSession*, nos permite acceder a la sesión asociada a la petición. A través de este objeto podemos, entre otras cosas, guardar objetos que serán accesibles desde cualquier JSP de la sesión o invalidarla.

Para guardar y recuperar información usaremos:

```
Object session.getAttribute("clave");
void session.setAttribute("clave", Object objeto);
```

Y para invalidar la sesión:

```
void session.invalidate();
```

Ejemplo:

```
<%@ page session="true" %>
<%
java.util.ArrayList accesos =
(java.util.ArrayList)session.getAttribute("accesos");
if (accesos==null)
   accesos=new java.util.ArrayList();
accesos.add(new java.util.Date().toString());
session.setAttribute("accesos", accesos);
if (request.getParameter("invalidaSesion")!=null)
   session.invalidate();
%>
<html>
   <body>
      <form>
         <input type="submit" name="invalidaSesion" value="Invalidar sesión"/>
         <input type="submit" value="Recargar página"/>
      </form>
      <br/>
      Usted accedió a esta página en los siguientes momentos: <br>
      <% for (int i=0;i<accesos.size();i++){ %>
         <%= accesos.get(i) %>
      <% } %>
   </body>
</html>
```

2.4.5. application

Es un objeto de la clase *ServletContext*. Este objeto es común para toda la aplicación web y entre otras cosas, nos permite almacenar información que será accesible desde todas las páginas de la aplicación web, independientemente de la sesión. Ejemplo:

```
Object application.getAttribute("clave");
void application.setAttribute("clave", Object objeto);
```

2.4.6. pageContext

Es un objeto de la clase *PageContext*. Entre otras cosas, nos permite almacenar información localmente a la página. Para guardar y recuperar valores:

```
Object pageContext.getAttribute("clave");
void pageContext.setAttribute("clave", Object objeto);
```

También podemos usar *PageContext* para almacenar y recuperar información en sesión y en aplicación:

Almacenar en contexto de página:

```
PageContext.setAttribute("clave", obj, PageContext.PAGE_SCOPE);
PageContext.setAttribute("clave", obj);
```

Almacenar en contexto de sesión:

```
PageContext.setAttribute("clave",obj,PageContext.SESSION _SCOPE);
session.setAttribute("clave", objeto);
```

Almacenar en contexto de aplicación:

```
PageContext.setAttribute("clave",obj,PageContext.APPLICATION_SCOPE);
application.setAttribute("clave",objeto);
```

2.4.7. config

Es un objeto de la clase *ServletConfig*. Permite acceder a parámetros de inicialización del servlet y a su contexto.

2.4.8. page

Es un sinónimo de *this*, no tiene utilidad en el estado actual de la especificación.

2.5. BIBLIOTECAS DE ETIQUETAS

La tecnología *JavaServer Pages Standard Tag Library* (*JSTL*) es un componente de Java EE que extiende las ya conocidas *JavaServer Pages* (*JSP*) proporcionando bibliotecas de etiquetas (*Tag Libraries*) con utilidades ampliamente utilizadas en el desarrollo de páginas web dinámicas.

Estas bibliotecas de etiquetas extienden de la especificación de JSP (la cual a su vez extiende de la especificación de Servlet). Su API nos permite además desarrollar nuestras propias bibliotecas de etiquetas.

La utilización de etiquetas en nuestras páginas JSP brinda ventajas como la reutilización del código y la interacción que tienen con los objetos implícitos; reduciendo la complejidad que podría llegar a tener una página JSP. Para un diseñador de páginas es muy útil contar con un grupo de etiquetas que se parezcan al HTML; facilitando el diseño y mantenimiento del proyecto.

2.5.1. Elementos de una biblioteca de etiquetas

Una biblioteca de etiquetas consta de tres partes importantes:

- **Clases de manipuladores de etiquetas.-** proporcionan la funcionalidad de las etiquetas.
- **Un descriptor de biblioteca de etiquetas *TLD*.-** describe las etiquetas y hace coincidir las etiquetas con las clases de manipuladores de etiquetas.
- **La directriz *taglib*.-** está es una directriz situada en la parte superior de su JSP, que le permite utilizar una biblioteca de etiquetas en particular.

2.5.1.1. Manipulador de etiquetas

Un manipulador de etiquetas es un tipo de clase especial que contiene el código que ejecuta una etiqueta; es decir, la funcionalidad de la etiqueta se encuentra dentro del manipulador de etiquetas. A diferencia de los JavaBeans, que son conjuntos de funciones comunes, un manipulador de etiquetas tiene una finalidad estrictamente definida, mantener una única etiqueta. Una biblioteca de etiquetas tiene un manipulador de etiquetas para cada etiqueta a medida.

El motor JSP necesita saber qué clases de manipuladores de etiquetas corresponden a unas etiquetas determinadas en una página JSP, y qué métodos llamar en esas clases. Esta información se almacena en un descriptor de la biblioteca de etiquetas.

2.5.1.2. Descriptor de biblioteca de etiquetas

Un descriptor de bibliotecas de etiquetas (*TLD*) es un documento *XML* que contiene información sobre una o más etiquetas a medida. Un *TLD* es un archivo que describe las etiquetas a medida y las relaciona con sus clases de manipuladores de etiquetas.

Como norma general, un archivo *TLD* se denomina como la clase a la que se refiere. Debe ser guardado con la extensión *".tld"* y almacenado en el directorio *WEB-INF* de la aplicación web.

2.5.1.3. Directriz *taglib*

Para utilizar las etiquetas de una biblioteca de etiquetas a medida en sus páginas JSP, debe añadir una directriz *taglib* en la parte superior de la página JSP. Por ejemplo:

```
<%@ taglib uri="http://jakarta.apache.org/taglibs/request-1.0" prefix="req" %>
```

Esta directriz se debe especificar en las páginas JSP para cada biblioteca de etiquetas que se desee utilizar, y los atributos *uri* y *prefix* tienen valores específicos para cada biblioteca de etiquetas.

El valor de un atributo *uri* identifica la biblioteca de etiquetas. A menudo consta de una URL, de la organización que mantiene la biblioteca de etiquetas. En realidad el sistema no intenta acceder a la URL, únicamente existe para identificar la biblioteca de etiquetas. La utilización de una URL simplemente ayuda a documentar sobre donde se originó la biblioteca de etiquetas y para garantizar que tiene un nombre único. Sin embargo, URI puede ser una ruta de acceso con un nombre de archivo añadido. En este caso, el sistema asume que este archivo es su archivo TLD e intenta cargarlo. Si la ruta de acceso no apunta al archivo TLD, el sistema buscará el archivo TLD en otro sitio de la aplicación web.

Siempre que un motor JSP encuentra una extensión de etiquetas en una JSP, mira el valor del atributo *uri* en la directriz *taglib* y después analiza sintácticamente el descriptor de la biblioteca de etiquetas para encontrar la clase de manipulación de etiquetas requerida. Entonces genera un código para que interactúe con el manipulador de etiquetas y le permita a su JSP utilizar las etiquetas.

El valor del atributo *prefix* se utiliza para identificar una etiqueta en una página JSP como parte de una biblioteca de etiquetas en particular. De esta forma, si tenemos dos etiquetas con el mismo nombre, pero procedentes de distintas bibliotecas de etiquetas, es posible identificarlas de forma única. Por lo tanto, si tuviéramos una etiqueta con el nombre *myTag* y un prefijo *myPrefix*, la etiqueta se escribiría de la siguiente forma:

```
<myPrefix:myTag />
```

No existen restricciones sobre lo que puede utilizar como valor de *prefix*, excepto que no puede utilizar *jsp, jspx, java, javax, servlet, sun* o *sunw* ya que son palabras reservadas.

2.5.2. Bibliotecas de etiquetas personalizadas

Para la implementación de una etiqueta personalizada existen dos etapas. Primero se debe crear una clase de manipulador de etiquetas y después presentarla a su correspondiente etiqueta personalizada utilizando un descriptor de bibliotecas de etiquetas.

Todos los manipuladores de etiquetas deben soportar ciertos métodos, siendo los más importantes: *doStartTag()* y *doEndTag()*. Cuando se lee la etiqueta de apertura, por ejemplo *<example:time>*, el contenedor llama al método *doStartTag()* del manipulación de etiquetas. Cuando se lee la correspondiente etiqueta de cierre *</example:time>*, el contenedor llama al método *doEndTag()*. Por lo general, crear un manipulador de etiquetas no es nada más que definir un código personalizado para que sea ejecutado por cada uno de estos métodos.

Para asegurarse de que el objeto manipulador de etiquetas contiene todos los métodos correctos, debe implementar una interfaz de manipuladores de etiquetas. La especificación 2.2 proporciona tres interfaces de manipuladores de etiquetas:

- Tag
- IterationTag
- BodyTag

2.5.2.1. La interfaz *Tag*

La interfaz Tag, es la interfaz de manipuladores de etiquetas básica que define los métodos requeridos por todos los manipuladores de etiquetas. La interfaz requiere seis métodos: *doStartTag(), doEndTag(), getParent(), setParent(), release()* y *setPageContext()*. Sin embargo, se acostumbra implementar solo uno o dos de estos métodos en su clase de manipuladores de etiquetas. Afortunadamente, existe una clase de ayuda, *TagSupport*, que ya implementa la interfaz *Tag*. Si su clase de manipulador de etiquetas extiende *TagSupport*, solo tenemos que implementar los métodos que necesitamos para conseguir nuestro objetivo.

Cuando extendemos la clase *TagSupport*, los dos métodos que normalmente invalidamos se relacionan con la JSP que lee la etiqueta personalizada.

Cuando se encuentra la etiqueta de apertura, se llama al método *doStartTag()*, el valor que devuelve este método determina lo que se hace después. Los valores válidos devueltos incluyen *EVAL_BODY_INCLUDE* o *SKIP_BODY*. La devolución de *EVAL_BODY_INCLUDE* hará que se evalúe el contenido del cuerpo de la etiqueta y que esté disponible para su uso en el manipulador de etiquetas. Si se devuelve *SKIP_BODY*, se salta el cuerpo de la etiqueta, por lo que debería ser utilizado en etiquetas que no tuvieran cuerpo o en etiquetas en las que no quisiéramos evaluar el cuerpo.

Cuando se encuentra la etiqueta de cierre, se llama al método *doEndTag()*. Si tiene una etiqueta vacía, este método sigue siendo llamado, pero será llamado justo después de *doStartTag()*. Valores válidos de devolución de este método son *EVAL_PAGE* o *SKIP_PAGE*. Si el método *doEndTag()* devuelve *EVAL_PAGE*, la ejecución de la JSP continúa normalmente después de la etiqueta personalizada. Sin embargo, si *doEndTag()* devuelve *SKIP_PAGE*, la JSP se detendrá después de evaluar la etiqueta personalizada. Esto quiere decir que se saltará completamente el resto de la página JSP. Se debe tener mucho cuidado al momento de devolver este valor, ya que el código que aparezca después de la etiqueta no será utilizado.

Para vincular los manipuladores de etiquetas con las etiquetas personalizadas necesitamos de un archivo descriptor de bibliotecas de etiquetas (TLD). Un archivo TLD tiene el siguiente aspecto aproximadamente:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library
1.2//EN" "http://java.sun.com/dtd/Web-jsptaglibrary_1_2.dtd">
```

```
<taglib>
…
</taglib>
```

Un TLD es un documento XML que describe su biblioteca de etiquetas personalizadas.

La primera parte de un TLD es siempre igual. Para amoldarse a un documento XML, la primera línea es siempre la declaración XML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

La siguiente línea comienza la etiqueta que une el documento XML con una definición y un mecanismo de validación para un TLD. Esto será igual para todos los archivos TLD de JSP.

```
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library
1.2//EN""http://java.sun.com/dtd/Web-jsptaglibrary_1_2.dtd">
```

El elemento raíz definido para un descriptor de bibliotecas de etiquetas es *<taglib>*. El elemento *<taglib>* debe aparecer después de los dos encabezamientos predeterminados y debe rodear el resto de los elementos que describen nuestra biblioteca de etiquetas.

```
<taglib>
…
</taglib>
```

El elemento *<taglib>* debe incluir algunos otros elementos para describir una biblioteca de etiquetas personalizada. A continuación, se listan algunos subelementos *<taglib>*.

ELEMENTO	DESCRIPCIÓN
<tlib-version>	La versión de la implementación de la biblioteca de etiquetas. Es arbitraria y establecida por los desarrolladores de una biblioteca de etiquetas.
<jsp-version>	La versión de la especificación JSP de la que depende la biblioteca de etiquetas.
<short-name>	Un nombre corto y fácil de recordar de una *taglib*.
<uri>	Un URI que identifica de forma única una *taglib*.
<display-name>	Un nombre corto para *taglib*, que se supone será mostrado por las herramientas.
<tag>	Soporta información de una única etiqueta en esta biblioteca de etiquetas. Se debe utilizar tantos elementos *<tag>* como etiquetas se describan en la *taglib*.
<description>	Una cadena que describe el propósito de la biblioteca de etiquetas.

Tabla 2.1. Elementos de la etiqueta *taglib.*

Cada etiqueta personalizada debe tener un elemento *<tag>* que la una a un manipulador de etiquetas. Todos los elementos *<tag>* deben contener, al menos, los dos elementos *<name>* y *<tag-class>* y dos etiquetas no pueden tener el mismo nombre. Por ejemplo:

```
<taglib>
   <tag>
      <name>time</name>
      <tag-class>coop.atu.timeTag</tag-class>
   </tag>
</taglib>
```

El elemento *<tag>* tiene subelementos adicionales que describen la etiqueta personalizada. Entre los más importantes tenemos:

ELEMENTO	REQ.	DESCRIPCIÓN
<name>	Si	Un nombre único para la etiqueta. Este nombre corresponde directamente con el subelemento *name* de la etiqueta JSP.
<tag-class>	Si	La clase de manipuladores de etiquetas que implementa la interfaz *Tag, IterationTag* o *BodyTag*.
<tei-class>	No	Una clase opcional de información extra de etiquetas.
<display-name>	Si	Un nombre corto para la etiqueta.
<description>	No	Una breve descripción de la etiqueta.
<attribute>	No	Información sobre un atributo utilizado por la etiqueta.
<variable>	No	Crea una variable scripting que puede ser utilizada por la etiqueta.

Tabla 2.2. Elementos de la etiqueta *tag.*

2.5.2.2. La interfaz *IterationTag*

La interfaz *IterationTag* es muy similar a la interfaz *Tag*, pero se utiliza cuando una etiqueta personalizada necesita re-evaluar su cuerpo repetidamente. Esta funcionalidad se utiliza a menudo para hacer bucles a través de una colección de objetos sin utilizar un scriplet. La *IterationTag* completa el bucle extendiendo la interfaz *Tag* e implementando un nuevo método, *doAfterBody()*. Este método es llamado solo si se devuelve *EVAL_BODY_INCLUDE* de *doStratTag()*.

El método *doAfterBody()* es llamado después de que se haya evaluado el cuerpo de una etiqueta. El valor devuelto de este método determinará si se deberá volver a evaluar el cuerpo. Si *doAfterBody()* devuelve el valor *EVAL_BODY_AGAIN*, entonces se debería volver a evaluar el cuerpo. Después de la evaluación se vuelve a invocar a *doAfterBody()* una y otra vez hasta esta devuelva el valor *SKIP_BODY* o *EVAL_BODY_INCLUDE*.

La creación de un manipulador de etiquetas que implemente *IterationTag* se hace extendiendo *TagSupport*, de igual manera que con la interfaz *Tag*.

2.5.2.3. La interfaz *BodyTag*

BodyTag es la interfaz más grande y más versátil de las tres interfaces de etiquetas. La interfaz *BodyTag* extiende la interfaz *IterationTag* y proporciona funcionalidad para manipular los contenidos de su cuerpo. Para comprender completamente la interfaz *BodyTag* primero necesitamos aprender el objeto *javax.servlet.jsp.tagext.BodyContent*.

El objeto *BodyContent* sostiene los resultados de evaluar el cuerpo de nuestra etiqueta personalizada. *BodyContent* tiene métodos para aclarar sus contenidos, leerlos y convertirlos en un *String*.

Podemos manipular el objeto *BodyContent* cada vez que hagamos bucles y reevaluemos el cuerpo de nuestra etiqueta. Esto continúa tanto tiempo como se necesite y termina cuando se devuelve *SKIP_BODY*.

No piense que las interfaces *BodyTag* e *IterationTag* son idénticas. Con *IterationTag*, existe una limitación en los cambios que hace al contenido el cuerpo. Con *BodyTag*, no existe tal restricción; los cambios pueden ser almacenados en un objeto *BodyContent* y utilizados como se desee.

Ahora tenemos un nuevo método disponible en la interfaz *BodyTag*, *doInitBody()*. Este método es invocado después de que se llame al método *doStartTag()* y solo si el valor *EVAL_BODY_BUFFERED* es devuelto por *doStartTag()*. En este caso se creará un nuevo objeto *BodyContent* para almacenar los resultados del cuerpo de la etiqueta. Entonces se volverá a usar y a evaluar el objeto *BodyContent*.

Existe otro objeto pre-construido llamado *BodyTagSupport*. La clase *BodyTagSupport* se utiliza al codificar un manipulador de etiquetas que implementa la interfaz *BodyTag*. De manera similar a como se extiende el objeto *TagSupport*, podemos extender también el objeto *BodyTagSupport* para hacer un manipulador de etiquetas.

```
public class exampleTag extends BodyTagSupport{
…
}
```

2.5.3. Funcionalidad de las etiquetas personalizadas

Existen tres principales maneras de extender la funcionalidad de las etiquetas personalizadas.

- Crear atributos
- Crear variables scripting
- Utillizar clase *Tag Extra Info* (TEI, Información extra de etiquetas)

2.5.3.1. Creación de atributos

El concepto de adición de atributos con manipuladores de etiquetas puede resultar novedoso, pero desde el punto de vista de la codificación, los atributos resultaran muy familiares. La clase

de manipuladores de etiquetas implementa un atributo utilizando un método setter del mismo nombre en su código de manipuladores de etiquetas.

Los atributos para una etiqueta tienen que ser primero declarados en el archivo TLD. El elemento *<attribute>* declarado en el archivo TLD es un subelemento de *<tag>*. Tiene que haber un elemento *<attribute>* para cada atributo de una etiqueta personalizada.

```
<tag>
   <attribute>
   ...
   </attribute>
</tag>
```

El elemento *<attribute>* tiene subelementos adicionales que contienen más información sobre el atributo, siendo el subelemento *name* necesario para la adición de un atributo.

ELEMENTO	DESCRIPCIÓN
<name>	El nombre de un atributo. Indispensable para <attribute>.
<required>	Especifica si se necesita el atributo para la etiqueta. Puede especificar true o yes para hacer que el atributo sea necesario. El valor false o no para indicar que el atributo es opcional. El valor por defecto es false.
<rtexprvalue>	Especifica si el atributo puede ser dinámicamente creado por un scriptlet en el tiempo de ejecución. Los valores disponibles para este elemento son true o yes para un valor de scriplet en tiempo de ejecución y false o no para rechazarlo.
<description>	Una breve descripción del atributo.

Tabla 2.3. Descripción de los elementos de la etiqueta *attribute.*

No existe restricción alguna en el número de atributos para una etiqueta personalizada. Para añadir atributos adicionales, solo tenemos que añadir el método setter apropiado en el manipulador de etiquetas y añadir el atributo a su TLD. Por defecto, cualquier atributo añadido no será necesario para la etiqueta; sin embargo, otros atributos podrían ser fundamentales para hacer que la etiqueta funcione.

En estos casos es necesario especificar que el atributo es necesario para la etiqueta en el TLD. Para requerir el valor de un atributo se debe insertar el elemento *<required>* como hijo del elemento *<attribute>*.

```
<attribute>
   <name>ejemplo</name>
   <required>true</required >
</attribute>
```

La utilización del elemento *<required>* y el establecimiento del cuerpo de este como *true* obligarán al usuario a colocar la información en el atributo antes de utilizar la etiqueta.

2.5.3.2. Creación de variables scripting

Uno de los usos más generalizados de las bibliotecas de etiquetas es eliminar scriptlets de las páginas JSP, pero las bibliotecas de etiquetas no se limitan a esta finalidad. Un manipulador de etiquetas puede también crear objetos y establecerlos en el objeto *PageContext*, estos pueden entonces ser utilizados por código scriptlet en su JSP.

Estos objetos creados por el manipulador de etiquetas se conocen como variables scripting. En efecto, proporcionan la misma funcionalidad que las etiquetas de acción, pero de una forma más flexible.

Existen dos formas de recuperar una variable scripting dentro de una JSP: declarándolas en el TLD o utilizando clases de ayuda *TagExtraInfo*.

Si se quiere crear una variable en el manipulador de etiquetas que sea accesible para el código scriptlet de la JSP, la forma más sencilla es declararla en el archivo TLD. Entre las ventajas que podemos encontrar, es que con una etiqueta de cooperación no necesita cooperar siempre con otras etiquetas; trabajar con un scriptlet es perfectamente válido.

Sin embargo, al hablar de scriptlet existen dos cuestiones que deberíamos considerar y mejorar. La primera es la forma en la que se introduce la variable scripting en la JSP. Las bibliotecas de etiquetas personalizadas pueden también imitar esta funcionalidad e introducir directamente variables scripting.

Otras justificaciones para eliminar el scriptlet son la reutilización y el mantenimiento. Cuando utilizamos scriptlets tenemos un código específico en cada JSP. A medida que pasa el tiempo, este código se hace difícil de mantener porque cada página con un scriptlet necesita ser modificada. La etiqueta personalizada pone todo nuestro código en un sitio y deja que la sintaxis de la etiqueta se utilice en la JSP.

El elemento *<variable>* en el TLD es un elemento de *<tag>* y declara variables scripting para la etiqueta a medida:

```
<tag>
   <variable>
   ...
   </variable>
</tag>
```

El elemento *<variable>* tiene elementos adicionales para más información. Todos los subelementos que se describen a continuación son necesarios, excepto la etiqueta *<name-given>*.

ELEMENTO	DESCRIPCIÓN
<name-given>	Contiene el nombre de la variable scripting.
<variable-class>	El valor de este elemento representa el tipo Java de la variable scripting.
<declared>	El valor de este elemento tiene que ser un argumento booleano de true o yes si la variable scripting es nueva.
<scope>	Hay tres valores disponibles que definen el ámbito de la variable scripting. Si el valor en AT_BEGIN, la variable scripting tiene alcance después de la etiqueta de apertura; AT_END después de la etiqueta de cierra y NESTED entre las etiquetas de apertura y de cierre.

Tabla 2.4. Subelementos de un elemento *variable*.

2.5.3.3. Utilizando clases *TagExtraInfo*

La declaración de una variable scripting mediante un TLD es simple y funciona, pero existe otro método. Cada etiqueta personalizada puede tener una clase de ayuda *TagExtraInfo* (TEI) declarada para acompañarla. Una ventaja de utilizar la clase TEI es la capacidad de crear variables scripting dinámicamente en vez de basarse en la información codificada en el TLD. Además de crear variables scripting, una clase TEI puede llevar a cabo también la validación, de una etiqueta personalizada.

Antes de entrar en detalles de cómo declarar una variable scripting utilizando una clase TEI, debemos entender dos objetos:

- ***TagData***, que soporta información sobre los atributos de la etiqueta.
- ***VariableInfo***, que describe una nueva variable scripting.

El objeto *TagData*

El objeto *TagData* proporciona información sobre los atributos de una etiqueta personalizada en el momento de la traducción; es generado de forma dinámica por el contenedor JSP basado en la etiqueta a medida correspondiente. Desde el punto de vista del desarrollador, solo utilizamos dicho objeto *TagData* para conseguir información sobre la etiqueta personalizada para la que se declaró el TEI.

El objeto *TagData* tiene algunos métodos diferentes que podemos utilizar para obtener información sobre los atributos de la etiqueta personalizada. El primero es el método *getAttributes()*, que devuelve un objeto denominado enumeración de la clase *java.util. Enumeration*. La numeración devuelta por *getAttributes()* contiene todos los atributos en el objeto *TagData*.

Otro método útil es *getAttributeString()*, que toma un *String* que representa el nombre de un atributo en particular como argumento y devuelve otro *String* que representa el valor del atributo. El método *getAttribute()* es una variación sobre este mismo tema, en el sentido de que toma como argumento el *String* del atributo *name*, pero devuelve el valor del atributo como un *Object*.

El objeto *VariableInfo*

El objeto *VariableInfo* describe una nueva variable scripting. Cuando creamos un objeto *VariableInfo* utilizamos la misma información que en el TLD, pero como parámetros de un constructor, por ejemplo:

```
VariableInfo vi =
new VariableInfo(varName, className, true, VariableInfo.AT_END)
```

El primer parámetro, *varName*, representa el nombre de la nueva variable scripting como un *String*. El segundo, *className*, es el nombre completo del tipo de la variable representada como una cadena. El tercero, en este caso *true*, es un valor booleano que indica si la variable scripting es nueva en la JSP y en consecuencia tiene que ser declarada. El último parámetro es un valor *integer* que representa el ámbito requerido de la variable scripting. Los valores disponibles se muestran en la tabla que aparece a continuación:

CAMPO	DESCRIPCIÓN
VariableInfo.AT_BEGIN	La nueva variable scripting estará en ámbito después de la etiqueta personalizada de apertura.
VariableInfo.AT_END	La nueva variable scripting estará en ámbito después de la etiqueta personalizada de cierre.
VariableInfo.NESTED	La nueva variable scripting estará solo en ámbito dentro del cuerpo de la etiqueta personalizada.

Tabla 2.5. Listado de valores disponibles para el objeto *VariableInfo.*

Las nuevas variables scripting son definidas mediante clases TEI utilizando el método *getVariableInfo()*. El método toma un objeto *TagData* como argumento y devuelve una matriz de objetos *VariableInfo*. Se crea una variable scripting para cada objeto *VariableInfo* en la matriz devuelta. Cada objeto *VariableInfo* se define a medida por el desarrollador de la JSP cuando se invalida este método en la clase TEI.

2.5.4. Empaquetación de bibliotecas de etiquetas

Un archivo de Java JAR es un método práctico para empaquetar juntos muchos archivos de clase en un archivo principal. Un JAR es un archivo que utiliza compresión ZIP para combinar muchos recursos como archivos de clase y descriptores de bibliotecas de etiquetas. En el caso de bibliotecas de etiquetas personalizadas, se utiliza un archivo JAR para combinar todas las clases de manipuladores de etiquetas y los archivos TLD apropiados.

El empaquetar una biblioteca de etiquetas personalizadas en un JAR es ideal para la portabilidad. Para instalar un JAR se necesita colocar el archivo JAR en el directorio *WEB-INF/lib* de la aplicación JSP y recargar el contenedor de servlets.

Actualmente existen dos formas de hacer referencia a archivos TLD dentro de un archivo JAR. La primera soporta la utilización de un solo archivo TLD en un JAR. El archivo TLD debe

llamarse *taglib.tld* y ser colocado en el directorio *META-INF* de su JAR. Entonces se le hace referencia desde las JSP utilizando la siguiente directriz de página, donde el *uri* debe apuntar a la ubicación de su archivo JAR.

```
<%@ taglib prefix=”ejemplo” uri=”WEB-INF/lib/ejemplo.jar” %>
```

A partir de la especificación JSP 1.2, se implementa un nuevo método; en el cual se puede empaquetar cualquier número de bibliotecas de etiquetas en el mismo JAR, siempre que cada una de ellas tenga los archivos de clase de manipuladores de etiquetas adecuados y un TLD en el directorio *META-INF* con la extensión ".tld". Además, cada TLD debe especificar valores *uri* diferentes. No puede haber nunca dos bibliotecas de etiquetas a las que se haga referencia con el mismo URI.

CAPÍTULO III

XML

eXtensible Markup Language

INTRODUCCIÓN

ORÍGENES DEL LENGUAJE XML

CARACTERÍSTICAS DEL ESTÁNDAR XML

VENTAJAS DEL ESTÁNDAR XML

ESTRUCTURA DE UN DOCUMENTO XML

3.1. INTRODUCCIÓN

XML (*eXtensible Markup Language* o *Lenguaje de marcas extensible*), es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium*[1] (*W3C*). Es una simplificación y adaptación del SGML[2] (*Standard Generalized Markup Language*) y permite definir la gramática de lenguajes específicos, de la misma manera que HTML[3] es a su vez un lenguaje definido por SGML[2]. Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML[4].

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

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

3.2. ORÍGENES DEL LENGUAJE XML

XML proviene de un lenguaje inventado por IBM[5] en los años setenta, llamado GML (*Generalized Markup Language*), que surgió por la necesidad que tenía la empresa de almacenar grandes cantidades de información. Este lenguaje gustó a la ISO[6], por lo que en 1986 trabajaron para normalizarlo, creando SGML[2], capaz de adaptarse a un gran abanico de problemas. A partir de él se han creado otros sistemas para almacenar información.

En el año 1989, Tim Berners Lee[7] creó la web, y junto con ella el lenguaje HTML. Este lenguaje se definió en el marco de SGML[2] y fue de lejos la aplicación más conocida de este estándar. Los navegadores web sin embargo siempre han puesto pocas exigencias al código HTML que interpretan y así las páginas web son caóticas y no cumplen con la sintaxis. Estas páginas web dependen fuertemente de una forma específica de lidiar con los errores y las ambigüedades, lo que hace a las páginas más frágiles y a los navegadores más complejos.

Otra limitación del HTML es que cada documento pertenece a un vocabulario fijo, establecido por el DTD[8]. No se pueden combinar elementos de diferentes vocabularios. Asi mismo es imposible para un intérprete, por ejemplo un navegador, analizar el documento sin tener

[1] World Wide Web Consortium (W3C), consorcio internacional que produce recomendaciones para World Wide Web.
[2] Standard Generalized Markup Language consiste en un sistema para la organización y etiquetado de documentos.
[3] HyperText Markup Language, es el lenguaje de marcado predominante para la elaboración de páginas web.
[4] Ejemplo de languajes basados en eXtensible Markup Language.
[5] International Business Machines, empresa multinacional estadounidense de tecnología con sede en Armonk, Nueva York.
[6] Organización Internacional de Normalización, es el organismo encargado de promover el desarrollo de normas internacionales de fabricación, comercio y comunicación para todas las ramas industriales a excepción de la electrónica.
[7] Tim Berners Lee nació el 8 de junio de 1955 en Londres, Reino Unido, creador del lenguaje HTML.
[8] Document Type Definition, es una descripción de estructura y sintaxis de un documento XML o SGML.

conocimiento de su gramática (DTD[8]). Por ejemplo, el navegador sabe que antes de una etiqueta * * debe haberse cerrado cualquier *<p>* previamente abierto. Los navegadores resolvieron esto incluyendo lógica ad hoc para el HTML, en vez de incluir un analizador genérico. Ambas opciones, de todos modos, son muy complejas para los navegadores.

Se buscó entonces definir un subconjunto del SGML[2] que permita:

- Mezclar elementos de diferentes lenguajes. Es decir que los lenguajes sean extensibles.
- La creación de analizadores simples, sin ninguna lógica especial para cada lenguaje.
- Empezar de cero y hacer hincapié en que no se acepte nunca un documento con errores de sintaxis.

Para hacer esto XML deja de lado muchas características de SGML[2] que estaban pensadas para facilitar la escritura manual de documentos. XML en cambio está orientado a hacer las cosas más sencillas para los programas automáticos que necesiten interpretar el documento.

3.3. CARACTERÍSTICAS DEL ESTÁNDAR XML

El estándar XML se define como *"el formato universal para documentos y datos estructurados en Internet"*, y sus principales características de funcionamiento, tal y como la propia W3C lo describe, son:

- **XML es un estándar para escribir datos estructurados en un fichero de texto.** Por datos estructurados entendemos tipos de documentos que van desde las hojas de cálculo, o las libretas de direcciones de internet, hasta parámetros de configuración, transacciones financieras o dibujos técnicos. Los programas que los generan, utilizan normalmente formatos binarios o de texto. XML es un conjunto de reglas, normas y convenciones para diseñar formatos de texto para tales tipos de datos, de forma que produzca ficheros fáciles de generar y de leer, que carezcan de ambigüedades y que eviten problemas comunes, como la falta de extensibilidad, carencias de soporte debido a características de internacionalización, o problemas asociados a plataformas específicas.
- **XML parece HTML pero no lo es.** En efecto, en XML se usan marcas y atributos, pero la diferencia estriba en que, mientras en HTML cada marca y atributo está establecido mediante un significado, en XML sólo se usan las marcas para delimitar fragmentos de datos, dejando la interpretación de éstos a la aplicación que los lee.
- **XML está en formato texto, pero no para ser leído.** Esto le da innumerables ventajas de portabilidad, depuración, independencia de plataforma, e incluso de edición, pero

[8] Document Type Definition, es una descripción de estructura y sintaxis de un documento XML o SGML.
[2] Standard Generalized Markup Language consiste en un sistema para la organización y etiquetado de documentos.

su sintaxis es más estricta que la de HTML; una marca olvidada o un valor de atributo sin comillas convierten el documento en inutilizable. No hay permisividad en la construcción de documentos, ya que esa es la única forma de protegerse contra problemas más graves.

- **XML consta de una familia de tecnologías.** Por supuesto, existe una definición de XML 1.0 que viene desde Febrero de 1998, pero su desarrollo se ha ido enriqueciendo paulatinamente a medida que se veían sus posibilidades. De esa forma, contamos con una especificación Xlink[9], que describe un modo estándar de añadir hipervínculos a un documento XML. XPointer y XFragments[10] son especificaciones para establecer la forma de vincular partes de un documento XML. Incluso el lenguaje de hojas de estilo CSS[11], se puede utilizar con XML al igual que se hace con HMTL. XSL[12] es precisamente, una extensión del anterior, en la que se dispone de todo un lenguaje de programación exclusivamente para definir criterios de selección de los datos almacenados en un documento XML, y que funciona conjuntamente con las CSS o con HTML para suministrar al programador y al usuario mecanismos de presentación y selección de información, que no requieran de la intervención constante del servidor. Se basa en un lenguaje anterior para transformación (XSLT[13]) que permite modificar atributos y marcas de forma dinámica.
- **El Modelo de Objetos de Documento (DOM).** es un conjunto estándar de funciones para manipular documentos XML y HMTL, mediante un lenguaje de programación. XML-Namespaces, es una especificación que describe cómo puede asociarse una URL a cada etiqueta de un documento XML, otorgándoles un significado adicional. Y finalmente, XML-Schemas es un modo estándar de definir los datos incluidos en un documento de forma más similar a la utilizada por los programadores de bases de datos, mediante los metadatos asociados. Y hay otros en desarrollo, pero todos están basados en el principal: XML.
- **XML es prolijo, pero eso no supone un problema.** Los ficheros resultantes, son casi siempre mayores que sus equivalentes binarios. Esto es intencionado, y las ventajas ya las hemos comentado más arriba, mientras que las desventajas, siempre pueden ser soslayadas mediante técnicas de programación. Dado el reducido coste actual del espacio en disco y la existencia gratuita de utilidades de compresión, junto al hecho de

[9] Lenguaje de vínculos XML que permite crear elementos XML que describen relaciones cruzadas entre archivos de la red.
[10] Lenguajes basados en XML que son un estándar del World Wide Web Consortium.
[11] Cascading Style Sheets, lenguaje usado para definir la presentación de un documento escrito en HTML o XML.
[12] Extensible Stylesheet Language, es una familia de lenguajes basados en el estándar XML que permite describir cómo la información contenida en un documento XML cualquiera debe ser formateada para su presentación en un medio.
[13] Extensible Stylesheet Language Transformations, presenta una forma de transformar documentos XML en otros.

que los protocolos de comunicación soportan sistemas rápidos de compresión, este aspecto no debe resultar problemático.

- **XML no requiere licencias,** es independiente de la plataforma y tiene un amplio soporte. La selección de XML como soporte de aplicaciones, significa entrar en una comunidad muy amplia de herramientas y desarrolladores, y en cierto modo se parece a la elección de SQL respecto a las bases de datos. Todavía hay que utilizar herramientas de desarrollo, pero la tranquilidad del uso del estándar y de su formato, hacen que las ventajas a la larga sean notables.

3.4. VENTAJAS DEL ESTÁNDAR XML

Las principales ventajas del XML, son las siguientes:

- Es extensible, después de diseñado y puesto en producción, es posible extender XML con la adición de nuevas etiquetas, de modo que se pueda continuar utilizando sin complicación alguna.
- El analizador es un componente estándar, no es necesario crear un analizador específico para cada versión de lenguaje XML. Esto posibilita el empleo de cualquiera de los analizadores disponibles. De esta manera se evitan bugs y se acelera el desarrollo de aplicaciones.
- Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarla. Mejora la compatibilidad entre aplicaciones. Podemos comunicar aplicaciones de distintas plataformas, sin que importe el origen de los datos, es decir, podríamos tener una aplicación en Linux con una base de datos Postgres y comunicarla con otra aplicación en Windows y base de datos MS-SQL Server.
- Transformamos datos en información, pues se le añade un significado concreto y los asociamos a un contexto, con lo cual tenemos flexibilidad para estructurar documentos.

3.5. ESTRUCTURA DE UN DOCUMENTO XML

La tecnología XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y que esas partes se componen a su vez de otras partes. Estas partes se llaman elementos, y se las señala mediante etiquetas.

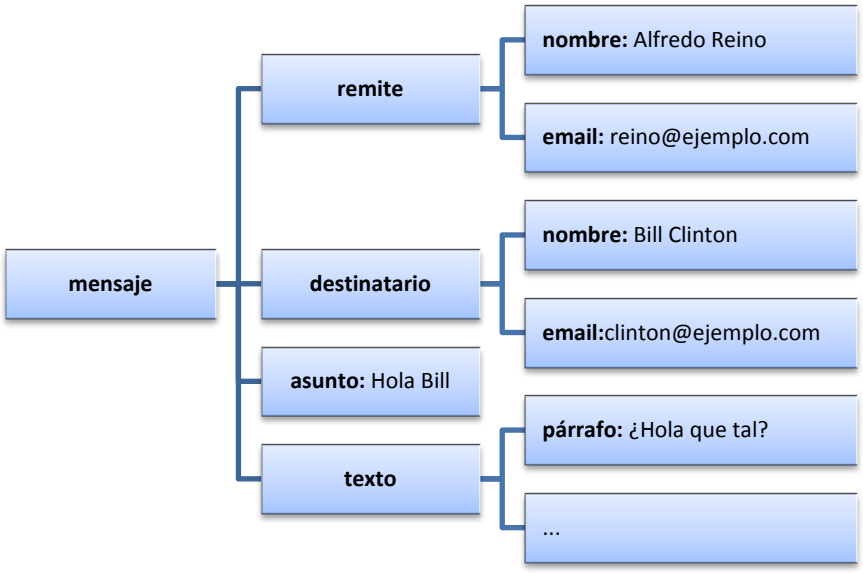
Una etiqueta consiste en una marca hecha en el documento, que señala una porción de éste como un elemento. Las etiquetas tienen la forma <nombre>, donde nombre es el nombre del elemento que se está señalando.

Aunque a primera vista, un documento XML puede parecer similar a HTML, hay una diferencia principal. Un documento XML contiene datos que se autodefinen, exclusivamente. Un documento HTML contiene datos mal definidos, mezclados con elementos de formato. En XML se separa el contenido de la presentación de forma total.

A continuación se muestra un ejemplo para entender la estructura de un documento XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE MENSAJE SYSTEM "mensaje.dtd">
<mensaje>
   <remite>
      <nombre>Alfredo Reino</nombre>
      <email>alf@ibium.com</email>
   </remite>
   <destinatario>
      <nombre>Bill Clinton</nombre>
      <email>president@whitehouse.gov</email>
   </destinatario>
   <asunto>Hola Bill</asunto>
   <texto>
      <parrafo>
¿Hola qué tal? Hace <enfasis>mucho</enfasis> que no escribes. A ver si llamas
y quedamos para tomar algo.
      </parrafo>
   </texto>
</mensaje>
```

Este mismo documento puede ser visto de forma gráfica, para comprender mejor la estructura de un documento XML.



Fuente: [Propia]

Figura 3.1. Ejemplo de la estructura de un documento XML.

3.5.1. Documentos XML bien formados

Los documentos denominados como *"bien formados"*, son aquellos que cumplen con todas las definiciones básicas de formato y pueden, por lo tanto, analizarse correctamente por cualquier analizador sintáctico que cumpla con la norma.

3.5.1.1. Estructura jerárquica de elementos

Los documentos XML deben seguir una estructura estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente incluida en otra, es decir, las etiquetas deben estar correctamente anidadas. Además, los elementos con contenido deben estar correctamente cerrados. En el siguiente ejemplo, la primera línea sería incorrecta en XML, no así la segunda.

```
<LI>HTML <B>permite <I>esto</B></I>.
<LI>En XML la <B>estructura <I>es</I> jerárquica</B>.</LI>
```

3.5.1.2. Etiquetas vacías

HTML permite elementos sin contenido. XML también, pero la etiqueta debe ser de la siguiente forma. En el siguiente ejemplo, la primera línea sería incorrecta en XML, no así la segunda.

```
<LI>Esto es HTML<BR>en el que casi todo está permitido</LI>
<LI>En XML, somos<BR/> más restrictivos.</LI>
```

3.5.1.3. Un solo elemento raíz

Los documentos XML, sólo permiten un elemento raíz, del que todos los demás sean parte, es decir, la jerarquía de elementos de un documento XML bien-formado sólo puede tener un elemento inicial.

3.5.1.4. Valores de atributos

Los valores atributos en XML siempre deben estar encerrados entre comillas simples o dobles.

3.5.1.5. Tipo de letra, espacios en blanco

El XML es sensible al tipo de letra utilizado, es decir, trata las mayúsculas y minúsculas como caracteres diferentes. Si un elemento de XML está definido como "ELEMENTO", no podemos usar "elemento", ni "Elemento", ni "eleMENto" para referirnos a él.

Existe un conjunto de caracteres denominados "espacios en blanco" que los procesadores XML tratan de forma diferente en el marcado XML. Estos caracteres son los espacios, tabuladores, retornos de carro y los saltos de línea. La especificación XML 1.0 permite el uso de esos "espacios en blanco" para hacer más legible el código, y en general son ignorados por los procesadores XML.

En otros casos, sin embargo, los "espacios en blanco" resultan muy significativos, por ejemplo, para separar las palabras en un texto, o separar líneas de párrafos diferentes.

3.5.1.6. Nombrando cosas

Al utilizar XML, es necesario asignar nombres a las estructuras, tipos de elementos, entidades, elementos particulares, etc. En XML los nombres tienen algunas características en común.

Según la especificación XML 1.0, un nombre "empieza" con una letra o uno o más signos de puntuación y "continúa" con letras, dígitos, guiones, rayas, dos puntos o puntos, denominados de forma global como caracteres de nombre. Los nombres que empiezan con la cadena "xml", se reservan para la estandarización de esta o de futuras versiones de esta especificación.

Resumiendo, no se pueden crear nombres que empiecen con la cadena "xml", "xMl", "XML" o cualquier otra variante. Las letras y rayas se pueden usar en cualquier parte del nombre. También se pueden incluir dígitos, guiones y caracteres de punto, pero no se puede empezar por ninguno de ellos. El resto de caracteres, como algunos símbolos, y espacios en blanco, no se pueden usar.

3.5.1.7. Marcado y datos

Las construcciones como etiquetas, referencias de entidad y declaraciones se denominan "*marcas*". Éstas son las partes del documento que el procesador XML espera entender. El resto del documento que se encuentra entre las marcas, son los datos que resultan entendibles por las personas.

Es sencillo reconocer las marcas en un documento XML. Son aquellas porciones que empiezan con "<" y acaban con ">", o bien, en el caso de las referencias de entidad, empiezan por "&" y acaban con ";".

3.5.2. Partes de un documento XML

Un documento XML está formado por el prólogo y por el cuerpo del documento así como texto de etiquetas que contiene una gran variedad de efectos positivos o negativos en la referencia opcional a la que se refiere el documento, hay que tener mucho cuidado de esa parte de la gramática léxica para que se componga de manera uniforme.

3.5.2.1. Prólogo

Aunque no es obligatorio, los documentos XML pueden empezar con unas líneas que describen la versión XML, el tipo de documento y otras cosas.

El prólogo de un documento XML contiene:

- **Declaración XML.** Define la versión de XML usada. En la "declaración XML" especificamos la codificación del documento, que puede ser, por ejemplo: US-ASCII,

UTF-8, UCS-2, EUC-JP, Shift_JIS, Big5, ISO-8859-1 hasta ISO- 8859-7. Además, se puede incluir una declaración de documento autónomo, que controla qué componentes de la DTD son necesarios para completar el procesamiento del documento.

- **Declaración de tipo de documento.** Define el tipo de documento que estamos creando para ser procesado correctamente, es decir, definimos que DTD[8] (*Document Type Definition*) valida y define los datos que contiene nuestro documento XML. Aquí se define el tipo de documento, y dónde encontrar la información sobre su DTD[8], mediante un identificador público (PUBLIC) que hace referencia a dicha DTD8, o mediante un Identificador Universal de Recursos (URI[14]) precedido por la palabra SYSTEM.

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE MENSAJE SYSTEM "mensaje.dtd">
<!DOCTYPE HTML PUBLIC "-/ /W3C/ /DTD HTML 3.2 Final/ /EN">
<!DOCTYPE LABEL SYSTEM "http://www.empresa.com/dtds/label.dtd">
```

3.5.2.2. Cuerpo

A diferencia del prólogo, el cuerpo no es opcional en un documento XML, el cuerpo debe contener solo un elemento raíz, característica indispensable también para que el documento esté bien formado. Sin embargo es necesaria la adquisición de datos para su buen funcionamiento.

Elementos

Los elementos XML pueden tener contenido (más elementos, caracteres, o ambos a la vez) o bien ser elementos vacíos. Un elemento con contenido es, por ejemplo:

```
<nombre>Fulano Mengánez</nombre>
<aviso tipo="emergencia" gravedad="mortal">Que no cunda el pánico</aviso>
```

Siempre empieza con una *<etiqueta>* que puede contener atributos o no, y termina con una *</etiqueta>* que debe tener el mismo nombre. Al contrario que HTML, en XML siempre se debe "cerrar" un elemento.

Hay que tener en cuenta que el símbolo "<" siempre se interpreta como inicio de una etiqueta XML. Si no es el caso, el documento no estará bien-formado. Para usar ciertos símbolos se usan las entidades predefinidas, que se explican más adelante. Un elemento vacío, es el que no tiene contenido "claro". Por ejemplo:

[8] Document Type Definition, es una descripción de estructura y sintaxis de un documento XML o SGML.
[14] Uniform Resource Identifier, es una cadena de caracteres corta que identifica inequívocamente un recurso.

```
<identificador DNI="23123244"/>
<linea-horizontal/>
```

Al no tener una etiqueta de "cierre" que delimite un contenido, se utiliza la forma *<etiqueta/>*, que puede contener atributos o no. La sintaxis de HTML permite etiquetas vacías tipo *<hr>* o *<img src=" ">*; en HTML reformulado para que sea un documento XML bien-formado, se debería usar *<hr/>* o **.

Atributos

Como se ha mencionado antes, los elementos pueden tener atributos, que son una manera de incorporar características o propiedades a los elementos de un documento. Por ejemplo, un elemento "chiste" puede tener un atributo "tipo" y un atributo "calidad", con valores "vascos" y "bueno" respectivamente.

```
<chiste tipo="vascos" calidad="bueno"> Esto es un chiste
</chiste>
```

En una DTD, se especifican los atributos que puede tener cada tipo de elemento, así como sus valores y tipos de valor posible. Al igual que en otras cadenas literales de XML, los atributos pueden estar marcados entre comillas verticales (') o dobles ("). Cuando se usa uno para delimitar el valor del atributo, el otro tipo se puede usar dentro.

```
<verdura clase="zanahoria" longitud='15" y media'>
<cita texto="'Hola buenos dias', contesto">
```

A veces, un elemento con contenido, puede modelarse como un elemento vacío con atributos. Un concepto se puede representar de muy diversas formas, pero una vez elegida una, es aconsejable fijarla en el DTD, y usar siempre la misma consistentemente dentro de un documento XML.

```
<gato><nombre>Silvestre</nombre><raza>Persa</raza></gato>
<gato raza="Persa">Silvestre</gato>
<gato raza="Persa" nombre="Silvestre"/>
```

3.5.3. Estructuras XML

3.5.3.1. Entidades predefinidas

En XML, se definen cinco entidades para representar caracteres especiales y que no se interpreten como marcado por el procesador XML; es decir, que podemos usar el carácter "<" sin que se interprete como el comienzo de una etiqueta XML.

ENTIDAD	CARACTER
&amp;	&
&lt;	<
&gt;	>
&apos;	'
&quot;	"

Fuente: [W01]

Tabla 3.1. Entidades predefinidas en XML.

3.5.3.2. Secciones CDATA

Existe otra construcción en XML que permite especificar datos, utilizando cualquier carácter, especial o no, sin que se interprete como marcado XML. La razón de esta construcción llamada CDATA (*Character Data*) es que a veces es necesario para los autores de documentos XML, poder leerlo fácilmente sin tener que descifrar los códigos de entidades. Especialmente cuando son muchas. Ejemplo:

```
<ejemplo>
&lt;HTML>
&lt;HEAD>&lt;TITLE>Rock &amp; Roll&lt;/TITLE>&lt;/HEAD>
</ejemplo>
<ejemplo>
<![CDATA[
<HTML>
<HEAD><TITLE>Rock & Roll</TITLE></HEAD>
]]>
</ejemplo>
```

Como vemos, dentro de una sección CDATA podemos poner cualquier cosa, que no será interpretada como algo que no es. Existe empero una excepción, y es la cadena "]] >" con la que termina el bloque CDATA. Esta cadena no puede utilizarse dentro de una sección CDATA.

3.5.3.3. Comentarios

A veces es conveniente insertar comentarios en el documento XML, que sean ignorados por el procesado de la información y las reproducciones del documento. Los comentarios tienen el mismo formato que los comentarios de HTML. Es decir, comienzan por la cadena "<!--" y terminan con "-->".

[W01] Wikipedia, eXtensible Markup Language, *http://es.wikipedia.org/wiki/Extensible_Markup_Language*.

```
<?xml version="1.0"?>
<!-- Aquí va el tipo de documento -->
<!DOCTYPE EJEMPLO [
<!-- Esto es un comentario -->
<!ELEMENTO EJEMPLO (#PCDATA)>
<!-- ¡Eso es todo por ahora! -->
]>
<EJEMPLO>texto texto texto bla bla bla
<!-- Otro comentario -->
</EJEMPLO>
<!-- Ya acabamos -->
```

Se pueden introducir comentarios en cualquier lugar de la instancia o del prólogo, pero nunca dentro de las declaraciones, etiquetas, u otros comentarios.

CAPÍTULO IV

JasperReports

INTRODUCCIÓN

API DE JASPERREPORTS

PRINCIPALES TAREAS Y PROCESOS

DISEÑO DEL REPORTE

DATOS DEL REPORTE

SECCIONES DEL REPORTE

SCRIPTLETS

ELEMENTOS DEL REPORTE

SUBREPORTES

JASPERREPORTS AVANZADA

4.1. INTRODUCCIÓN

JasperReports es una poderosa herramienta para la generación de reportes en Java; con la habilidad de producir contenido amplio y completo para la pantalla, para la impresora o inclusive exportarlo a diferentes formatos de archivos como: PDF, XLS, CSV, HTML, XML, entre otros.

La librería está completamente escrita en Java y se puede utilizar en una variedad de aplicaciones Java, incluyendo JEE o aplicaciones web. Su principal objetivo es ayudar a generar de forma sencilla y flexible reportes listos para imprimir.

JasperReports organiza la información de acuerdo a la plantilla de diseño definida en un archivo XML. Dicha información puede provenir de diversas fuentes de datos como: bases de datos relacionales, colecciones o arrays de objetos Java, etc. Los usuarios pueden integrar la librería JasperReports con varias herramientas de programación, para personalizar los orígenes de datos y diseñar reportes a través de una interfaz más simple y amigable.

Con el fin de llenar un reporte con datos, el diseño en XML debe ser compilado. A través de la compilación, un objeto es generado y luego es serializado con el fin de almacenarlo en un disco o enviarlo a través de la red. Este objeto serializado se utiliza cuando una aplicación necesita llenar el reporte con datos. De hecho, la compilación del diseño, implica la compilación de todas las expresiones Java definidas en el archivo XML que representa el reporte. Varias validaciones se realizan durante la compilación, para comprobar la consistencia del diseño. El objeto generado está listo para ser llenado con los diferentes conjuntos de datos y generar documentos.

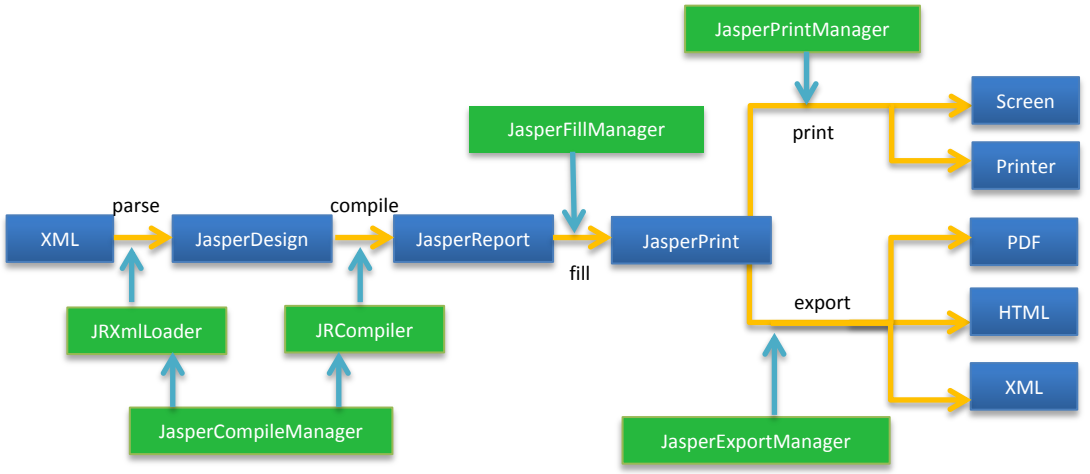
Para llenar un diseño, el motor de JasperReports necesita recibir los datos para el reporte. Estos datos pueden provenir de diversas fuentes, algunos se pueden pasar como parámetros del reporte, pero la mayoría de los datos se encuentran en las fuentes de datos definidas en el diseño. El motor generador de reportes puede recibir los datos directamente de las fuentes de datos o el reporte puede obtener directamente los datos que se encuentran en una base de datos relacional, incorporando al diseño un objeto de conexión.

El resultado de esta operación, es un nuevo objeto que representa el documento listo. Este también es serializado para almacenarlo en el disco o transferirlo a través de la red. Este documento se puede visualizar directamente en el visor incorporado de JasperReports, o se puede exportar a otros formatos más populares como: PDF, HTML o XML.

4.2. API DE JASPERREPORTS

La mayoría de veces, cuando se utiliza la librería JasperReports, los programadores trabajan con unas pocas clases y no llegan a conocer toda la API, para beneficiarse de toda la funcionalidad, características y ventajas de esta librería.

A continuación, se describe detalladamente las clases e interfaces más importantes que integran esta librería y exponer la forma de utilizarlas en el desarrollo de aplicaciones Java y aprovechar al máximo la funcionalidad y prestaciones que nos ofrece la API de JasperReports.



Fuente:[Propia]

Figura 4.1. Clases e interfaces principales de la librería JasperReports.

4.2.1. Clase *dori.jasper.engine.design.JasperDesign*

Comenzamos con esta clase porque las instancias de esta representan la materia prima que la librería JasperReports utiliza para fines de generación de reportes. Estas instancias se obtienen después de que el archivo de diseño XML es analizado por la librería XML interna de JasperReports. Entre las muestras suministradas que vienen con los archivos fuentes del proyecto, hay una llamada noxmldesign que se puede verificar para ver cómo crear dinámicamente un objeto *dori.jasper.engine.design.JasperDesign* sin necesidad de editar el archivo de diseño XML.

Todas las instancias de la clase *dori.jasper.engine.design.JasperDesign* están sujetas a la compilación antes de ser utilizadas por el proceso de llenado y generación de documentos. Esta es la razón por la que se considera la materia prima para la librería.

4.2.2. Clase *dori.jasper.engine.JasperReport*

Las instancias de esta clase representan los objetos compilados de los diseños. Estos sólo se pueden obtener como resultado del proceso de compilación de reportes de JasperReports y están listos para ser llenados con los datos y generar los documentos finales.

Mediante la compilación, y los diversos controles de consistencia y reorganización de los elementos del reporte para una utilización más rápida, la librería crea un archivo temporal de la clase que contiene todas las expresiones del reporte como son las expresiones de variables, campos de texto, imágenes, grupos, etc.

Este archivo fuente temporal de Java es compilado sobre la marcha utilizando las clases del compilador Java de la JDK, que se utilizará para ejecutar la aplicación. Si el archivo *tools.jar* no

se encuentra en el *classpath*, la compilación de todos modos seguirá adelante lanzándola al *runtime* del compilador *javac.exe*. Los *bytecode* de la clase resultante son almacenados en la colección de resultados de la instancia de la clase *dori.jasper.engine.JasperReport*, para luego ser llenados con los datos y evaluar las diversas expresiones del reporte en el *runtime*.

4.2.3. Clase *dori.jasper.engine.JasperCompileManager*

Esta es la clase que expone toda la funcionalidad de la librería en lo que respecta a la compilación de reportes. Existen varios métodos para la compilación de los archivos XML que representan los diseños, permitiendo a los usuarios utilizar el más adecuado a sus necesidades pudiendo obtener los datos de diversas fuentes como: archivos almacenados en disco, a través de flujos de entrada o a partir de bases de datos. También permite compilar en memoria diseños para directamente pasar un objeto *dori.jasper.engine.design.JasperDesign* y ser receptado por el correspondiente objeto *dori.jasper.engine.JasperReport*.

Otra utilidad, es la inclusión de métodos para la verificación del diseño del informe y la generación del diseño XML, para construirlo en memoria instanciando la clase *dori.jasper.engine.design.JasperDesign*. Esta es muy utilizada, especialmente en herramientas GUI que simplifican el trabajo del diseño de reportes.

4.2.4. Clase *dori.jasper.engine.JasperPrint*

Después de compilar un reporte está lleno con datos, el documento resultante viene como una instancia de la clase *dori.jasper.engine.JasperPrint*. Este tipo de objeto se puede visualizar directamente con el visor de reportes embebido en JasperReports o se puede serializar para almacenarlo en disco o para enviarlo a través de la red.

Las instancias de esta clase representan el resultado del proceso de llenado del reporte de la librería JasperReports y puede ser considerado como un formato personalizado para el almacenamiento de documentos completos. Ellos pueden ser transformados en otros formatos más populares como: PDF, HTML, XML u otros mediante el uso de las librerías de exportación.

4.2.5. Interfaz *dori.jasper.engine.JRDataSource*

JasperReports es una herramienta para la elaboración de reportes muy flexible en cuanto a las fuentes de datos. Que permite a los desarrolladores utilizar la fuente de datos que ellos deseen, siempre y cuando está proporcione una adecuada implementación de esta interfaz, de modo que el motor generador de reportes pueda interpretar y recuperar los datos de esa fuente de datos para el llenado de los reportes.

Normalmente, cada vez que un informe se está llenando, una instancia de esta interfaz es siempre suministrada o creada por debajo por el motor generador de reportes.

4.2.6. Clase *dori.jasper.engine.JRResultSetDataSource*

Esta es una implementación predeterminada de la interfaz *dori.jasper.engine.JRDataSource*. Dado que la mayoría de los reportes se generan a partir de datos que proviene de una base de datos relacional, JasperReports incluye por defecto esta implementación que envuelve un objeto *java.sql.ResultSet*.

Esta clase puede ser instanciada por los programadores, para cargar los datos existentes en una colección de resultados, antes de pasar estos datos a las rutinas de llenado del informe, pero esta también es utilizada por el motor generador de reportes para capturar los datos proporcionados por las bases de datos después de ejecutar la consulta del informe a través un conector JDBC.

4.2.7. Clase *dori.jasper.engine.data.JRTableModelDataSource*

Esta clase representa otra implementación por defecto de la interfaz *dori.jasper.engine.JRDataSource* que es proporcionada con la librería. Esta ocupa un objeto *javax.swing.table.TableModel* y puede ser utilizada en aplicaciones Swing de Java para generar reportes a partir de los datos que hayan sido cargados en tablas de la pantalla.

4.2.8. Clase *dori.jasper.engine.JREmptyDataSource*

Siendo la implementación más simple de la interfaz *dori.jasper.engine.JRDataSource*, esta clase puede ser utilizada en los reportes que no muestren los datos desde la fuente de datos proporcionada, sino más bien a partir de parámetros, y sólo cuando el número de filas virtuales en la fuente de datos es importante.

Muchos de los ejemplos que se proporcionan, tales como: fuentes, imágenes y formas utilizan una instancia de esta clase cuando llenan los reportes, para simular una fuente de datos con un registro en el mismo, pero con todos los campos nulos.

4.2.9. Clase *dori.jasper.engine.JasperFillManager*

Esta clase es la fachada de la funcionalidad de la librería JasperReports en el proceso de llenado del reporte. Esta expone una variedad de métodos que reciben un reporte compilado en forma de objeto, archivo o flujo de entrada y produce un documento también en diversas formas de salida: objeto, archivo o flujo de salida.

Pero junto con el diseño de reporte, el motor de llenado de reportes tiene que recibir también el origen de datos para recuperar los datos y los valores de los parámetros del informe, con el fin de generar los documentos. Los valores de los parámetros son suministrados siempre en un objeto *java.util.Map* en el cual los valores importantes son los nombres de los parámetros del informe. El origen de los datos puede suministrarse de dos formas diferentes, dependiendo de la situación:

Normalmente, este puede ser suministrado como un objeto *dori.jasper.engine.JRDataSource*, como ya explicamos anteriormente. Pero en la mayoría de los diseños, la información necesaria para el llenado de los reportes es obtenida de bases de datos relacionales, JasperReports incorpora un comportamiento predeterminado de acceso a las bases de datos que permite a los usuarios especificar una consulta SQL en el diseño. Esta consulta SQL se ejecuta con el fin de recuperar los datos a utilizarse para generar el reporte en tiempo de ejecución.

En estos casos, lo único que necesita JasperReports es un objeto *java.sqlConnection*, en lugar del objeto de origen de datos habitual. Es necesario este objeto de conexión para conectar con el sistema de administración de bases de datos relacionales a través de un JDBC y ejecutar la consulta del informe.

Automáticamente, se crea una instancia de la clase *dori.jasper.engine.JRResultSetDataSource* para envolver el objeto *java.sql.ResultSet* retornado después de la ejecución de la consulta SQL y la pasa al proceso de llenado del reporte.

4.2.10. Clase *dori.jasper.engine.JRAbstractScriptlet*

Scriptlets es una característica muy potente de la librería JasperReports. Permiten a los usuarios escribir código personalizado que será ejecutado por el motor de reportes durante el proceso de llenado del reporte. Este código de usuario puede tratar con la manipulación de datos del reporte y se ejecuta en momentos bien definidos, tales como los saltos de página, columna o grupo; abriendo un nuevo abanico de posibilidades para personalizar el contenido de los documentos generados.

4.2.11. Clase *dori.jasper.engine.JRDefaultScriptlet*

Esta es una subclase de la clase *dori.jasper.engine.JRAbstractScriptlet*. La mayoría del tiempo los usuarios optaron por esta subclase, cuando trabajan con scriptlets, para que no tengan que implementar todos los métodos abstractos declarados en la clase abstracta.

4.2.12. Clase *dori.jasper.engine.JasperPrintManager*

Hemos dicho que JasperReports es una herramienta para crear reportes en Java y que los documentos que la herramienta genera se pueden imprimir. Después de llenar el informe, tenemos la opción de visualizarlo o de exportarlo a un formato diferente y por último, y no menos importante, la impresión del mismo.

En JasperReports, podemos imprimir reportes usando esta clase particular, que es una fachada para la funcionalidad de impresión expuesta por la librería.

Podemos encontrar aquí varios métodos, que envían a la impresora de documentos enteros o sólo partes de ellos, ya sea mediante la visualización del diálogo de impresión o no.

El contenido de una página de un documento JasperReports se puede visualizar mediante la instanciación de un objeto *java.awt.Image* para que use esta clase gestora.

4.2.13. Clase *dori.jasper.engine.JasperExportManager*

Como ya se mencionó, JasperReports permite la transformación de los documentos generados a partir de su formato original hacia formatos de documentos más populares, tales como: PDF, HTML o XML. Con el tiempo, esta parte de la funcionalidad de JasperReports se ha extendido para soportar otros formatos como: CSV, XLS y otros.

Esta clase posee varios métodos para procesar los datos que provienen de las diferentes fuentes y enviarlos a diferentes destinos, como: archivos, flujos de entrada y salida, etc.

4.2.14. Clase *dori.jasper.engine.JasperRunManager*

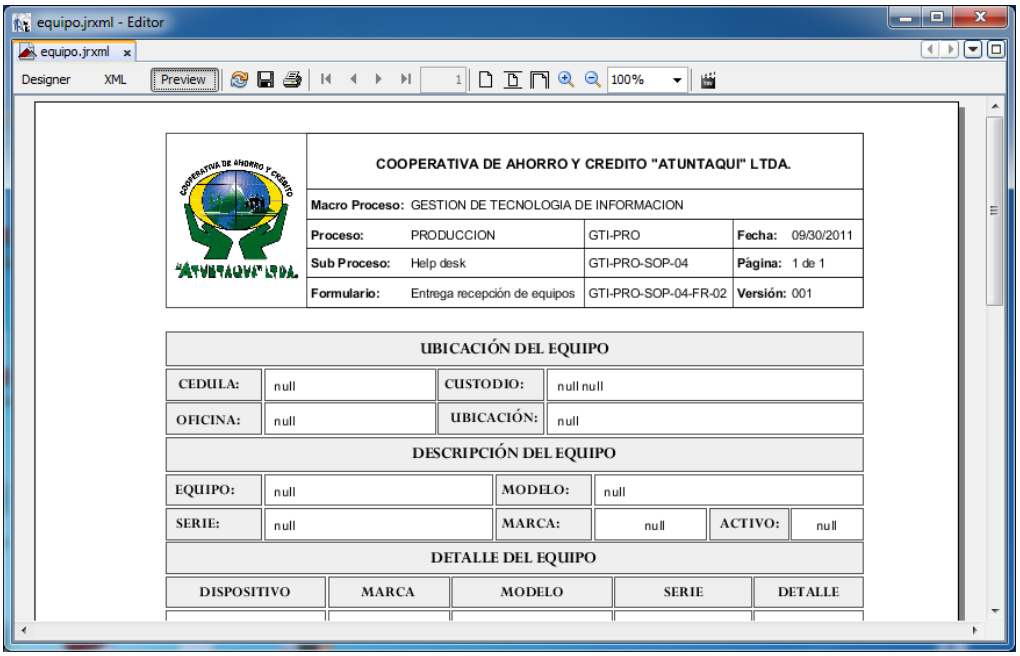
A veces es útil para producir documentos solo en los formatos populares, tales como: PDF o HTML, sin tener que almacenarlo en el disco serializado, intermediando un objeto de la clase *dori.jasper.egine.JasperPrint*, producido por el proceso de llenado del informe.

Esto se puede lograr con la gestión de esta clase, la cual inmediatamente exporta los documentos producidos por el proceso de llenado del informe en el formato de salida deseado.

4.2.15. Clase *dori.jasper.view.JRViewer*

Esta clase es diferente del resto de las clases mencionadas anteriormente, en la sentido de que es más como un componente visual que una clase de utilidad.

Se puede utilizar en aplicaciones basadas en Swing para visualizar los reportes generados por la librería JasperReports.



Fuente: [Propia]

Figura 4.2. Vista previa de un diseño con el visor de JasperReports

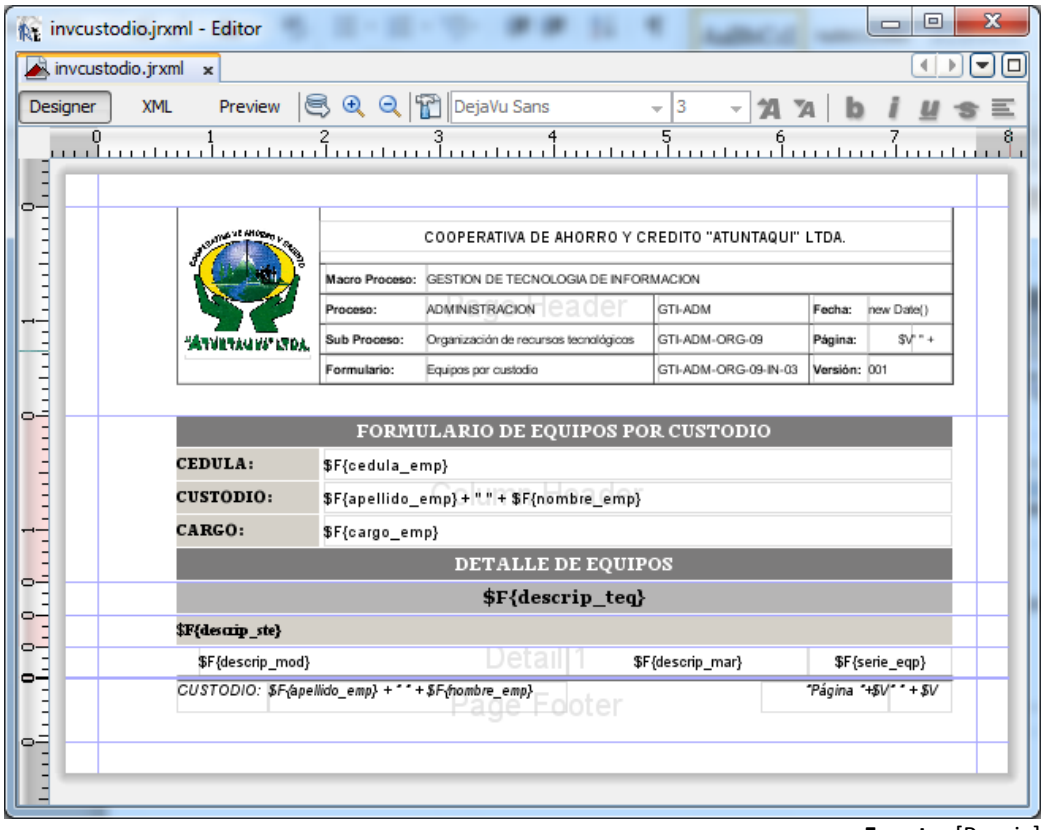
La intención de este componente visual no es la de satisfacer a todos. Se incluyó en la librería principal, más como un componente de demostración, para demostrar como la funcionalidad de impresión puede utilizarse, para visualizar los reportes en aplicaciones basadas en Swing, mediante la generación de objetos *java.awt.Image* de las páginas del documento, utilizando la clase *dori.jasper.engine.JasperPrintManager*.

4.2.16. Clase *dori.jasper.view.JasperViewer*

Esta también, es más como una clase de finalidad didáctica que utiliza el componente *dori.jasper.view.JRViewer* para mostrar los reportes. Representa una sencilla aplicación Java Swing que puede cargar y visualizar los reportes. Se utiliza en casi todas las muestras suministradas para mostrar los documentos generados.

4.2.17. Clase *dori.jasper.view.JasperDesignViewer*

Por lo general, una aplicación que utiliza la librería JasperReports con fines de información, no se pondrá a trabajar con esta clase. Esta clase se puede utilizar en tiempo de diseño para previsualizar las plantillas de reportes antes de publicarlos en producción y ayudar con el trabajo de diseño. Se incluyó en la librería principal, como una herramienta de desarrollo, con el fin de compensar la falta de diseño visual.



Fuente: [Propia]

Figura 4.3. Vista diseño de un reporte en JasperReports.

Esta también se utiliza en todas las muestras para ver los diseños del reporte, ya sea en formato XML o de manera compilada.

4.2.18. Clase *dori.jasper.engine.util.JRLoader*

Todos los procesos principales de JasperReports, como la compilación de reportes, el llenado y exportación de reportes, generalmente trabajan con objetos serializados. A veces, es muy útil cargar manualmente los objetos serializados antes de someterlos al proceso de JasperReports deseado.

Es por eso que contamos con la clase *dori.jasper.engine.util.JRLoader*, que es una clase de utilidad que nos ayuda a cargar los objetos serializados que se encuentran en varios lugares, tales como: archivos, direcciones URL o flujos de entrada.

El método más interesante expuesto por esta clase es el método *loadObjectFromLocation (String location)*. Cuando se llama a este método para cargar un objeto desde la ubicación suministrada, el programa primero tratará de interpretar la ubicación como una URL válida. Si esto falla, entonces el programa considera que la ubicación suministrada es el nombre de un archivo en disco y tratará de leerlo. Si no se encuentra el archivo en dicha ubicación, tratará de encontrar un recurso a través del *classpath* que corresponda a la ubicación. Sólo después de este tercer intento fallido, la excepción será lanzada.

4.3. PRINCIPALES TAREAS Y PROCESOS

En esta sección se describe lo que hay que saber para poder analizar los diseños en XML; compilarlos, llenarlos, visualizarlos, imprimirlos o exportarlos a otros formatos.

4.3.1. Análisis XML

JasperReports utiliza la API SAX[1] 2.0 para analizar los archivos XML. Sin embargo, no está ligado a una particular implementación de SAX[1] 2.0, como Xerces[2] por ejemplo, sino que es capaz de decidir en tiempo de ejecución que analizador XML utilizará.

Para instanciar una clase de análisis, JasperReports utiliza el método *createXMLReader()* de la clase *org.xml.sax.helpers.XMLReaderFactory*. En este caso, será necesario establecer en tiempo de ejecución la propiedad del sistema Java *org.xml.sax.driver*, el nombre completo de la clase del controlador SAX[1], como se especifica en la documentación SAX[1] 2.0. Se puede lograr esto de dos maneras, explicaremos ambas usando el analizador XML Xerces[2].

La primera forma de poder establecer una propiedad en el sistema es mediante el uso de la opción '-D' en la línea de comandos al iniciar la máquina virtual de Java:

```
java -Dorg.xml.sax.driver=org.apache.xerces.parsers.SAXParser
MySAXAppSample.xml
```

En todos los ejemplos previstos usamos la herramienta de construcción ANT para ejecutar las diferentes tareas. Suministramos esta propiedad del sistema a la JVM utilizando el elemento *<sysproperty>* de *<java>* integrando la tarea:

[1] Simple API for XML, API creado para el lenguaje Java, que después se convirtió en la API de facto para usar XML en JAVA.
[2] Xerces es un parser XML derivado del que anteriormente era de IBM.

```
<sysproperty key="org.xml.sax.driver"
value="org.apache.xerces.parsers.SAXParser"/>
```

La segunda manera de establecer una propiedad del sistema es mediante el uso del método *java.lang.System.setProperty (clave String, el valor String)*, de la siguiente manera:

```
System.setProperty("org.xml.sax.driver",
"org.apache.xerces.parsers.SAXParser");
```

4.3.2. Compilación del diseño

Con el fin de generar un reporte, uno tiene que crear el diseño en primer lugar, ya sea mediante la edición de un archivo XML o mediante programación la construcción de un objeto *ori.jasper.engine.design.JasperDesign*. Esta investigación, especialmente se relacionará con el enfoque XML, ya que es la mejor forma de utilizar la librería JasperReports y de esta manera entenderemos de mejor manera su comportamiento.

Es muy probable que las herramientas GUI[3] existentes para el desarrollo de aplicaciones, ayuden y simplifiquen el trabajo de diseño al utilizar directamente la API de JasperReports para crear los objetos del diseño, sin necesidad de pasar a través del formato XML. Pero, nuestro objetivo está orientado a explicar el contenido y la sintaxis de los diseños XML.

Como ya se mencionó, los diseños XML son la materia prima que la librería utiliza para generar reportes. Esto es porque este contenido XML tiene que ser analizado y cargado en un objeto *dori.jasper.engine.design.JasperDesign*, que tiene que pasar el proceso de compilación de reportes antes de estar listo para ser presentado con los datos por el motor de reportes.

Se debe tomar en cuenta que la mayoría de las veces, la compilación de reportes debe ser considerada como una tarea de la etapa de desarrollo. Pues se deben compilar los diseños de la aplicación, de la misma manera como se compilan los archivos fuente de Java. Esto debido a que en la mayoría de casos, los diseños son aplicaciones estáticas y algunas deben ofrecer a sus usuarios la posibilidad de generar de forma dinámica los diseños, los mismos que necesitan ser compilados en tiempo de ejecución.

El objetivo principal del proceso de compilación de reportes es producir y carga los *bytecode* de la clase que contiene todas las expresiones del reporte. Esta clase creada dinámicamente se utilizará en el llenado del reporte para evaluar todas las expresiones. Pero, antes de continuar con la generación de la clase, el motor verifica la consistencia del diseño y no continuará si al menos una comprobación de validación falla.

Hay por lo menos tres aspectos importantes relativos a la forma en que, el *bytecode* de la clase que contiene todas las expresiones del informe se obtienen:

- El directorio de trabajo temporal
- El compilador Java utilizado.

[3] Graphical User Interface, es un programa que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

- El classpath.

Con el fin de poder compilar un archivo fuente de Java, este archivo debe ser creado y guardado en el disco. La salida del proceso de compilación de Java es también un archivo con la extensión ".*class*". Esto debido a que JasperReports necesita tener acceso a un directorio temporal de trabajo, para crear la clase que contiene las expresiones del informe y para su compilación. Después que la tarea de compilación del reporte esté terminada, los archivos temporales de la clase se eliminarán automáticamente y los *bytecode* resultantes se almacenan en el *dori.jasper.engine.JasperReport*, que puede ser serializado y almacenado en el disco, si se lo desea.

De manera predeterminada, el directorio temporal de trabajo es el directorio actual cuando se inicia la JVM, y se obtiene interrogando a la propiedad del sistema *user.dir*. Este se puede cambiar fácilmente, proporcionando un valor a una propiedad del sistema llamada *jasper.reports.compile.temp*. Esta opción es muy útil, especialmente en el entorno web, cuando no se quiere terminar usando el mismo directorio que contiene los archivos batch que inician el servidor web, como un directorio temporal de trabajo para el proceso de compilación de reportes.

El segundo aspecto mencionado consiste en el compilador Java utilizado para compilar las clases de las expresiones del reporte. Primero, el motor generador de reportes trata de compilar el archivo fuente de Java utilizando la clase *sun.tools.javac.Main*; este método puede tener éxito sólo si el archivo "tools.jar" que contiene esta clase, normalmente ubicado en la carpeta "/lib" del directorio de instalación del JDK, está disponible a través del *classpath*. Si la carga de la clase *sun.tools.javac.Main* falla, el programa trata de iniciar de manera dinámica el proceso de compilación de Java, como se lo hace normalmente desde la línea de comandos, utilizando el programa javac.exe ubicado en la carpeta "/bin" del directorio de instalación del JDK. Por eso, en el árbol del proyecto disponible para descargar, copiar el archivo tools.jar desde el directorio del JDK a la carpeta "/lib" del proyecto JasperReports, es una operación opcional. Si el archivo tools.jar no se encuentra en el *classpath*, JasperReports muestra una advertencia y continúa como se ha mencionado.

Cuando se compila los archivos fuente Java, lo más importante parece ser el *classpath*. Si el compilador Java no encuentra en el *classpath* suministrado todas las clases que son referenciadas desde los archivos fuente; todo el proceso falla y se detiene, los errores generados son mostrados en la consola.

Lo mismo ocurre cuando JasperReports compila las clases de las expresiones del reporte. Por eso es importante asegurarse que suministramos al compilador Java el *classpath* correcto, para que el proceso de compilación tenga éxito. Si no hay *classpath* especial para la compilación de las clases suministradas, el motor utilizará el *classpath* actual, devuelto por la propiedad del sistema *java.class.path*. Este comportamiento por defecto puede cambiarse

poniendo el *classpath* deseado en la propiedad del sistema llamada *jasper.reports.compile. class.path*.

Se pueden consultar los archivos *jsp/compile.jsp* y *WEB-INF/classes/servlets/CompileServlet. java* en el *webapp* proporcionado, para el fragmento de código que utiliza estas propiedades del sistema Java para modificar el comportamiento predeterminado del proceso de compilación de reportes de JasperReports. La mayoría de las veces, la compilación de un informe sólo requiere una simple llamada a la librería JasperReports, como se indica en la siguiente línea de código:

```
dori.jasper.engine.JasperCompileManager.compileReport(myXmlFileName);
```

Esta llamada la realizada la compilación del diseño y guarda un archivo con la extensión "*.jasper*" en el mismo directorio donde se encuentra el archivo XML del diseño suministrado.

4.3.3. Vista previa del diseño

La librería JasperReports no se suministra con una avanzada herramienta GUI para ayudar en la tarea de diseño de reportes. Sin embargo, en la actualidad existen algunas herramientas de desarrollo que conjuntamente con esta librería proporcionan una interfaz simple y amigable para el diseño de reportes. La biblioteca contiene un componente visual muy útil que permite a los desarrolladores realizar una vista previa de los diseños de reportes y ver cómo van quedando los reportes.

La clase *dori.jasper.view.JasperDesigner* es una aplicación swing sencilla basada en Java que puede cargar y mostrar un diseño de informe, ya sea en su forma XML o en forma compilada. Todos los ejemplos suministrados tienen preparado las tareas ANT[4] en sus archivos *build.xml* que pondrá en marcha este visor de diseño para que se pueda visualizar los diseños de informe. De hecho, hay dos tareas ANT[4] por cada ejemplo de informe: *viewDesign* y *viewDesignXML*. La primera carga el diseño de informe compilado que normalmente se encuentra en el archivo ".jasper". La segunda carga el archivo XML del diseño, siendo el más útil ya que se puede editar el archivo XML y pulsar el botón actualizar para ver inmediatamente la modificación en la pantalla.

Si usted tiene la herramienta ANT[4] de construcción instalada en el sistema, con el fin de obtener una vista previa del diseño, basta con ir al directorio deseado de la muestra y el ejecutar en la línea de comandos algo como esto:

```
>ant viewDesignXML
```

O

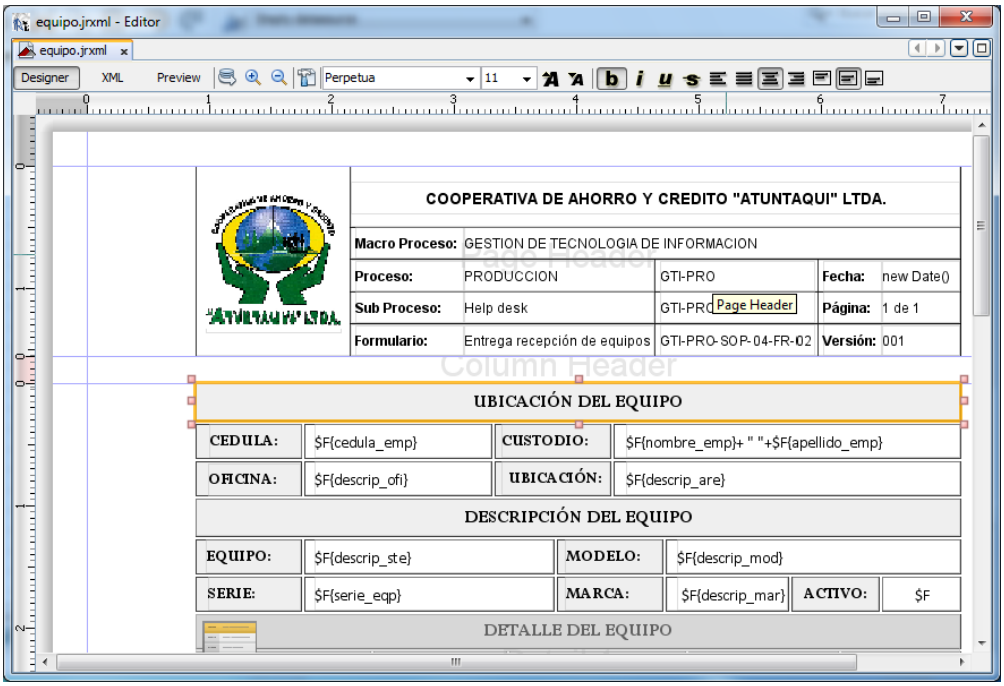
```
>ant viewDesign
```

[4] ANT, herramienta usada durante la fase de compilación y construcción para la realización de tareas mecánicas.

En caso de no tener instalada la herramienta ANT[4], aquí se presenta la línea de comandos completa que ejecutará el visualizador para obtener una vista previa del diseño.

```
>java -classpath ./;../../../lib/commons-digester.jar;
../../../lib/commons-beanutils.jar;../../../lib/commons-collections.jar;
../../../lib/xerces.jar;../../../lib/jasperreports.jar
-Dorg.xml.sax.driver=org.apache.xerces.parsers.SAXParser
dori.jasper.view.JasperDesignViewer -XML -FFirstJasper.xml
```

Con el lanzamiento de esta línea de comandos, debe ser capaz de ver la siguiente ventana:



Fuente: [Propia]

Figura 4.4. Vista previa de un diseño de un reporte.

4.3.4. Llenado del Reporte

El proceso de llenado del reporte es el más importante de toda la funcionalidad de la librería JasperReports. Este representa el objetivo principal de este componente de software, ya que es el proceso que manipula los conjuntos de datos con el fin de producir documentos de alta calidad, al igual que cualquier herramienta de reportes.

Hay tres cosas que deben ser proporcionadas al proceso de llenado del informe:

- El diseño
- Los parámetros
- La fuente de datos

El resultado es siempre un documento listo para visualizar, imprimir o exportar a otros formatos.

[4] ANT, herramienta usada durante la fase de compilación y construcción para la realización de tareas mecánicas.

Nosotros hemos visto que para llenar un informe, tenemos que utilizar la clase *dori.jasper.engine.JasperFillManager*. Esta clase tiene varios métodos que nos permiten llenar los diseños que se encuentran en disco, vienen desde los flujos de entrada o son directamente suministrados como objetos *dori.jasper.engine.JasperReport* en memoria. El resultado generado siempre corresponde al tipo de entrada recibido; es decir, al recibir un nombre de archivo para el diseño, el reporte generado también se pondrá en un archivo en disco. Cuando el diseño se lee de un flujo de entrada, el reporte generado se escribirá a un flujo de salida.

Podría ser que los varios métodos de utilidad para llenar los reportes no sean suficientes para una aplicación en particular, que por ejemplo quisiera cargar los diseños de reportes como recursos del *classpath* y los documentos generados a archivos en disco, en cierta ubicación.

En algunos casos, los diseñadores deben considerar la carga los objetos de los diseños antes de pasarlos a las rutinas de llenado del reporte, usando la clase de utilidad *dori.jasper.engine.util. JRLoader*. De esta manera, ellos podrían recuperar algunas propiedades del diseño como el nombre del reporte, para que ellos puedan construir el nombre del documento resultante y situarlo en la ubicación del disco deseada.

Hay muchos escenarios posibles en el llenado de reportes que podría imaginarse en una aplicación del mundo real, y la clase que administra el llenado del reporte sólo intenta cubrir una parte de este. Pero esto no debe ser un problema para alguien que quiera personalizar el proceso de llenado del reporte que utiliza la funcionalidad básica de la librería.

Los valores de los parámetros del reporte siempre se proporcionan por un objeto de la clase *java.util.Map* que tiene los nombres de los parámetros, así como sus valores de entrada.

En cuanto al tercer elemento que el proceso de llenado de reporte espera recibir, la fuente de los datos, existen dos escenarios posibles.

Normalmente, el motor generador de reportes trabaja con una instancia de la interfaz *dori.jasper.engine.JRDataSource*, desde donde se extrae los datos para llenar el reporte. Y la fachada de la clase *dori.jasper.engine.JasperFillManager* tiene una conjunto completo de métodos que reciben un objeto *dori.jasper.engine.JRDataSource*, como la fuente de datos del reporte que va a ser llenado.

Pero hay otro conjunto de métodos para el llenado del reporte, en este la clase administradora recibe un objeto *java.sql.Connection* como un parámetro, en lugar del objeto de fuente de datos esperado. Esto es porque la mayoría de los reportes son generados usando datos que provienen de tablas de bases de datos relacionales.

Los usuarios tienen la posibilidad de especificar la consulta SQL necesaria para recuperar los datos del informe desde la base de datos, en el propio diseño. El runtime, la única cosa que necesita será un objeto de conexión JDBC, usado para conectarse a la base de datos relacional deseada, ejecutar la consulta SQL y recuperar los datos del informe. Por debajo, el motor

generador de reportes utilizará un objeto especial, pero esto es transparentemente para el programa que lo invoca.

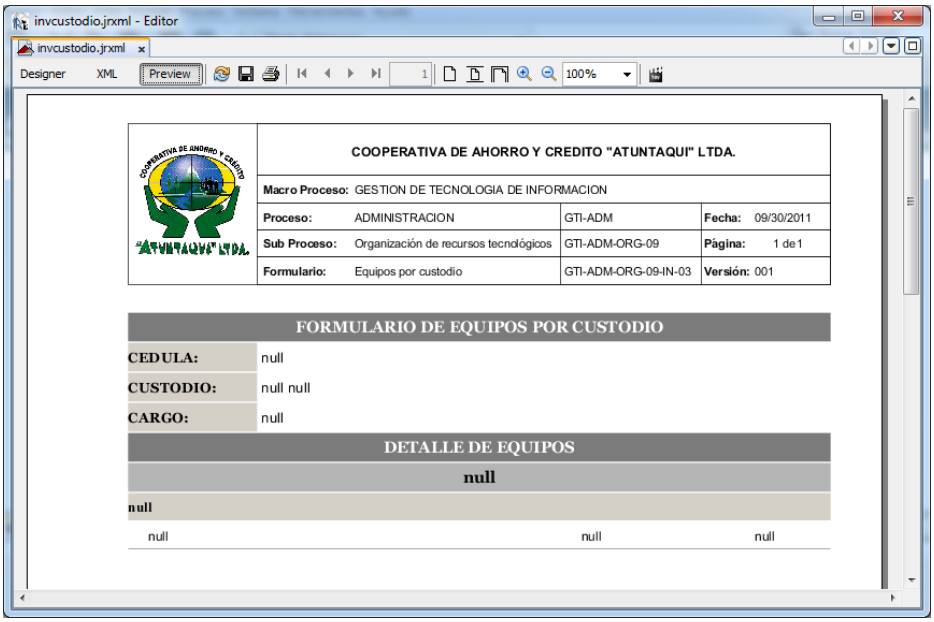
4.3.5. Visualización de reportes

El resultado del proceso de llenado del informe siempre es un objeto *dori.jasper.engine.JasperPrint*. Si nosotros serializamos este objeto y lo guardamos en disco, normalmente en un archivo ".jrprint", nosotros podríamos decir que éste es el formato propietario en que JasperReports guarda sus documentos generados.

Para ver los reportes generados en este formato propietario o en el formato XML propietario producido por el exportador XML interno, JasperReports tiene un visualizador incorporado. Este es un componente basado en swing que puede integrarse fácilmente con otras aplicaciones Java que deseen incorporar esta funcionalidad, sin la utilidad de exportar los documentos en los formatos más populares, para que ellos puedan visualizarse. La clase de *dori.jasper.view.JRViewer* representa este componente visual. Esta puede personalizarse para responder a las necesidades de aplicaciones en particular, de esta manera podríamos agregar o quitar los botones de la barra de herramientas existente que se despliega, o realizar otras modificaciones.

JasperReports también viene con una aplicación swing simple que usa el componente visual para visualizar los reportes. Este nos permite visualizar los reportes guardados en el disco, en el formato propietario de JasperReports "jrprint" cuando lo invocamos o en el formato XML producido por el exportador de XML embebido.

Esta aplicación Java swing esta implementada en la clase *dori.jasper.view.JasperViewer* y se utiliza para mostrar los reportes generados.



Fuente: [Propia]

Figura 4.5. Visualización de un reporte en el visor de JasperReports.

El visualizador incorporado en la clase *dori.jasper.view.JasperViewer* debe ser considerado más como una herramienta de prueba que muestra cómo el componente *dori.jasper.view.JRViewer* puede usarse en aplicaciones swing para visualizar los reportes.

Si se invoca directamente desde las aplicaciones los métodos públicos y estáticos que expone *viewReport()*, se terminará notando que cuando se cierra el marco de visualización del reporte, la aplicación terminará inesperadamente. Esto es porque la clase *JasperViewer* hace una llamada al *System.exit(0)* y este eliminará el *java.awt.event.WindowListener* que se ha registrado.

4.3.6. Impresión de reportes

El objetivo principal de la librería JasperReports y el de todas las herramientas existentes para la generación de reportes, es crear documentos listos para imprimir. La mayoría de reportes que se generan por las aplicaciones terminan o se supone que terminaran en el papel.

Nosotros podemos imprimir los documentos generados por la librería JasperReports utilizando la clase *dori.jasper.engine.JasperPrintManager*. También pueden imprimirse los documentos después de exportarse a otros formatos como HTML o PDF. Pero nosotros vamos a explicar cómo usar la clase administradora especializada mencionada, para imprimir documentos fueron guardados o transferidos en el formato propietario de JasperReports.

Entre los varios métodos que la clase *dori.jasper.engine.JasperPrintManager* expone, nosotros podemos encontrar algunos que permiten imprimir un documento entero, una sola página o un rango de páginas, con y sin desplegar el diálogo de la impresión.

Aquí se muestra cómo imprimir un documento entero sin desplegar el diálogo de la impresión normal:

```
dori.jasper.engine.JasperPrintManager.printReport(myReport, false);
```

Y ahora, mostramos el código para imprimir el rango de páginas de 5 a 11 del documento, después de haber desplegado el diálogo de la impresión normal:

```
dori.jasper.engine.JasperPrintManager.printPages(myReport, 4, 10, true);
```

4.3.7. Exportación de reportes

En algunos ambientes de aplicación, es conveniente transformar el documento JasperReports generado, del formato propietario a otros formatos más populares como PDF o HTML. Haciendo esto, usted puede hacer que otras personas puedan ver estos reportes sin necesidad de instalar visualizadores especiales en sus sistemas, especialmente cuando enviamos los documentos a través de la red.

Existe también una clase fachada en JasperReports para este tipo de funcionalidad. Su nombre es *dori.jasper.engine.JasperExportmanager* y puede usarse para obtener contenido PDF, HTML o XML para los documentos producidos por el proceso de llenado de reportes.

Exporta recursos utilizando un objeto *dori.jasper.engine.JasperPrint* que representa un documento *JasperReport* y lo transforma en un formato diferente. La razón principal para exportar los reportes en otros formatos es permitir a más personas ver estos reportes. El formato HTML puede verse por alguien en estos días, desde un navegador web que esté disponible en cualquier sistema. Para ver los documentos JasperReports en su formato original se requiere la instalación de un software especial dependiendo de la plataforma, por lo menos en la forma de un applet de Java si no más.

En conclusión, la habilidad de exportar los reportes a otros formatos es una característica muy útil y con el tiempo, más y más formatos de salida son soportados.

Para exportar los reportes a otros formatos nuevos, se tiene que implementar una interfaz especial llamada *dori.jasper.engine.JRExporter* o extender la clase *dori.jasper.engine. JRAbstractExporter* correspondiente.

Por el momento, la librería se proporciona con tres clases especiales de exportación que producen contenido PDF, HTML y XML. Estos se encuentran en el paquete *dori.jasper.engine.export*, pero puede usarse llamando los métodos apropiados en la clase fachada mencionada anteriormente.

A continuación mostramos un ejemplo para exportar un informe al formato de HTML.

```
dori.jasper.engine.JasperExportManager.exportReportToHtmlFile(myReport);
```

4.4. DISEÑO DEL REPORTE

El diseño del reporte representa una plantilla que será usada por el motor generador de reportes de JasperReports para entregar un documento listo a la impresora, pantalla o web. La información almacenada en la base de datos es organizada durante el proceso de llenado del reporte de acuerdo al diseño para obtener documentos listos.

Generalmente, un diseño contiene toda la información concerniente a la estructura y el aspecto de los documentos que se generarán cuando los datos se proporcionen. Esta información involucra la posición y el contenido de varios elementos de texto o de gráficos que se mostrarán en el documento, su apariencia, los cálculos personalizados, datos que se agrupan y la manipulación de los datos que debe realizarse al generar los documentos, etc.

Normalmente, los diseños se definen en archivos XML con una estructura especial que veremos en detalle más adelante y son sujetos al proceso de compilación de JasperReports antes de ser llenados con los datos. Pero ellos también pueden construirse en memoria, usando el API de JasperReports. Existe un ejemplo llamado noxmldesign enviado en el código fuente del proyecto JasperReports que muestra cómo crear diseños directamente en memoria, sin editar ningún archivo XML en absoluto.

4.4.1. Referencia de DTD

Cuando se trabaja con diseños XML, JasperReports usa su propio archivo DTD para validar el contenido XML recibido para procesar. Si la validación del XML es correcta, significa que el diseño proporcionado corresponde a la sintaxis y estructura XML requerida por JasperReports y el motor generador de reportes puede generar la versión compilada del diseño.

Los diseños XML válidos siempre señalan a la DTD interna de JasperReports para la validación. Sin la referencia específica de la DTD, el proceso de compilación de reporte falla abruptamente. Esto no debe ser considerado como un grave problema ya que la referencia al DTD siempre es la misma y simplemente puede copiarse de los diseños anteriores, en un inicio se lo puede copiar de los ejemplos proporcionados.

Como se mencionó anteriormente, el motor generador de reportes reconoce sólo las referencias de DTD que apuntan a sus archivos de DTD internas. No se puede hacer una copia de los archivos DTD encontrados en los archivos fuente de la librería y re direccionar a esta copia en los diseños XML. Si se quiere hacer esto, se tendrá que modificar el código de algunas clases de la librería incluida la clase *dori.jasper.engine.xml.JRXmlDigester*. Si alguna vez se encuentra con problemas como que el motor generador de reportes no encuentra su propio archivo DTD interior debido a algún problema de carga, debemos asegurarnos de haber eliminado cada causa posible antes de decidir usar los archivos DTD externos. Este problema es muy improbable ya que el mecanismo de carga de la librería se ha ido mejorando con el tiempo.

Existen dos referencias DTD válidas para el diseño de reportes XML y estas son las siguientes:

```
<!DOCTYPE jasperReport PUBLIC "-//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
```

o

```
<!DOCTYPE jasperReport PUBLIC "-//JasperReports//DTD Report Design//EN"
"http://www.jasperreports.com/dtds/jasperreport.dtd">
```

El elemento raíz de un diseño de informe XML es <jasperReport> y así es cómo un archivo XML de diseño de informe en JasperReports usualmente se presenta:

```
<?xml version="1.0"?>
<!DOCTYPE jasperReport PUBLIC "-//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
<jasperReport name="name_of_the_report" ... >
...
</jasperReports>
```

4.4.2. Codificación XML

Cuando creamos diseños XML en diferentes idiomas, una especial atención se debe otorgar al atributo de la codificación que puede usarse en la cabecera del archivo XML. Por defecto, si ningún valor se especifica para este atributo, el parser XML usa "UTF-8" como la codificación para el contenido del archivo XML.

Esto es muy importante porque el diseño de informe a menudo contiene textos estáticos que son introducidos cuando editamos manualmente el archivo de XML.

Para la mayoría de los idiomas europeos orientales, la codificación "ISO-8859-1", también conocida como LATIN1, debe ser suficiente para tratar con caracteres especiales como é, â, è, ç que posee el idioma francés.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE jasperReport PUBLIC "-//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
<jasperReport name="name_of_the_report" ... >
...
</jasperReports>
```

Para averiguar, la codificación exacta de un idioma en particular cuándo editamos archivos XML, se debe consultar la documentación referente a XML.

4.4.3. Propiedades del reporte

Nosotros ya hemos visto que *<jasperReport>* es el elemento raíz de un diseño XML. En este apartado se da a conocer en detalle las propiedades de un objeto del diseño y cuáles son sus atributos XML.

```
<!ELEMENT jasperReport (reportFont*, parameter*, queryString?, field*,
variable*, group*, title?, pageHeader?, columnHeader?, detail?,
columnFooter?, pageFooter?, summary?)>
<!ATTLIST jasperReport
        name NMTOKEN #REQUIRED
        columnCount NMTOKEN "1"
        printOrder (Vertical | Horizontal) "Vertical"
        pageWidth NMTOKEN "595"
        pageHeight NMTOKEN "842"
        orientation (Portrait | Landscape) "Portrait"
        whenNoDataType (NoPages | BlankPage | AllSectionsNoDetail) "NoPages"
        columnWidth NMTOKEN "555"
        columnSpacing NMTOKEN "0"
        leftMargin NMTOKEN "20"
        rightMargin NMTOKEN "20"
        topMargin NMTOKEN "30"
        bottomMargin NMTOKEN "30"
        isTitleNewPage (true | false) "false"
        isSummaryNewPage (true | false) "false"
        scriptletClass NMTOKEN #IMPLIED
>
```

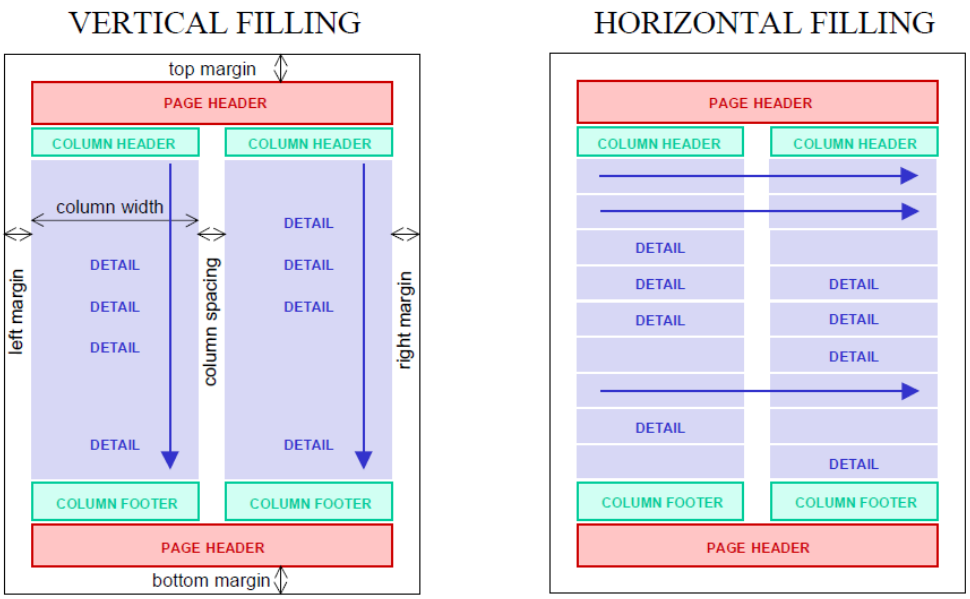
4.4.3.1. Nombre del reporte

Cada diseño tiene que tener un nombre. Este nombre es importante porque la librería lo usa cuando genera archivos, especialmente cuando el procedimiento predefinido es para la compilación, llenado o exportación del informe.

El nombre del informe es especificado usando el atributo *name* del elemento *<jasperReport>* y es obligatorio. Los espacios no se permiten en el nombre del informe pues tiene que ser una palabra.

4.4.3.2. Número de Columnas

JasperReports permite crear reportes con más de una columna en cada página, como en la siguiente figura, dónde nosotros podemos ver el diseño de un reporte con dos columnas:



Fuentes: [L01]

Figura 4.6. Estructura de un diseño de reporte con varias columnas.

Por defecto, el motor generador de reportes crea el reporte con una columna por cada página.

4.4.3.3. Orden de impresión

Para los reportes que tienen más de una columna, es importante especificar el orden en que las columnas se llenarán y esto se puede hacer usando el atributo *printOrder* del elemento *<jasperReport>*.

Existen dos posibles situaciones:

- **Llenado vertical:** seleccionando esta opción se garantiza que las columnas serán llenadas de arriba hacia abajo y de izquierda a derecha (*printOrder="Vertical"*).

[L01] The JasperReports Ultimate Guide, Teodor Danciu.

- **Llenado horizontal:** seleccionando esta opción las columnas serán llenadas de izquierda a derecha y de arriba hacia abajo (*printOrder="Horizontal"*).

El orden de impresión por defecto es:

```
printOrder="Vertical"
```

4.4.3.4. Tamaño de la página

Hay dos atributos a este nivel para especificar el tamaño de la página del documento que va a ser generado: el *pageWidth* y el *pageHeight*.

Como todos los otros atributos de JasperReports que representan las dimensiones y la posición del elemento, deben ser especificados en pixeles. JasperReports usa la resolución de Java por defecto de 72 puntos por pulgada. Esto significa que un *pageWidth = "595"* hacen las 8.26 pulgadas aproximadas correspondientes al ancho de una hoja tamaño A4.

El tamaño de página predefinido corresponde a una hoja de tamaño A4:

```
pageWith="595" pageHeight="842"
```

4.4.3.5. Orientación de la página

El atributo de orientación se usa para especificar si estamos creando documentos usando la orientación "*Portrait*" o "*Landscape*". JasperReports requiere reajustar el ancho y alto de la página cuando cambiamos la orientación del documento de "*Portrait*" a "*Landscape*" o viceversa.

Por ejemplo, si queremos crear un reporte con un tamaño de hoja A4 usando la orientación de página "*Portrait*", entonces el valor de los atributos serán:

```
pageWidth="595" pageHeight="842" orientation="Portrait"
```

Si por el contrario, decidimos usar la orientación "*Landscape*" para nuestro documento en tamaño A4, debemos asegurarnos que las dimensiones del tamaño de página se modificaron de acuerdo la siguiente línea de código:

```
pageWidth="842" pageHeight="595" orientation="Landscape"
```

Esto es porque JasperReports tiene que saber exactamente el ancho y alto de las páginas que utilizará, y no necesariamente considera el valor que proporcionamos en el atributo de orientación, por lo menos no en el momento de llenado del reporte.

Este atributo de orientación se utiliza sólo el momento de imprimir el reporte, para informar a la impresora o a algunos exportadores especiales sobre la orientación de la página.

La orientación de página por defecto es "Portrait".

4.4.3.6. Márgenes de la página

Una vez definido el tamaño de la página, se puede especificar que márgenes deberá preservar el motor generador de reportes cuando genere los reportes. Hay cuatro atributos para esta tarea: *topMargin*, *leftMargin*, *bottomMargin* y *rightMargin*.

El tamaño de margen predefinido para arriba y debajo de la página es de 20 pixeles y el margen predefinido para los márgenes de la izquierda y la derecha es de 30 pixeles.

4.4.3.7. Tamaño y espaciado de columna

Los reportes pueden tener más de una columna, como hemos visto en la sección que trata sobre el atributo *columnCount*. Pero el motor generador de reportes tiene que saber cuán grande puede ser una columna y que espaciado debe permitir entre columnas. Hay dos atributos para esta tarea: *columnWidth* y *columnSpacing*.

Hay también un chequeo de validación realizado cuando compilamos el diseño, que no nos permiten crear los reportes si el valor del ancho de columna y el espaciado entre columnas no encajan con el ancho y los márgenes de la página especificados.

Como por defecto existe una sola columna, el espaciado predeterminado de la columna es de 0 pixeles y el ancho predefinido de la columna es igual al ancho predefinido de la página, menos los valores de los márgenes predefinidos de la izquierda y derecha, que llegan a ser 555 pixeles.

4.4.3.8. Fuentes de datos vacías

A veces la fuente de los datos que proporcionamos a los reportes no tiene ningún registro. En este caso, no está claro lo que se debe hacer. En esta situación, algunos pueden esperar para visualizar un documento en blanco y otros podrían querer tener algunas secciones del reporte desplegadas. Hay un atributo llamado *whenNoDataType* que permite decidir cómo debe mostrarse el documento generado cuando no hay datos en la fuente de datos proporcionada a este.

Existen tres posibilidades para elegir:

- **Documento vacío:** El documento generado no tendrá ninguna página. Los visualizadores podrían lanzar un error cuando intenten cargar el documento (*whenNoDataType = "NoPages"*).
- **Página en blanco:** El documento generado contendrá una sola página en blanco (*whenNoDataType = "BlankPage"*).
- **Desplegar todas las secciones:** En este caso todas las secciones del informe excepto la sección de detalle aparecerán en el documento final (*whenNoDataType = "AllSectionsNoDetail"*).

El valor por defecto para este atributo es *whenNoDataType = "NoPages"*

4.4.3.9. Colocación de las secciones *Title* y *Summary*

Si se quiere tener la sección del título y/o la sección del resumen desplegadas en páginas separadas, se deberá asignar el valor de "*true*" a uno o a los dos atributos siguientes: "*isTitleNewPage*" y "*isSummaryNewPage*".

Estos dos atributos son booleanos y su valor por defecto es "*false*".

Note que aun cuando se elige desplegar la sección de resumen en el espacio restante de la última página, una nueva página se iniciará automáticamente si el informe tiene más de una columna y la segunda columna ya se inició en esta última página.

4.4.3.10. Scriptlet Class

El atributo *scriptletClass* permite especificar el nombre de la clase scriptlet designada para el actual reporte. Si ningún valor se proporciona a este atributo, el motor generador de reportes de todos modos utilizará una instancia de la clase *dori.jasper.engine.JRDefaultScriptlet*.

4.5. DATOS DEL REPORTE

Cuando hablamos sobre el proceso de llenado del reporte, mencionamos que existen tres cosas que se deben proporcionar: el diseño, los valores de los parámetros y la fuente de datos. En la sección anterior vimos algunos aspectos sobre los diseños; ahora vamos a explicar más detenidamente los otros dos aspectos: los parámetros y la fuente de datos. Estos representan la única fuente de datos que el generador de reportes puede utilizar para el llenado del informe. Esta información se organizará de acuerdo a la plantilla definida en el diseño y producirá un documento listo para imprimir.

4.5.1. Expresiones

Las expresiones son una poderosa característica de JasperReports. Pueden usarse para declarar variables del reporte que realizan varios cálculos para agrupar los datos del reporte, para especificar los campos de texto que contiene el reporte o para después personalizar la apariencia de los objetos en el reporte.

Básicamente, todas las expresiones del informe son expresiones Java que pueden referenciar a los parámetros, campos y variables del reporte, utilizando una sintaxis especial.

En un diseño XML existen varios elementos que definen expresiones: *<variableExpression>*, *<initialValueExpression>*, *<groupExpression>*, *<printWhenExpression>*, *<imageExpression>*, *<textFieldExpression>*, entre otros.

Puesto que todas las expresiones de JasperReports son expresiones Java reales, se puede usar cualquier clase de Java, simplemente se debe tener en cuenta que la referencia a esta debe tener su nombre completo, es decir incluyendo el paquete al cual pertenece. También se debe tener en cuenta que las clases que se estén usando en las expresiones del informe estén disponibles en el *classpath* cuando se compile el reporte y cuando se llene este con los datos.

Las expresiones del reporte serían inútiles, si no existiera alguna manera de referenciar en estas los parámetros, campos o variables declaradas en el reporte. Afortunadamente, existe una sintaxis JasperReports especial que permite introducir estas referencias en las expresiones que se introducen en un diseño.

Para introducir referencias a los parámetros del reporte se usa los caracteres "*$P{ }*", como se indica en el siguiente ejemplo:

```
<textFieldExpression>
        $P{ReportTitle}
</textFieldExpression>
```

En este ejemplo se asume que existe un parámetro del reporte llamado *ReportTitle* declarado en el diseño, que es la clase *java.lang.String*. El campo del texto desplegará el valor de este parámetro cuando el reporte sea llenado.

Para usar una referencia de campo del reporte en una expresión, el nombre del campo debe ponerse entre los caracteres "*$F {*" y "*}*". Por ejemplo, para desplegar un campo de texto en el informe, que concatene dos valores presentados por la fuente de datos, se deberá definir una expresión, como la que sigue:

```
<textFieldExpression>
        $F{FirstName} + " " + $F{LastName}
</textFieldExpression>
```

La expresión puede ser aún más compleja:

```
<textFieldExpression>
$F{FirstName} + " " + $F{LastName} + " was hired on " +
(new SimpleDateFormat("MM/dd/yyyy")).format($F{HireDate}) + "."
</textFieldExpression>
```

Para referenciar una variable del informe en una expresión, debemos poner el nombre de la variable entre "$V {" y "}" como en el siguiente ejemplo:

```
<textFieldExpression>
        "Total quantity: " + $V{QuantitySum} + " kg."
</textFieldExpression>
```

Como se puede observar, la introducción de referencias de parámetros, campos y variables debido a la sintaxis especial de JasperReports son realmente objetos Java. Y conociendo la clase del parámetro, campo o declaración de variable en el diseño, es posible invocar a los métodos de los objetos referenciados en las expresiones.

Aquí, vemos cómo podemos extraer y desplegar la primera letra de un campo del informe de tipo *java.lang.String*:

```
<textFieldExpression>
        $F{FirstName}.substring(0, 1)
</textFieldExpression>
```

4.5.2. Parámetros

Los parámetros son referencias a objetos que son proporcionados durante el proceso de llenado del reporte. Estos son muy útiles para pasar al motor generador de reportes los datos que normalmente no se encontrar en la fuente de datos. Por ejemplo, podríamos pasar al generador de reportes, el nombre del usuario que ha lanzado la operación de llenado del reporte, para desplegar en el reporte el nombre del usuario, o para cambiar el título del informe dinámicamente.

```
<!ELEMENT parameter (parameterDescription?, defaultValueExpression?)>
<!ATTLIST parameter
        name NMTOKEN #REQUIRED
        class NMTOKEN #REQUIRED
        isForPrompting (true | false) "true"
>
<!ELEMENT parameterDescription (#PCDATA)>
<!ELEMENT defaultValueExpression (#PCDATA)>
```

Declarar un parámetro en el diseño es muy simple y este requiere especificar sólo su nombre y su clase, ejemplos:

```
<parameter name="ReportTitle" class="java.lang.String"/>
```

```
<parameter name="MaxOrderID" class="java.lang.Integer"/>
```

```
<parameter name="SummaryImage" class="java.awt.Image"/>
```

Los valores proporcionados de los parámetros pueden usarse en varias expresiones del reporte, en la consulta SQL del reporte o incluso en la clase scriptlet del reporte.

Estos son los componentes que definen por completo un parámetro:

- Nombre del parámetro
- Clase del parámetro
- Solicitando los valores de los parámetros
- Valor predefinido del parámetro

4.5.2.1. Nombre del parámetro

El atributo *name* del elemento *<parameter>* es obligatorio y permite referenciar el parámetro por medio de este nombre. Las denominaciones convencionales de JasperReports son similares a las denominaciones convencionales del lenguaje Java; esto significa que, el nombre del parámetro debe ser una sola palabra, sin ocupar caracteres especiales, como el punto o la coma.

4.5.2.2. Clase del parámetro

El segundo atributo obligatorio para un parámetro es el que especifica el nombre de la clase para los valores del parámetro. El atributo clase puede tener un valor como largo sea el nombre de la clase que representa, misma que debe estar disponible en el *classpath* para los procesos de compilación y llenado del reporte.

4.5.2.3. Solicitando los valores de los parámetros

En algunas aplicaciones GUI[3], sería útil tener una manera de establecer los valores de los parámetros del reporte, para que la aplicación le pida al usuario el ingreso de estos antes de lanzar el proceso de llenado del informe. También sería útil tener una manera de especificar una descripción de texto que intuya al usuario sobre el valor a ingresar para cada uno de los parámetros.

Esto es posible gracias al atributo booleano *isForPrompting* en la secuencia de la declaración del parámetro y el elemento interno del parámetro *<parameterDescription>*. En el siguiente ejemplo, podemos ver la declaración de un parámetro de texto, junto con la descripción que podría usar el runtime cuando requiera al usuario ingresar el valor del parámetro, en una ventana de diálogo personalizada:

```
<parameter name="Comments" class="java.lang.String" isForPrompting="true">
<parameterDesciption>
<![CDATA[
Please type here the report comments if any
]]>
</parameterDesciption>
</parameter>
```

4.5.2.4. Valor predefinido del parámetro

Los valores del parámetro son proporcionados al proceso de llenado del reporte de forma empaquetada en un objeto *java.util.Map* con el nombre del parámetro y su valor, de esta manera no se obliga a proporcionar un valor para cada parámetro. Si no se proporciona un valor para un parámetro, se considerará que su valor es nulo, pero esto no sucede si se especifica una expresión de valor predeterminado para un parámetro en particular del diseño. Esta expresión sólo se evalúa en caso de que no se proporcione un valor para el parámetro.

A continuación tenemos un ejemplo de un parámetro de tipo *java.util.Date* cuyo valor predefinido será la fecha actual, si no se proporciona un valor específico al momento de llenar el reporte:

[3] Graphical User Interface, es un programa que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

```
<parameter name="MyDate" class="java.util.Date">
<defaultValueExpression>
new java.util.Date()
</defaultValueExpression>
</parameter>
```

En la expresión de valor predefinido de un parámetro, solo podemos usar parámetros definidos previamente en el reporte.

4.5.2.5. Parámetros predefinidos

Cada diseño contiene algunos parámetros predefinidos, los cuales el programador de diseños de reportes decide introducir. Estos parámetros incorporados se describen a continuación.

Parámetro *REPORT_PARAMETERS_MAP*

Este es un parámetro integrado que siempre apunta a un objeto *java.util.Map* que contiene los parámetros definidos por el usuario y que se pasa cuando se llama al proceso de llenado del reporte.

Este parámetro es útil especialmente cuando se desea pasar a los subreportes el mismo conjunto de parámetros que recibió el reporte maestro o principal.

Parámetro *REPORT_CONNECTION*

Este parámetro apunta a un objeto *java.sql.Connection* que se proporciona al generador de reportes con el fin de utilizarlo para la ejecución de la consulta SQL del reporte a través de un JDBC, si es el caso.

Este tiene un valor diferente de "null" sólo si el reporte o subreporte ha recibido un *java.sql. Connection* cuando el proceso de llenado del reporte es invocado y no una instancia de *dori.jasper.engine.JRDataSource*.

Este también es útil para pasar a los subreportes el mismo objeto de conexión que se utilizó en el reporte maestro.

Parámetro *REPORT_DATASOURCE*

Al llenar un reporte, siempre tenemos un objeto de origen de datos ya sea directamente suministrado por la aplicación principal o creado entre bastidores por el motor generador de reportes cuando una conexión JDBC es suministrada. Este parámetro incorporado nos permite acceder a la fuente de datos del reporte en las expresiones del reporte o en los scriptlets.

Parámetro *REPORT_SCRIPTLET*

Incluso si el reporte no hace uso de scriptlets, este parámetro incorporado apunta a una instancia de *dori.jasper.engine.JRAbstractScriptlet*, que en este caso es un objeto *dori.jasper. engine.JRDefaultScriptlet*.

Pero cuando se utiliza scriptlets, esta referencia a la instancia de la clase scriptlet que se crea cuando se llena el reporte permitirá llamar a los métodos específicos de esta, para manipular o utilizar los datos que el objeto scriptlet ha preparado durante el proceso de llenado.

4.5.3. Origen de datos

Al llenar el reporte, el generador de reportes de JasperReports recorre los registros del objeto de origen de datos suministrado y genera todas las secciones de acuerdo con la plantilla de diseño definida.

Normalmente, el motor espera recibir un objeto *dori.jasper.engine.JRDataSource* como origen de datos del reporte que ha de llenar. Sin embargo, como veremos, existe una característica que permite a los usuarios proporcionar un objeto de conexión JDBC en lugar del objeto de origen de datos habitual cuando los datos del informe se encuentran en una base de datos relacional.

La interfaz *dori.jasper.engine.JRDataSource* es muy simple y tenemos que tratar con sólo dos métodos si queremos ponerla en práctica:

```
public boolean next() throws JRException;
```

El método *next ()* se llama en el objeto de origen de datos mediante el motor generador de reportes cuando se itera con los datos, durante el tiempo de llenado del informe.

```
public Object getFieldValue(JRField jrField) throws JRException;
```

El segundo método proporciona un listado de valores para cada campo del informe en el registro actual del origen de datos.

Es muy importante saber que la única manera de recuperar los datos de la fuente de datos es mediante el uso de los campos del reporte. Un objeto de origen de datos se parece más a una tabla con filas y columnas que contiene los datos en las celdas. Las filas de esta tabla son los registros a través del cual el generador de reportes itera cuando se llena el informe y cada columna se debe asignar a un campo del reporte, por lo que podemos hacer uso del contenido de origen de datos en las expresiones del reporte.

Hay varias implementaciones por defecto de la interfaz *dori.jasper.engine.JRDataSource* y vamos a echar un vistazo a cada una de ellas:

4.5.3.1. Clase *dori.jasper.engine.JRResultSetDataSource*

Esta es una implementación muy útil de la interfaz *dori.jasper.engine.JRDataSource*, porque envuelve un objeto *java.sql.ResultSet*. Como la mayoría de los reportes se generan usando datos ubicados en bases de datos relacionales, es muy probable que esta sea la implementación más utilizada para la interfaz de origen de datos.

Lo que es interesante saber es que se podría terminar usando incluso no la instancia de esta clase, al rellenar sus reportes. Esto es lo que sucede:

Si opta por especificar en el diseño, la consulta SQL para recuperar los datos del reporte de ciertas tablas ubicadas en una base de datos relacional, el motor generador de reportes hará la ejecución de la consulta SQL especificada, incluyendo el retorno de un objeto *java.sql.ResultSet* como una instancia de *dori.jasper.engine.JRResultSetDataSource*. Lo único que el generador de reportes necesitaría es un objeto *java.sql.Connection* para que se ejecute la consulta como se ha mencionado. Que este objeto de conexión sea suministrado en lugar de suministrar el objeto de origen de datos habitual.

Por supuesto, usted puede ejecutar la consulta SQL en la aplicación padre, fuera de JasperReports, si se quiere o la situación le obliga. Y luego de forma manual cerrar el *java.sql.ResultSet* obtenido, utilizando una instancia de esta clase de origen de los datos, antes de llamar al proceso de llenado reporte.

Lo más importante que debe saber cuándo se utiliza este tipo de fuente de datos es que hay que declarar un campo del reporte para cada columna del conjunto de resultados. El nombre del campo del reporte tiene que ser igual al nombre de la columna que se asigna así como el tipo de dato. Para una máxima portabilidad, como se indica en la documentación de JDBC, los valores de un objeto *java.sql.ResultSet* deben ser recuperados, de izquierda a derecha y de una sola vez, así que para hacer eso podría considerar la declaración de los campos de informe en el mismo orden que aparecen en la consulta SQL.

4.5.3.2. Clase *dori.jasper.engine.JREmptyDataSource*

Como ya se mencionó, esta implementación es muy útil, ya que permite la generación de reportes en los que los datos no necesariamente provienen de la fuente de datos, pero probablemente a partir de los parámetros y sólo el número de los registros virtuales en la fuente de datos es importante.

Muchos de los ejemplos proporcionados, tales como *fonts*, *images*, *shapes* y *unicode* utilizan una instancia de esta clase al rellenar los reportes, para simular una fuente de datos con un registro, pero con todos los campos "*null*".

4.5.3.3. Clase *dori.jasper.engine.data.JRTableModelDataSource*

Esta implementación predeterminada de la interfaz *dori.jasper.engine.JRDataSource* que envuelve un objeto *javax.swing.table.TableModel* y puede ser utilizada en aplicaciones Swing de Java para generar reportes a partir de los datos que hayan sido cargados en las tablas que aparecen en pantalla.

Existen dos maneras de utilizar este tipo de fuente de datos:

Normalmente, con el fin de recuperar datos de sí mismo, hay que declarar un campo en el reporte para cada columna del objeto *javax.swing.table.TableModel*, mostrando el mismo

nombre de la columna asignada. Pero a veces esto es imposible ya que los campos del reporte tienen que respetar las convenciones de nomenclatura de Java para declarar variables y columnas de la tabla.

Afortunadamente, todavía se puede asignar como columnas a los campos del informe usando su índice en lugar de sus nombres. Por ejemplo, una columna de la tabla denominada "Product Description" nunca podría ser asignada a un campo de informe denominado "Product Description", porque al momento de compilar el informe se lanzara un error que diga que no se pude utilizar espacios en el nombre de campo del reporte. Pero si se sabe con exactitud que en el modelo de objetos de la tabla esta columna en particular es la tercera (*índice = 2*), entonces se podría nombrar al correspondiente campo como "column2" y utilizar los datos de la columna sin problemas.

4.5.3.4. Clase *dori.jasper.engine.data.JRBeanArrayDataSource*

Esta implementación que también se suministra con la librería envuelve un conjunto de JavaBeans y utiliza la reflexión para recuperar los valores de los campos del reporte.

Un objeto JavaBean representa un registro del tipo de la fuente de datos. Si tenemos un campo del informe de nombre "ProductDescription", al recuperar el valor de este campo, el programa intentará llamar a través de la reflexión un método llamado *getProductDescription ()* del objeto JavaBean actual.

Para los campos booleanos, el programa también tratará con el prefijo "*is*", si el prefijo habitual "get" para las propiedades del objeto JavaBean no devuelve un valor, como se indica en las especificaciones JavaBeans.

4.5.3.5. Clase *dori.jasper.engine.data.JRBeanCollectionDataSource*

Esta implementación de la interfaz *dori.jasper.engine.JRDataSource* es muy similar a la anterior que utiliza la reflexión y la nomenclatura JavaBeans, pero envuelve un objeto *java.util. Collection* en lugar de una matriz de objetos JavaBean.

4.5.4. Consulta del reporte

Con el fin de llenar un reporte, tenemos que proporcionar al motor generador de reportes los datos del reporte, o por lo menos instrucciones sobre cómo obtener esta información.

JasperReports normalmente espera recibir un objeto *dori.jasper.engine.JRDataSource* como fuente de datos del reporte, pero también fue mejorado para trabajar con JDBC de modo que pueda recuperar datos desde bases de datos relacionales, si es necesario.

La librería permite a los usuarios especificar en el diseño, la consulta SQL que debe ejecutarse en tiempo de ejecución para recuperar los datos del reporte, si los datos se encuentran en una base de datos relacional.

La consulta SQL especificada en el diseño se toma en cuenta y ejecuta sólo si un objeto *java.sql.Connection* es suministrado en lugar del objeto *dori.jasper.engine.JRDataSource* para llenar el reporte.

Esta consulta se puede introducir en el diseño XML utilizando el elemento *<queryString>*. Si está presente, este elemento viene después de las declaraciones de los parámetros del reporte y antes de los campos del reporte.

```
<!ELEMENT queryString (#PCDATA)>
```

Aquí una simple consulta SQL que recupera datos desde una tabla llamada "pedidos" situada en una base de datos relacional:

```
<queryString><![CDATA[SELECT * FROM pedidos]]></queryString>
```

Un aspecto importante es el uso de parámetros en la cadena de consulta del reporte, con el fin de poder personalizar aún más el conjunto de datos recuperados de la base de datos. Estos parámetros pueden actuar como filtros dinámicos en la consulta que proporciona los datos para el informe y se introducen utilizando una sintaxis especial, similar a la utilizada en las expresiones del reporte.

Hay dos posibles maneras de utilizar parámetros en la consulta:

1. Los parámetros son utilizados como parámetros normales *java.sql.PreparedStatement*, usando la siguiente sintaxis:

```
<queryString>
<!CDATA[ SELECT * FROM Orders WHERE OrderID <= $P{MaxOrderID} ORDER BY
ShipCountry ]]>
</queryString>
```

2. A veces es útil el uso de parámetros para modificar dinámicamente partes de la consulta SQL o para pasar toda la consulta SQL como un parámetro a las rutinas de llenado del reporte. En este caso, la sintaxis difiere un poco, como en el ejemplo siguiente, nótese el carácter "!":

```
<queryString>
<!CDATA[ SELECT * FROM $P!{MyTable} ORDER BY $P!{OrderByClause}]]>
</queryString>
```

¿Qué es diferente en este segundo ejemplo? Los parámetros utilizados hacen de nombre de la tabla faltante en la cláusula FROM y los nombres de las columnas que faltan en la cláusula ORDER BY. No se puede usar IN en los parámetros para cambiar dinámicamente partes de la consulta que se ejecuta utilizando un objeto *java.sql.PreparedStatement*.

La sintaxis especial que introduce los valores del parámetro en este ejemplo, se aseguran que el valor que nosotros proporcionamos para esos parámetros reemplacen las referencias del

parámetro en la consulta, antes de que se envíe al servidor de base de datos utilizando un objeto *java.sql.PreparedStatement*.

De hecho lo que pasa es que el generador de reportes primero trata con las referencias del parámetro "$P! { }" para obtener la forma final de la consulta SQL y sólo después de eso transformará el resto de la referencia normal del parámetro "$P { }" en el parámetro usual IN usado cuando trabajamos con instrucciones JDBC preparadas.

Para más detalles sobre qué tipo de parámetros se han de utilizar en las consultas del reporte, se tiene que estar más familiarizado con la tecnología JDBC y especialmente con el uso de la interfaz de *java.sql.PreparedStatement* y sus parámetros.

Este segundo tipo de referencia de los parámetros utilizados en la consulta SQL permite pasar toda la consulta SQL en tiempo de ejecución, si lo desea:

```
<queryString>$P!{MySQLQuery}</queryString>
```

No se puede poner otras referencias de parámetros en un valor de parámetro. Es decir, cuando suministro toda la consulta SQL como un parámetro del informe, el generador de reportes no espera encontrar referencias de parámetros en la consulta que se envía a la misma y considera esta consulta para finalizar y enviarla "tal cual", al servidor de base de datos.

4.5.5. Campos

Los campos del reporte representan la única forma de asignar los datos de la fuente de datos en el diseño y utilizar estos datos en las expresiones del reporte, para obtener el resultado deseado.

Al declarar los campos del reporte, se tiene que asegurar que el origen de datos que se suministra durante el proceso de llenado del reporte será capaz de proporcionar los valores a todos los campos.

Por ejemplo, en caso de utilizar la implementación *dori.jasper.engine.JRResultSetDataSource*, esto ocurre cuando la consulta del reporte utiliza SQL, entonces se debe asegurar que no tenga una columna para cada campo en el conjunto de resultados obtenidos tras la ejecución de la consulta. La columna correspondiente tiene que llevar el mismo nombre y el mismo tipo de dato como el campo que se asigna.

```
<!ELEMENT field (fieldDescription?)>
<!ATTLIST field
name NMTOKEN #REQUIRED
class (java.lang.Object | java.lang.Boolean | java.lang.Byte |
java.util.Date | java.sql.Timestamp | java.lang.Double | java.lang.Float |
java.lang.Integer | java.io.InputStream | java.lang.Long | java.lang.Short |
java.math.BigDecimal | java.lang.String) "java.lang.String"
>
<!ELEMENT fieldDescription (#PCDATA)>
```

Aquí está un pequeño ejemplo que muestra los campos que tienen que declararse para asignar las columnas de una tabla determinada de una base de datos. Vamos a considerar una tabla llamada "empleados", con la siguiente estructura:

COLUMNA	TIPO	TAMAÑO
EmployeeID	int	4
LastName	varchar	50
FirstName	varchar	50
HireDate	datetime	8

Fuente:[L01]

Tabla 4.1. Ejemplo de tabla empleados

Los campos del reporte deben declarar el campo como se muestra a continuación:

```
<field name="EmployeeID" class="java.lang.Integer"/>
<field name="LastName" class="java.lang.String"/>
<field name="FirstName" class="java.lang.String"/>
<field name="HireDate" class="java.util.Date"/>
```

Si declaramos un campo que no tiene una columna correspondiente en el conjunto de resultados, una excepción será lanzada en tiempo de ejecución. Las columnas que se presentan en el conjunto de resultados producido por la ejecución de la consulta SQL que no tienen campos correspondientes en el diseño, no afectan el proceso de llenado del reporte, pero también no será accesible su visualización en el reporte.

Éstos son los componentes que debe tener la definición de un campo del reporte:

4.5.5.1. Nombre del Campo

El atributo *name* del elemento *<field>* es obligatorio y permite hacer referencia al campo desde las expresiones del reporte por medio del nombre, que debe ser una sola palabra sin caracteres especiales, como el punto o la coma.

4.5.5.2. Clase del Campo

El segundo atributo de un campo delreporte es el que especifica el nombre de la clase para los valores del campo. Su valor por defecto es *java.lang.String*, pero se puede cambiar a uno de los siguientes nombres de clases aceptables:

- java.lang.Object
- java.lang.Boolean
- java.lang.Byte
- java.util.Date
- java.sql.Timestamp
- java.lang.Double
- java.lang.Float
- java.lang.Integer
- java.io.InputStream
- java.lang.Long
- java.lang.Short
- java.math.BigDecimal

[L01] The JasperReports Ultimate Guide, Teodor Danciu.

Cuando se trabaja con fuentes de datos personalizadas que tienen campos con valores que son instancias de clases personalizadas, el nombre de clase de estos campos debe ser *java.lang.Object*, porque a diferencia de las definiciones de los parámetros, sólo se puede elegir un nombre de clase de la lista anterior. Cuando se trabaja con los campos de las expresiones del reporte, puede convertirla a su clase conocida, asegurándose de que la clase está disponible en el *classpath*, tanto en el momento de compilación del reporte como en el momento de llenado del reporte.

4.5.5.3. Descripción del Campo

Este trozo de texto adicional que puede acompañar a un campo puede ser muy útil al momento de implementar una fuente de datos personalizada. En este atributo, podemos almacenar una clave o cualquier información que se pueda necesitar para recuperar el valor del campo desde el origen de datos personalizado, en tiempo de ejecución.

Mediante el uso opcional de este elemento *<fieldDesciption>*, se puede fácilmente superar las restricciones respecto a las convenciones de nombres del campo; usando la descripción del campo en lugar del nombre del campo cuando se recuperen los valores del campo desde la fuente de datos.

```
<field name="PersonName" class="java.lang.String" isForPrompting="true">
<fieldDesciption>PERSON NAME</fieldDesciption>
</field>
```

4.5.6. Variables

Las variables de reporte son objetos especiales declarados en la parte superior de una expresión del reporte. Pueden ser utilizadas para simplificar el diseño, al declarar sólo una vez una expresión que se usa mucho en todo el diseño o para llevar a cabo varios cálculos en la expresión correspondiente.

En la expresión, una variable puede hacer referencia a otras variables del reporte, pero sólo si las variables de referencia se han definido previamente en el diseño. Por lo tanto, el orden en que las variables se declaran en un el diseño es muy importante.

```
<!ELEMENT variable (variableExpression?, initialValueExpression?)>
<!ATTLIST variable
   name NMTOKEN #REQUIRED
   class NMTOKEN "java.lang.String"
   resetType (None | Report | Page | Column | Group) "Report"
   resetGroup CDATA #IMPLIED
   calculation (Nothing | Count | Sum | Average | Lowest | Highest |
   StandardDeviation | Variance | System) "Nothing"
>
<!ELEMENT variableExpression (#PCDATA)>
<!ELEMENT initialValueExpression (#PCDATA)>
```

4.5.6.1. Variable *name*

Al igual que para los parámetros y los campos, el atributo *name* del elemento *<variable>* es obligatorio y permite hacer referencia a la variable por su nombre declarado en las expresiones de reporte. Las convenciones de nomenclatura que hemos mencionado para los parámetros y los campos se aplican también a las variables.

4.5.6.2. Variable *class*

El atributo *class* contiene el nombre de clase a la que pertenece los valores de la variable. La clase por defecto es *java.lang.String*, pero puede declarar variables del reporte de cualquier clase que se desee, siempre y cuando la clase que usted eligió este disponible en el *classpath*, tanto para el proceso de compilación como para el proceso de llenado del reporte.

4.5.6.3. Reseteo de variable

El valor de una variable de reporte puede cambiar con cada iteración, pero puede ser retornada a su valor original con la expresión de valor inicial en los momentos de tiempo especificado, durante el proceso de llenado del reporte. Este comportamiento se controla usando el atributo *resetType*, el cual indica cuando la variable debe ser reiniciado durante el proceso de llenado del reporte.

Hay cinco formas de reiniciar una variable:

- **No Reseteo.-** La variable no se iniciará con la expresión de valor inicial y sólo se contienen los valores de obtener mediante la evaluación de la expresión de la variable (*resetType = "None"*).
- **Reseteo a nivel del reporte.-** La variable se inicializa una sola vez, al comienzo del proceso de llenado del reporte, con el valor retornado por el valor de la expresión de la variable inicial (*ResetType = "Report"*).
- **Reseteo a nivel de página.-** La variable se reinicializa al comienzo de cada nueva página (*ResetType = "Page"*).
- **Reseteo a nivel de columna.-** La variable se reinicializa al comienzo de cada nueva columna (*ResetType = "Column"*).
- **Reseteo a nivel de grupo.-** La variable se reinicializa cada vez que el grupo especificado por el atributo *resetGroup* es invocado (*resetType = "Group"*).

El valor predeterminado para este atributo es *resetType = "Report"*.

4.5.6.4. *ResetGroup*

Si está presente, el atributo *resetGroup* contiene el nombre de un grupo de reportes y sólo funciona en combinación con el atributo *resetType*, cuyo valor debe ser *resetType = "Group"*.

4.5.7. Cálculos

Como se mencionó, las variables pueden realizar varios tipos de cálculos en sus correspondientes valores de expresión. Aquí están todos los posibles valores para el atributo *calculation* del elemento de <*variable*>:

4.5.7.1. Cálculo *nothing*

Este es el tipo de cálculo por defecto que una variable lleva a cabo. Esto significa que el valor de la variable se actualiza con cada iteración con el origen de datos y que el valor retornado se obtiene mediante una simple evaluación de la expresión de la variable.

4.5.7.2. Cálculo *count*

Un contador de veces que siempre contarán para los valores no nulos retornados después de evaluar la expresión principal de la variable con cada iteración del origen de datos. Las variables contadores deben ser siempre de tipo numérico, pero pueden tener expresiones no numéricas como su expresión principal, ya que al generador de reportes no le importa el tipo de expresión, pero sólo cuenta los valores retornados no nulos, independientemente de su tipo.

Sólo, el valor de la expresión de la variable inicial debe ser también numérico y compatible con el tipo de la variable, ya que este valor será asignado directamente a la variable de conteo cuando se inicie.

4.5.7.3. Cálculo *sum*

El motor generador de reportes puede sumar todos los valores retornados por expresión de variable principal, si usted elige este tipo de cálculo; pero se debe asegurar que la variable sea de tipo numérico. No podemos calcular la suma de una variable de reporte tipo *java.lang.String* o *java.util.Date*.

4.5.7.4. Cálculo *average*

El motor generador de reportes también puede calcular el promedio de una serie de valores obtenidos mediante la evaluación de la expresión de la variable para cada registro del origen de datos. Este tipo de cálculo también se puede realizar sólo para las variables numéricas.

Con el fin de calcular el promedio, el generador de reportes genera por debajo una variable de reporte de ayuda que calcula la suma de los valores y la utiliza para calcular el promedio de estos valores. Esta variable de ayuda recibe su nombre desde la correspondiente variable promedio, con el sufijo "_SUM".

Por ejemplo, si se declara una variable de cálculo promedio llamada *MyAverageVariable* para una expresión numérica, el motor de reportes también creará una variable de reporte *MyAverageVariable_SUM* para ayudar a calcular el promedio.

4.5.7.5. Cálculos *lowest* y *highest*

Se puede elegir este tipo de cálculo cuando se quiere obtener el valor más bajo o el valor más alto de una serie de valores obtenidos mediante la evaluación de la expresión de una variable para cada registro del origen de datos.

4.5.7.6. Cálculos *standardDeviation* y *variance*

En algunos reportes, es posible que se desee usar tipos más avanzados de cálculos en una expresión numérica y JasperReports ha incorporado algoritmos para obtener la desviación estándar y la varianza de una serie de valores retornados por la evaluación de la expresión de una variable del reporte.

Al igual que para las variables que calcula el promedio, el generador de reportes crea y utiliza las variables de ayuda para primero obtener el valor de la suma y el número correspondientes a la serie de valores actual. El nombre de las variables auxiliares que se crean entre bastidores se obtiene adicionando el sufijo "*_SUM*" o el sufijo "*_COUNT*" a la variable de usuario y que pueden ser utilizadas en las expresiones de otra variable del reporte.

Para las variables que permiten calcular la desviación estándar, siempre hay una variable de ayuda presente, que calcula en primer lugar la variación de la serie de valores y su nombre tiene añadido el sufijo "*_VARIANCE*".

4.5.7.7. Cálculo *system*

Este tipo de cálculo se puede elegir sólo cuando no se desea que el generador de reportes para cuidar de calcular un valor para la variable. Esto significa que el valor se está calculando de la misma variable, casi con toda seguridad utilizando la funcionalidad de scriptlets de JasperReports.

Para este tipo de cálculo, lo único que el generador de reportes no conserva es el valor que se ha calculado, a partir de una iteración del origen de datos, a la siguiente. Ejemplos:

Esta es una declaración de variable de reporte simple que calcula la suma de un campo numérico del reporte denominado "*Quantity*":

```
<variable name="QuantitySum" class="java.lang.Double" calculation="Sum">
<variableExpression>$F{Quantity}</variableExpression>
</variable>
```

Si queremos tener la suma de este campo para cada página, aquí está la declaración de la variable completa necesaria:

```
<variable name="QuantitySum" class="java.lang.Double" resetType="Page"
calculation="Sum">
<variableExpression>$F{Quantity}</variableExpression>
<initialValueExpression>new Double(0)</initialValueExpression>
</variable>
```

En el ejemplo anterior, la variable suma de la página se iniciará con cero al comienzo de cada página nueva.

4.5.8. Variables integradas al reporte

También existen las siguientes variables internas del sistema, listas para usarse en las expresiones:

4.5.8.1. Variable *PAGE_NUMBER*

Esta variable contiene como valor el número de página actual. Para el final del proceso de llenado del reporte, esta contendrá el número total de páginas del documento final.

Se puede utilizar para mostrar tanto el número de la página actual y el número total de páginas usando una característica especial de los elementos campo de texto de JasperReports, esta es el atributo *evaluationTime*. Esto se puede apreciar en la mayoría de ejemplos proporcionados por la librería JasperReports.

4.5.8.2. Variable *COLUMN_NUMBER*

Variable incorporada que contiene el número de columna actual. Por ejemplo, en un reporte con tres columnas, en la segunda página probablemente se tendrá las columnas 4, 5 y 6, si los atributos como *isTitleNewPage* entre otros más no intervienen.

4.5.8.3. Variable *REPORT_COUNT*

Después de acabar la iteración a través de la fuente de datos, esta variable de reporte contiene el número total de registros que fueron procesados.

4.5.8.4. Variable *PAGE_COUNT*

Esta variable contiene el número de registros que se procesaron durante la generación de la actual página.

4.5.8.5. Variable *COLUMN_COUNT*

Esta variable contiene el número de registros que se procesaron durante la generación de la actual columna.

4.5.8.6. Variable *GroupName_COUNT*

Cuando se declara un grupo del reporte, el generador automáticamente crea una variable de conteo que calculará el número de registros que conforman el actual grupo actual.

El nombre de esta variable proviene del nombre de grupo correspondiente con el sufijo "*_COUNT*". Esta se la puede utilizar como cualquier otra variable del reporte, en cualquier expresión del reporte.

4.6. SECCIONES DEL REPORTE

Durante la elaboración del diseño de un reporte se tiene que definir el contenido y las propiedades de las secciones. Toda la estructura del diseño se basa en las siguientes secciones: *<title>, <pageHeader>, <columnHeader>, <groupHeader>, <detail>, <groupFooter>, <columnFoter>, <pageFooter>* y *<summary>*.

Las secciones son fragmentos de la plantilla del informe que tienen una determinada altura y ancho y pueden contener elementos del reporte como líneas, rectángulos, imágenes o campos de texto. Las secciones son llenadas en repetidas ocasiones durante el tiempo de generación del reporte y hacer el documento final que se está produciendo. Al declarar el contenido y características de una sección del reporte, en un diseño XML, se debe utilizar el elemento genérico *<band>*.

```
<!ELEMENT band (printWhenExpression?, (line | rectangle | image | staticText
| textField | subreport | elementGroup)*)>
<!ATTLIST band
   height NMTOKEN "0"
>
<!ELEMENT printWhenExpression (#PCDATA)>
```

Las secciones del reporte son denominadas a veces como las bandas del reporte y representan una característica que casi todas las herramientas de generación de reportes tienen y se usan de la misma manera.

Atributo *height*

El atributo *height* disponible en la declaración de una banda del reporte, específica la altura en píxeles de esa banda en particular y es muy importante en el diseño global del reporte.

Los elementos contenidos en una determinada banda del reporte siempre deben caber en las dimensiones definidas de la banda, para evitar posibles malos resultados al momento de generar los reportes. El generador de reportes emite una advertencia si encuentra elementos fuera de los límites establecidos de la banda, cuando se compilan los diseños.

Ignorando bandas

Todas las secciones del reporte permiten definir una expresión de reporte que será evaluado en tiempo de ejecución con el fin de decidir si esa sección en particular debe ser generada o ignorada al momento de generar el documento.

Esta expresión es introducida por la *<printWhenExpression>* que está disponible en cualquier elemento *<band>* del diseño XML y siempre retornará un objeto *java.lang.Boolean* o *null*.

4.6.1. Secciones principales

Un diseño pequeño no puede contener todas las secciones del reporte, porque cada una de estas es opcional. Pero como un diseño pequeño no va a producir documentos muy interesantes.

```
<!ELEMENT title (band?)>
<!ELEMENT pageHeader (band?)>
<!ELEMENT columnHeader (band?)>
<!ELEMENT detail (band?)>
<!ELEMENT columnFooter (band?)>
<!ELEMENT pageFooter (band?)>
<!ELEMENT summary (band?)>
```

4.6.1.1. *Title*

Esta es la primera sección del reporte. Esta se genera sólo una vez durante el proceso de llenado del reporte y va al inicio del documento final.

Siendo la primera sección del reporte significa que va a preceder incluso la sección de encabezado de página. Para cuando se quiera tener el encabezado de la página impresa antes de la sección del título, se tendrá que copiar los elementos presentes en el encabezado de la página también al inicio de la sección título. Se podría suprimir el encabezado de la página actual en la primera página con *<printWhenExpression>*, basado en la variable del reporte *PAGE_NUMBER*.

4.6.1.2. *Page Header*

Esta sección aparece en la parte superior de cada página del documento final.

4.6.1.3. *Column Header*

Esta sección aparece en la parte superior de cada columna del documento final.

4.6.1.4. *Detail*

Para cada registro existente en el origen de datos, el motor generador de reportes trata de generar esta sección.

4.6.1.5. *Column Footer*

Esta sección aparece en la parte inferior de cada columna del documento final. Nunca se extiende hacia abajo para visualizar el contenido de los elementos de texto que contiene y será siempre de una altura fija.

4.6.1.6. *Page Footer*

Esta sección aparece en la parte inferior de cada página del documento final. Al igual que la sección de la columna de pie de página, nunca el pie de página se extiende hacia abajo para mostrar el contenido de los elementos que contiene y será siempre de una altura fija.

4.6.1.7. *Summary*

Esta sección se genera sólo una vez en el reporte y aparece al final del documento final, pero no es necesariamente la última sección que se genera. Eso es porque en algunos casos, el pie de la columna y/o pie de página de la última página puede seguir.

La sección de resumen puede empezar en una nueva página, al establecer el atributo *isSummaryNewPage* a "*true*". Incluso si el atributo es "*false*", la sección de resumen siempre comenzará en una nueva página, si los elementos que contiene no caben en el espacio restante de la última página o si el reporte tiene más de una columna y en la última página ya ha iniciado una segunda columna.

Si las secciones de reportes principales que hemos visto no son suficientes, se puede considerar la introducción de secciones complementarias como grupos de encabezados y grupos de pies de páginas.

4.6.2. Agrupando datos

Los grupos representan una forma flexible de organizar los datos en un reporte. Un grupo del reporte está representado por la secuencia de registros consecutivos en el origen de datos que tienen algo en común, como el valor de un cierto campo del reporte, por ejemplo.

Un grupo del reporte tiene tres componentes:

- Expresión del grupo.
- Sección de encabezado del grupo.
- Sección de pie de página del grupo.

El valor de la expresión asociada al grupo es lo que hace que los registros del grupo sean más estrictos. Este valor es lo que tienen en común. Cuando el valor de la expresión del grupo cambia durante la iteración a través de la fuente de datos al momento de llenar el reporte, se produce una ruptura del grupo y las correspondientes secciones de grupo *<groupFooter>* y *<groupHeader>* son insertadas en el documento final.

Podemos tener tantos grupos como se desee en un reporte. El orden de los grupos declarados en un diseño es importante, porque cada grupo contiene a otro. Un grupo contiene el grupo siguiente y así sucesivamente; y cuando un grupo padre encuentra una ruptura, todos los grupos contenidos son reinicializados.

La agrupación de datos funciona como se espera cuando los registros del origen de datos ya están ordenados de acuerdo a las expresiones de grupos utilizados en el reporte. Por ejemplo, si se desea agrupar algunos productos por país y ciudad del fabricante, el generador de reportes espera para buscar los registros del origen de datos ya ordenados por país y ciudad.

Si no, usted debe esperar para buscar los registros que pertenecen a un determinado país o ciudad en diferentes partes del documento resultante, ya que JasperReports no ordena la fuente de datos para que usted, antes de usarla.

```
<!ELEMENT group (groupExpression?, groupHeader?, groupFooter?)>
<!ATTLIST group
        name NMTOKEN #REQUIRED
        isStartNewColumn (true | false) "false"
        isStartNewPage (true | false) "false"
        isResetPageNumber (true | false) "false"
        isReprintHeaderOnEachPage (true | false) "false"
        minHeightToStartNewPage NMTOKEN "0"
>
<!ELEMENT groupExpression (#PCDATA)>
<!ELEMENT groupHeader (band?)>
<!ELEMENT groupFooter (band?)>
```

4.6.2.1. Nombre del grupo

El nombre identifica de forma única al grupo y se lo utiliza en otros atributos XML, cuando se quiere hacer referencia a un grupo del reporte determinado. El nombre de un grupo es obligatorio y obedece a la misma convención de nombres que hemos mencionado para los parámetros, campos y variables del reporte.

4.6.2.2. Iniciando una nueva página o columna

A veces es útil introducir un salto de página o columna, cuando se inicia un nuevo grupo, probablemente debido a que ese grupo es más importante y debe comenzar en una página o una columna propia. Para indicar al generador de reportes que debe comenzar una nueva página o columna para un grupo determinado, en lugar de imprimir el espacio que queda en la parte inferior de la página o columna, se debe establecer el valor del atributo *isStartNewPage* o *isStartNewColumn* en "*true*", respectivamente.

Esos dos atributos son los únicos en toda la librería que permiten introducir voluntariamente saltos de página. En todas las otras situaciones, el generador de reportes presenta saltos de página de forma automática, si es necesario.

Sin embargo, si no se quiere introducir constantemente saltos de página o de columna para un grupo en particular, sino que si el espacio que queda en la parte inferior de la página o columna es demasiado pequeño, se deberá considerar el uso del atributo *minHeightToStartNewPage*. Este atributo especifica la cantidad mínima de espacio libre vertical necesaria para que el grupo no inicie una nueva página por su cuenta.

4.6.2.3. Restablecer el número de página

Si es necesario, los grupos del reporte tienen el poder de restablecer la variable del reporte integrada que contiene el número de página actual (*PAGE_NUMBER*). Esto puede lograrse estableciendo el atributo *isResetPageNumber* a "*true*".

4.6.2.4. Header del grupo

Esta sección es la que marca el inicio de un nuevo grupo en el documento final y se inserta en el documento cada vez que el valor de la expresión de grupo cambia durante la iteración a través del origen de datos.

4.6.2.5. Footer del grupo

Cada vez que un grupo del reporte cambia, el generador de reportes añade la sección de pie de página correspondiente a los nuevos grupos iniciados anteriormente o cuando el reporte finaliza.

4.7. SCRIPTLETS

Todos los datos mostrados en un reporte provienen de los parámetros y de los campos del reporte. Estos datos pueden ser procesados utilizando las variables y expresiones del reporte. Hay momentos específicos en el tiempo cuando una variable se encuentra procesando. Algunas variables se inicializan de acuerdo a su tipo de reseteo cuando el reporte se inicia, o cuando un salto de página o columna es encontrado, o cuando se cambia de grupo. Además, las variables se evalúan cada vez que los nuevos datos se obtienen de la fuente de datos.

Sin embargo, sólo las expresiones de variables simples no siempre implementan una funcionalidad compleja. Aquí es donde intervienen los scriptlets. Los scriptlets son secuencias de código Java que se ejecutan cada vez que ocurre un evento en el reporte. A través de los scriptlets, los usuarios tienen la posibilidad de afectar a los valores almacenados en las variables del reporte. Los scriptlets trabajan principalmente con variables de reportes y es muy importante tener un control total sobre el scriptlet desde el momento exacto que se ejecuta.

JasperReports permite la ejecución de código Java antes o después de inicializadas las variables del reporte de acuerdo a su tipo de reseteo: *Report*, *Page*, *Column* o *Group*. Con el fin de hacer uso de esta funcionalidad, los usuarios sólo tienen que crear una clase scriptlet extendiendo una de las dos clases siguientes:

- dori.jasper.engine.JRAbstractScriptlet
- dori.jasper.engine.JRDefaultScriptlet

El nombre completo de esta clase scriptlet personalizada se tiene que especificar en el atributo *scriptletClass* del elemento *<jasperReport>* y tiene que estar disponible en el classpath, en el momento de llenado del reporte, para que el generador de reportes pueda crear una instancia de esta. Si no se especifica un valor para el atributo *scriptletClass*, el generador de reportes instancia la clase *JRDefaultScriptlet*.

Al crear una clase scriptlet de JasperReports, existen varios métodos que los desarrolladores deben implementar o invalidar, como: *beforeReportInit (), afterReportInit (), beforePageInit (), afterPageInit (), beforeGroupInit (), afterGroupInit ()*, etc. Estos métodos serán llamados por el generador de reportes en el momento apropiado, al llenar el reporte.

Para reportes más complejos, es necesario utilizar la gran complejidad de las expresiones del reporte para agrupar o visualizar los datos, tal vez se deba considerar la transferencia de esta complejidad a una clase diferente para realizar las llamadas desde las expresiones del reporte de forma simplificada. La clase scriptlet es ideal para transferir esta complejidad porque el generador de reportes le proporciona una referencia al objeto scriptlet que se crea con el parámetro incorporado *REPORT_SCRIPTLET*.

4.8. ELEMENTOS DEL REPORTE

Los reportes generados estarían vacíos si no se ponen algunos elementos de reportes en el diseño. Los elementos del reporte pueden mostrar objetos como: textos estáticos, campos de texto, imágenes, líneas o rectángulos, que se ponen en las secciones de diseño del reporte para que aparezcan en el documento generado.

Los elementos del reporte pueden ser de dos tipos:

- **Elementos de texto.-** textos estáticos y campos de texto que despliegan contenido dinámico.
- **Elementos gráficos:** líneas, rectángulos e imágenes.

Cuando se agrega un elemento del reporte en una de las secciones del reporte, se tiene que especificar la posición relativa del elemento en la sección en particular y su tamaño, junto con otras propiedades generales de los elementos del reporte como: el color, transparencia, elasticidad, etc.

4.8.1. Propiedades generales de elementos

Las propiedades que son comunes a todos los tipos de elementos del reporte se han agrupado en la etiqueta *<reportElement>* que pueden aparecer en la declaración de todos los elementos del reporte.

```
<!ELEMENT reportElement (printWhenExpression?)>
<!ATTLIST reportElement
   positionType (Float | FixRelativeToTop | FixRelativeToBottom)
"FixRelativeToTop"
   isPrintRepeatedValues (true | false) "true"
   mode (Opaque | Transparent) #IMPLIED
   x NMTOKEN #REQUIRED
   y NMTOKEN #REQUIRED
   width NMTOKEN #REQUIRED
   height NMTOKEN #REQUIRED
   isRemoveLineWhenBlank (true | false) "false"
   isPrintInFirstWholeBand (true | false) "false"
   isPrintWhenDetailOverflows (true | false) "false"
   printWhenGroupChanges CDATA #IMPLIED
   forecolor CDATA #IMPLIED
   backcolor CDATA #IMPLIED
>
<!ELEMENT printWhenExpression (#PCDATA)>
```

4.8.1.1. Posición Absoluta

La atributos "*x*" e "*y*" de cualquier elemento del reporte son obligatorios y representan las coordenadas "*x*" e "*y*" medidas en píxeles, que indica la posición absoluta de la esquina superior izquierda del elemento especificado en la sección del reporte.

4.8.1.2. Posición Relativa

Algunos de los elementos del reporte, como los campos de texto tienen propiedades especiales que les permiten estirarse hacia abajo con el fin de obtener toda la información que tienen para mostrar. Su altura se calcula en tiempo de ejecución y puede afectar a los demás elementos vecinos presentes en la misma sección del reporte, especialmente los situados inmediatamente por debajo de ellos.

El atributo *positionType* específica el comportamiento que el elemento del reporte debe tener si el diseño de la sección del reporte en el que se da lugar se ve afectada por el estiramiento.

Hay tres posibles valores para el atributo *positionType*:

- **Posición flotante.-** El elemento se encuentra flotante en la sección padre, si se empuja hacia abajo por otros elementos por encima de ella. Se tratará de conservar la distancia entre él y los elementos vecinos situados por encima (*positionType = "float"*).
- **Posición fija con respecto a la parte superior de la banda padre.-** El elemento actual del reporte simplemente ignora lo que ocurre con los otros elementos de la sección y trata de conservar la posición y tamaño a partir de la parte superior de la sección padre del reporte (*positionType = "FixRelativeToTop"*).
- **Posición fija con respecto a la parte inferior de la banda padre.-** Si la altura de la sección padre de los reportes se ve afectada por los elementos que se extienden, el elemento actual trata de conservar la distancia original entre el margen inferior y la parte inferior de la banda padre (*positionType = "FixRelativeToBottom"*).

Por defecto, todos los elementos tienen una posición fija con respecto a la parte superior de la banda.

4.8.1.3. Elemento tamaño

Los atributos *width* y *height* son obligatorios y representan el tamaño del elemento de reporte medido en píxeles. Elementos adicionales que establecen el mecanismo de estiramiento del elemento determina si el motor generador de reportes deberá ignorar el elemento altura especificado. Sin embargo, este atributo es obligatorio, ya que incluso cuando la altura se calcule de forma dinámica, el elemento no debe ser menor a la altura original especificada.

4.8.1.4. Elemento color

Hay dos atributos que representan los colores: *forecolor* y *backcolor*. El color de primer plano es el que se utiliza para dibujar el texto de los elementos de texto y el borde de los elementos

gráficos. El color de fondo es el que se utiliza para rellenar el fondo del elemento de informe especificado, sino este no es transparente.

Usted puede especificar los colores mediante la representación decimal o hexadecimal del número entero que corresponde al color deseado. La forma preferida para especificar colores en XML es usando la representación hexadecimal, porque esta permite controlar el nivel de cada color base del sistema RGB.

Por ejemplo, se puede mostrar un texto en rojo si se establece el atributo *forecolor* del campo de texto correspondiente, de esta manera:

```
Forecolor = "#FF0000"
```

El equivalente usando la representación decimal sería:

```
Forecolor = "16711680"
```

Pero el inconveniente es evidente.

El color por defecto de *forecolor* es el negro y el color por defecto de *backcolor* es blanco.

4.8.1.5. Elemento transparencia

Los elementos del reporte pueden ser transparentes u opacos, dependiendo del valor que especifique para el atributo *mode*.

El valor predeterminado para este atributo depende del tipo de elemento del reporte. Los elementos gráficos como rectángulos y líneas son opacos por defecto, pero las imágenes son transparentes. Los textos estáticos y campos de texto son transparentes por defecto, al igual que los elementos del subreporte.

4.8.1.6. Saltando la visualización de elementos

El motor generador de reportes en tiempo de ejecución puede decidir si realmente debe mostrar un elemento del reporte, si se utiliza la expresión *<printWhenExpression>* que está disponible para todo tipo de elementos del reporte. Si esta expresión de reporte está presente deberá retornar un objeto *java.lang.Boolean* o *null* y es evaluada cada vez que la sección que contiene el elemento es generada, para ver si este elemento en particular debe aparecer o no en el reporte.

Si la expresión retorna un valor *null*, este valor equivale a retornar *java.lang.Boolean.FALSE* y si la expresión está presente, el elemento del reporte se imprimirá cada vez, que otro ajuste no interviene.

4.8.1.7. Reimprimiendo elementos

Cuando se genera una sección del reporte, el generador puede verse obligado a empezar una nueva página o columna, porque el espacio que queda en la parte inferior de la página o

columna actual no es suficiente para que todos los elementos de la sección quepan, probablemente debido a que algunos elementos se han estirado.

En estos casos, es posible volver a imprimir algunos de los elementos que ya hayamos mostrado en una nueva página o columna, para recrear el contexto en el que el salto de página o columna se produjo. Para lograr esto, se tiene que establecer *isPrintWhenDetailOverflows = "true"* para todos los elementos del reporte que se desea que se vuelva a mostrar en la siguiente página o columna.

4.8.1.8. Suprimiendo valores repetidos

En primer lugar, vamos a ver qué es exactamente un "valor repetido". Depende mucho del tipo de elemento de reporte que estamos hablando.

Para los elementos de campo de texto, esto es muy intuitivo. La siguiente lista contiene nombres de personas tomadas de una guía telefónica, se puede ver que para algunas líneas consecutivas, el valor de la columna "Familia" está repetido.

FAMILIA	NOMBRE	TELEFONO
Johnson	Adam	2 561 235
Johnson	Christine	2 995 452
Johnson	Peter	2 989 747
Johnson	Richard	2 973 665
Smith	Jhon	2 978 876
Smith	Laura	2 987 753
Smith	Denise	2 932 872

Fuente: [L01]

Tabla 4.2. Listado de nombres de guía telefónica.

Es posible que se desee suprimir los valores repetidos de la columna "Familia" e imprimirlos de la siguiente manera:

FAMILIA	NOMBRE	TELEFONO
Johnson	Adam	2 561 235
	Christine	2 995 452
	Peter	2 989 747
	Richard	2 973 665
Smith	Jhon	2 978 876
	Laura	2 987 753
	Denise	2 932 872

Fuente: [L01]

Tabla 4.3. Listado de nombres sin repetir la familia.

[L01] The JasperReports Ultimate Guide, Teodor Danciu.

Se puede hacer esto, si en el campo de texto donde se despliega "Familia", se establece:

```
isPrintRepeatedValues = "false"
```

Los elementos de texto estático se comportan de la misma manera. Como era de esperar, su valor siempre se repite y de hecho nunca cambia, hasta el final del reporte. Esto es porque llamamos estos textos estáticos. Por lo tanto, si se establece *isPrintRepeatedValues = "false"* para uno de los elementos *<staticText>*, se debe esperar para ver que aparece una sola vez, la primera vez, al comienzo del informe, y nunca más.

Ahora, ¿qué pasa con los elementos gráficos?

Una imagen es considera para repetirse a sí misma si sus bytes son exactamente los mismos de una ocurrencia a la siguiente. Esto sólo podría suceder si usted elige para almacenar en caché las imágenes utilizando el atributo *isUsingCache* disponible en el elemento *<image>* y si el *<imageExpression>* correspondiente retorna el mismo valor de una iteración a la siguiente.

Las líneas y los rectángulos están siempre repitiéndose, porque son elementos estáticos, al igual que los elementos de texto estático. Así, al decidir no mostrar valores repetidos de una línea o un rectángulo, se debe esperar ver que aparezca una sola vez, al comienzo del reporte, y luego se ignora hasta el final del informe.

Si se decide no mostrar los valores de repetición de algunos elementos del reporte, se tiene la posibilidad de suavizar o mejorar este comportamiento, indicando las ocasiones excepcionales en las que es posible que se quiera tener un valor particular que se vuelva a mostrar, durante el proceso de generación de reportes.

Cuando el valor de la repetición se extiende en varias páginas o columnas, se tiene la posibilidad de volver a mostrar este valor repetido al menos una vez por cada página o columna. Mediante el establecimiento de *isPrintInFirstWholeBand = "true"*, se asegura que el elemento del reporte vuelva a aparecer en la primera banda de una nueva página o columna que no es un desbordamiento de la página o columna anterior.

Además, si los valores repetidos suprimidos se extienden en varios grupos, se tiene la posibilidad de hacer reaparecer a principios de un grupo de reportes, si se especifica el nombre de ese grupo en particular en el atributo *printWhenGroupChanges*.

4.8.1.9. Eliminando espacios en blanco

Cuando los elementos del reporte no se muestran, por alguna razón: *<printWhenExpression>* evaluados para *Boolean.FALSE*, o el valor repetido es suprimido, un espacio en blanco se muestra en el elemento del reporte.

Este espacio en blanco también aparece si un campo de texto muestra solo caracteres en blanco o un texto vacío.

Hay una manera de eliminar estos espacios en blanco no deseados, en el eje vertical, sólo si ciertas condiciones se cumplen. Por ejemplo, si usted tiene tres campos de texto sucesivos, uno encima del otro de esta manera:

```
TextField1
TextField2
TextField3
```

Si el segundo tiene una cadena vacía como su valor, o contiene un valor repetido que ha elegido para suprimir, la salida sería así:

```
TextField1

TextField3
```

Con el fin de eliminar la brecha entre el primer campo de texto y el tercero, se tiene que configurar *isRemoveLineWhenBlank = "true"* para el segundo campo de texto. Se podría obtener algo como esto:

```
TextField1
TextField3
```

Sin embargo, hay ciertas condiciones que deben cumplirse para que esta funcionalidad pueda trabajar. El espacio en blanco no será removido, si el segundo campo de texto comparte un poco de espacio vertical con otros elementos del reporte que son impresos, incluso este segundo campo de texto no se imprime.

Por ejemplo, podría tener algunas líneas verticales en los lados de la sección de su informe como este:

```
|   TextField1   |
|                |
|   TextField3   |
```

Fuente:[L01]

Figura 4.7. Campos de texto acompañados de líneas verticales.

También se podría tener un rectángulo que dibuja una caja alrededor de los campos de texto:

```
------------------
|   TextField1   |
|                |
|   TextField3   |
------------------
```

Fuente:[L01]

Figura 4.8. Campos de texto con líneas verticales y horizontales.

[L01] The JasperReports Ultimate Guide, Teodor Danciu.

O incluso otros elementos de textos que se colocan en la misma horizontal con su segundo campo de texto:

```
Label1    TextField1
Label2
Label3    TextField3
```

Fuente:[L01]

Figura 4.9. Campos de texto con elementos de texto.

En todas estas situaciones, el espacio en blanco entre el primer y el tercer campo de texto no puede ser eliminado, porque este es usado por otro elemento del reporte que está impreso como se puede ver.

El espacio en blanco vertical entre los elementos puede ser eliminado usando el atributo *isRemoveWhenBlank*, sólo si no se utiliza por otros elementos, como se explicó anteriormente.

4.8.2. Elementos de texto

Hay dos tipos de elementos de texto en JasperReports: textos estáticos y campos de texto. Como sus nombres lo indican, el primero son elementos fijos con contenido estático, porque no cambian durante el proceso de llenado del reporte y se utilizan sobre todo para la introducción de títulos en el documento final. Los campos de texto sin embargo, tienen una expresión asociada, que se evalúa en tiempo de ejecución para producir el contenido del texto que se mostrará.

Ambos tipos de elementos de texto comparten algunas propiedades y los que son introducidos mediante un elemento <*textElement*>.

```
<!ELEMENT textElement (font?)>
<!ATTLIST textElement
       textAlignment (Left | Center | Right | Justified) "Left"
       lineSpacing (Single | 1_1_2 | Double) "Single"
>
```

4.8.2.1. Alineación del texto

Se puede especificar como el contenido de un elemento de texto debe estar alineado usando el atributo *textAlignment* y se puede elegir de entre cuatro valores posibles: "*Left*", "*Center*", "*Right*" o "*Justified*".

4.8.2.2. Interlineado del texto

La cantidad de espacio entre líneas consecutivas de texto se puede ajustar mediante el atributo *lineSpacing*:

- **Single.-** El interlineado del párrafo es normal con un desplazamiento igual a la altura de la línea de texto (*lineSpacing="Single"*).

[L01] The JasperReports Ultimate Guide, Teodor Danciu.

- **1,5 Líneas.-** El desplazamiento entre dos líneas de texto consecutivas es de 1 ½ líneas (*lineSpacing = "1_1_2"*).
- **Doble.-** El espacio entre líneas de texto es el doble de la altura de una sola línea de texto (*lineSpacing = "Double"*).

La configuración de fuente de los elementos de texto también es parte de la etiqueta *<textElement>*.

4.8.3. Texto Estático

Los textos estáticos son elementos de texto con contenido fijo, que no cambia durante el proceso de llenado del reporte. Son usados por lo general para introducir rótulos de texto estático en los documentos generados.

```
<!ELEMENT staticText (reportElement, textElement?, text?)>
<!ELEMENT text (#PCDATA)>
```

Como se puede apreciar en la sintaxis presentada, además de las propiedades generales de los elementos y las propiedades del texto específico que ya hemos explicado, existe además una definición de texto estático en la etiqueta *<text>*, que introduce el contenido fijo de texto del elemento de texto estático.

4.8.4. Campos de texto

A diferencia de los elementos de texto estático, que no cambian su contenido, los campos de texto tienen asociada una expresión que se evalúa en cada iteración con el origen de datos, a fin de obtener el contenido del texto que tiene que ser mostrado.

```
<!ELEMENT textField (reportElement, textElement?, textFieldExpression?,
anchorNameExpression?, hyperlinkReferenceExpression?,
hyperlinkAnchorExpression?, hyperlinkPageExpression?)>
<!ATTLIST textField
       isStretchWithOverflow (true | false) "false"
       evaluationTime (Now | Report | Page | Column | Group) "Now"
       evaluationGroup CDATA #IMPLIED
       pattern CDATA #IMPLIED
       isBlankWhenNull (true | false) "false"
       hyperlinkType (None | Reference | LocalAnchor | LocalPage |
       RemoteAnchor | RemotePage) "None"
>
<!ELEMENT textFieldExpression (#PCDATA)>
<!ATTLIST textFieldExpression
class (java.lang.Boolean | java.lang.Byte | java.util.Date |
java.sql.Timestamp | java.lang.Double | java.lang.Float | java.lang.Integer
| java.lang.Long | java.lang.Short | java.math.BigDecimal |
java.lang.String) "java.lang.String"
>
```

4.8.4.1. Variable altura

Teniendo en cuenta el hecho de que los campos de texto tienen un contenido dinámico, la mayoría de las veces no será capaz de anticipar con exactitud la cantidad de espacios que se debe establecer a los campos de texto para que puedan mostrar todo su contenido. Si el espacio que reserva para los campos de texto no es suficiente, el contenido del texto se trunca para que quepa en la superficie disponible.

Este escenario no siempre es aceptable y puede hacer que el generador de reportes calcule por sí mismo en tiempo de ejecución la cantidad de espacios necesario para mostrar el contenido completo del campo de texto y ajustar automáticamente el tamaño del elemento de reporte.

Esto se puede lograr establecimiento el atributo *isStretchWithOverflow* a "true" para los elementos de campos de texto en los que se desee establecer esta propiedad. De esta manera, se asegura que si la altura especificada por el campo de texto no es suficiente, automáticamente se incrementará (nunca disminuirá) con el fin de mostrar el contenido de texto completo.

Cuando los campos de texto se ven afectados por este mecanismo de estiramiento, la sección completa del reporte al que pertenecen también será extendida.

4.8.4.2. Evaluando Campos de Texto

Normalmente, todas las expresiones del reporte se evalúan inmediatamente, utilizando los valores actuales de todos los parámetros, campos y variables en ese momento en particular. Es como hacer una foto con todos los datos, por cada iteración con el origen de datos, durante el proceso de llenado del reporte.

Esto significa que en un momento determinado, se tiene acceso a los valores que se van a calcular más tarde, durante el proceso de llenado del reporte. Esto es perfectamente lógico, ya que todas las variables se calculan paso a paso y alcanzar su valor final sólo cuando la interacción llega al final del rango del origen de datos.

Por ejemplo, una variable de reporte que calcula la suma de un campo para cada página no contendrá la suma esperada hasta que el final de la página. Esto es porque la suma se calcula paso a paso, durante la interacción con los registros del origen de datos, y en determinado momento, la suma será sólo parcial, ya que no todos los registros del rango establecido han sido procesados. Si este es el caso, la forma de mostrar la suma de este campo en la página, es en el encabezado de la página, ya que este valor sólo se conocerá cuando el final de la página se alcance. Al principio de la página, cuando se genere el encabezado de la página, nuestra variable suma contendrá el valor de cero, o su valor inicial.

Afortunadamente, JasperReports tiene una característica muy interesante que permite decidir el momento exacto en que se desea que la expresión del campo de texto sea evaluada, para evitar el comportamiento predeterminado que hace que una expresión sea evaluada de inmediato, cuando se procesa la sección actual del reporte. Es el atributo *evaluationTime* del que estamos hablando y este puede tener uno de los siguientes valores:

- **Evaluación inmediata.-** La expresión de campo de texto es evaluada cuando se llena la banda a la que pertenece (*evaluationTime = "Now"*).
- **Evaluación al final del reporte.-** la expresión de campo de texto es evaluada al llegar al final del informe (*evaluationTime="Report"*).
- **Evaluación al final de la página.-** la expresión de campo de texto es evaluada al llegar al final de la página (*evaluationTime="Page"*).
- **Evaluación al final de la columna.-** la expresión del campo de texto es evaluada al llegar al final de la columna actual (*evaluationTime="Column"*).
- **Evaluación al final del grupo.-** la expresión del campo de texto es evaluada cuando el grupo especificado por el atributo *evaluationGroup* cambia (*evaluationTime="Group"*).

El valor predeterminado para este atributo es "*Now*". En el ejemplo presentado anteriormente, se puede especificar *evaluationTime = "Page"* para el campo de texto ubicado en la sección de encabezado de página, de manera que se muestre el valor de la variable suma sólo al llegar al final de la página.

La única restricción que se debe tener en cuenta, cuando se decide evitar la evaluación inmediata de la expresión de un campo de texto, es que en estos casos el campo de texto no se extenderá a fin de visualizar todo su contenido.

Esto se da porque la altura del elemento de texto se calcula cuando la sección del reporte es generada y hasta que el generador de reportes vuelva con el texto contenido en el campo de texto, el atributo altura no se adapta, porque dañara el campo creado.

4.8.4.3. Suprimiendo valores *null*

Si la expresión de un campo de texto devuelve un valor *null*, el campo de texto muestra el "null" en el documento generado. Una manera sencilla de evitar esto es estableciendo el atributo *isBlankWhenNull* a "*true*". De esta manera, el campo de texto dejará de mostrar "null" y mostrará una cadena vacía. De esta manera no va a aparecer en el documento si el valor del campo de texto es *null*.

4.8.4.4. Dando formato

Por supuesto, cuando se trata de valores numéricos o de fecha/hora, se puede utilizar la API de Java para formatear la salida de las expresiones de los campos de texto. Pero hay una manera más conveniente de hacerlo mediante el atributo disponible en el elemento *<textField>*.

El valor que se debe suministrar a este atributo, es el mismo que se suministra si se fuera a dar formato al valor mediante la clase *java.text.DecimalFormat* o la clase *java.text.SimpleDate Format*, dependiendo del tipo de valor a dar formato.

De hecho, lo que el generador de reportes hace es crear una instancia de la clase *java.text.DecimalFormat* si la expresión del campo de texto retorna una subclase de la clase *java.lang.Number* o crear una instancia de la clase *java.text.SimpleDataFormat* si la expresión del campo de texto retorna un objeto *java.util.Date* o un objeto *java.sql.Timestamp*.

4.8.4.5. Expresiones de campos de texto

Ya hemos hablado acerca de las expresiones de los campos de texto. No hay nada más que decir al respecto, excepto que este se introduce utilizando el elemento *<textFieldExpression>* y puede retornar valores de solo un rango limitado de clases listadas a continuación:

- java.lang.Boolean
- java.lang.Byte
- java.util.Date
- java.sql.Timestamp
- java.lang.Double
- java.lang.Float
- java.lang.Integer
- java.lang.Long
- java.lang.Short
- java.math.BigDecimal
- java.lang.String

Si la clase de la expresión del campo de texto no es especificada usando el atributo *class*, se asume por defecto la clase *java.lang.String*.

4.8.5. Fuentes y soporte Unicode

Cada elemento de texto presente en el reporte puede tener su configuración de fuente propia. Estos valores se pueden especificar usando la etiqueta *<font>* disponible en la etiqueta *<textElement>*.

Dado que la mayoría de las veces en un diseño, sólo hay unos pocos tipos de fuentes utilizadas, que son compartidas por diferentes elementos de texto, no tiene sentido obligar a los creadores de diseño XML especificar la misma fuente en cada elemento de texto, una y otra vez. Sino que podría hacerse referencia a una fuente en la declaración de un nivel de reporte y configurar algunos ajustes de la fuente, si un elemento de texto en particular lo requiere.

4.8.5.1. Fuente del reporte

Una fuente del reporte es en realidad una colección de configuraciones de la fuente declarada en el nivel de reporte que puede ser reutilizada en el diseño completo, al establecer las propiedades de fuente para los elementos de texto.

```
<!ELEMENT reportFont EMPTY>
<!ATTLIST reportFont
        name NMTOKEN #REQUIRED
        isDefault (true | false) "false"
        fontName CDATA "sansserif"
        size NMTOKEN "10"
        isBold (true | false) "false"
        isItalic (true | false) "false"
        isUnderline (true | false) "false"
        isStrikeThrough (true | false) "false"
        pdfFontName CDATA "Helvetica"
        pdfEncoding CDATA "CP1252"
        isPdfEmbedded (true | false) "false"
>
```

4.8.5.2. Nombre de la fuente del reporte

El atributo *name* de un elemento *<reportFont>* es obligatorio y debe ser único, ya que será utilizado para hacer referencia a la correspondiente fuente del reporte a través del reporte.

4.8.5.3. Fuente predeterminada del reporte

Se puede utilizar *isDefault = "true"* para declarar la fuente predeterminada del reporte, para declarar la fuente del reporte que desea utilizar para que el motor generador de reportes como la fuente base por defecto, cuando se trata de elementos de texto que no hacen referencia a una fuente del reporte en particular.Esta fuente por defecto también será utilizada por los elementos de texto que no tiene ninguna configuración de fuente en absoluto.

Todas las otras propiedades de fuente del informe son los mismos que los de un elemento *<font>* normal que vamos a ver a continuación.

```
<!ELEMENT font EMPTY>
<!ATTLIST font
        reportFont NMTOKEN #IMPLIED
        fontName CDATA #IMPLIED
        size NMTOKEN #IMPLIED
        isBold (true | false) #IMPLIED
        isItalic (true | false) #IMPLIED
        isUnderline (true | false) #IMPLIED
        isStrikeThrough (true | false) #IMPLIED
        pdfFontName CDATA #IMPLIED
        pdfEncoding CDATA #IMPLIED
        isPdfEmbedded (true | false) #IMPLIED
>
```

4.8.5.4. Referenciando una fuente del reporte

Al introducir la configuración de fuente de un elemento de texto del reporte, se tiene la posibilidad de utilizar una declaración de fuente del reporte como base para las configuraciones que se desea obtener. Todos los atributos del elemento *<font>*, si está presente, se utilizan para anular los atributos con los mismos nombres que están presentes en la declaración de fuente del reporte, referenciada mediante el atributo *reportFont*.

Por ejemplo, si tenemos una fuente del informe como el siguiente:

```
<reportFont
        name="Arial_Normal"
        isDefault="true"
        fontName="Arial"
        size="8"
        pdfFontName="Helvetica"
        pdfEncoding="Cp1252"
        isPdfEmbedded="false"
/>
```

y se desea crear un campo de texto que tenga básicamente la misma configuración de fuente que tiene el reporte, pero con un tamaño mayor; lo único que se debe hacer es referenciar la fuente del reporte con el atributo *reportFont* y especificar el tamaño de letra deseado:

```
<textElement>
        <font reportFont="Arial_Normal" size="14"/>
</ TextElement>
```

Cuando el atributo *reportFont* está ausente, la fuente por defecto del reporte es usada como fuente base.

4.8.5.5. Nombre de la fuente

En Java, hay dos tipos de fuentes: fuentes físicas y fuentes lógicas. Fuentes físicas son las fuentes provenientes de una biblioteca real, como por ejemplo, *TrueType* o *PostScript Type*. Las fuentes físicas pueden ser físicas *Arial*, *Time*, *Helvetica*, *Courier*, o cualquier otra fuente, incluidas las bibliotecas de fuentes internacionales.

Las fuentes lógicas son cinco tipos de fuente que han sido reconocidas por la plataforma Java desde la versión 1.0: *Serif*, *Sans-serif*, *Monospaced*, Dialog y *DialogInput*. Las fuentes lógicas no están en las bibliotecas de fuentes reales que son instaladas en el sistema, estas son simplemente nombres de tipos de fuente reconocidos por el runtime de Java, que debe ser asignado a alguna fuente física instalada en el sistema.

En el atributo *fontName* del elemento *<font>* o del elemento *<reportFont>*, se tiene que especificar el nombre de una fuente física o el nombre de una fuente lógica. Solamente se debe asegurar que la fuente especificada realmente existe y está disponible en el sistema.

4.8.5.6. Tamaño de fuente

El tamaño de la fuente se mide en puntos y puede ser especificada usando el atributo *size*.

4.8.5.7. Estilos de fuente

Hay cuatro atributos booleanos disponibles en el *<font>* y *<reportFont>,* elementos que controlan el estilo de la fuente. Estos son *isBold*, *isItalic*, *isUnderline* y *isStrikeThrough* y su significado es evidente para cualquier persona.

4.8.5.8. Nombre de la fuente PDF

Cuando exportamos los reportes al formato PDF, JasperReports usa la biblioteca iText. Como su nombre lo indica (*Portable Document Format*) los archivos PDF se pueden ver en varias plataformas y puede estar seguro que siempre tendrá el mismo aspecto. Esto es parte porque existe una manera especial de tratar con las fuentes.

Si se desea diseñar reportes que eventualmente puedan exportarse a PDF, se tiene que establecer la configuración de fuente PDF apropiada que corresponden a la configuración de fuente Java de los elementos de texto.

La biblioteca iText sabe cómo tratar con colección de fuentes y archivos TTF. Esta reconoce los siguientes nombres fuentes:

- Courier
- Courier-Bold
- Courier-BoldOblique
- Courier-Oblique
- Helvetica
- Helvetica-Bold
- Helvetica-BoldOblique
- Helvetica-Oblique
- Symbol
- Times-Roman
- Times-Bold
- Times-BoldItalic
- Times-Italic
- ZapfDingbats

La biblioteca iText requiere que especifiquemos un nombre de fuente de la lista anterior, o el nombre de un archivo TTF que se pueda localizar en el disco, cada vez que se trabaje con las fuentes. El nombre de la fuente introducido por el atributo *fontName* se explicó anteriormente que no sirve de nada cuando se exporta a PDF. Esto se debe porque se cuenta con atributos de fuente especiales, que podemos especificar en la configuración de fuentes de la biblioteca iText.

El atributo *pdfFontName* puede contener el nombre de una fuente PDF incorporada de la lista anterior o el nombre de un archivo TTF que se puede localizar en el disco durante la exportación del reporte a formato PDF.

El creador del diseño de reportes debe elegir el valor correcto para el atributo *pdfFontName* que corresponda a la fuente física o lógica específica de Java, usando el atributo *fontName*. Si estas dos fuentes, una utilizada por los visualizadores de Java y las impresoras y la otra utilizada en el formato PDF, no representan en realidad el mismo tipo de letra, o al menos es posible que se obtenga resultados inesperados al momento de exportar al formato PDF.

Fuentes PDF adicionales se puede instalar en el sistema si se cambia uno de los paquetes de tipos de fuentes de Acrobat Reader's. Por ejemplo, al instalar el paquete de fuentes asiáticas de Adobe en el sistema, se podría establecer para el atributo *pdfFontName* los nombres de fuentes como:

LENGUAJE	FUENTE PDF
Simplified Chinese	STSong-Light
Traditional Chinese	MHei-Medium
	MSung-Light
Japanese	HeiseiKakuGo-W5
	HeiseiMin-W3
Korean	HYGoThic-Medium
	HYSMyeongJo-Medium

Fuente:[L01]

Tabla 4.4. Tipos de fuentes PDF.

4.8.5.9. Codificación PDF

Cuando diseñamos reportes en diferentes lenguajes y se desea exportarlos a PDF, se tiene que establecer el tipo de codificación de caracteres adecuada para el reporte.

Por ejemplo, un tipo de codificación utilizado en Europa es *Cp1252*, también conocida como *LATIN1*. Otros posibles tipos de codificación son los siguientes:

SET DE CARACTERES	CODIFICACION
Latin 2: Eastern Europe	Cp1250
Cyrillic	Cp1251
Greek	Cp1253
Turkish	Cp1254
Windows Baltic	Cp1257
Simplified Chinese	UniGB-UCS2-H
	UniGB-UCS2-V
Traditional Chinese	UniCNS-UCS2-H
	UniCNS-UCS2-V
Japanese	UniJIS-UCS2-H
	UniJIS-UCS2-V
	UniJIS-UCS2-HW-H
	UniJIS-UCS2-HW-V
Korean	UniKS-UCS2-H
	UniKS-UCS2-V

Fuente:[L01]

Tabla 4.5. Set de caracteres con sus codificaciones.

4.8.5.10. Fuentes PDF Incorporadas

Si desea utilizar un archivo TTF al momento de exportar los reportes a formato PDF y se quiere asegurar que todo el mundo sea capaz de ver el reporte sin ningún problema, se tiene que asegurar que al menos una de las siguientes condiciones se cumpla:

[L01] The JasperReports Ultimate Guide, Teodor Danciu.

- Todos tienen ese tipo de letra TTF instalado en sus sistemas.
- Incorporar el tipo de fuente en el documento PDF.

No es fácil cumplir con la primera condición y es por eso que la mejor manera de hacerlo, es incorporando el TTF en los documentos PDF generados y que son distribuidos.

Esto se puede realizar estableciendo el atributo *isPdfEmbedded* a "true".

4.8.6. Elementos gráficos

La segunda gran categoría de los elementos del reporte, además de elementos de texto que hemos visto, son los elementos gráficos. En esta categoría tenemos líneas, rectángulos e imágenes.

Todos estos elementos tienen algunas propiedades en común y los que se agrupan bajo los atributos de la etiqueta *<graphicElement>*.

```
<!ELEMENT graphicElement EMPTY>
<!ATTLIST graphicElement
       stretchType (NoStretch | RelativeToTallestObject |
       RelativeToBandHeight) "NoStretch"
       pen (None | Thin | 1Point | 2Point | 4Point | Dotted) #IMPLIED
       fill (Solid) "Solid"
>
```

4.8.6.1. Comportamiento

El atributo *stretchType* de un elemento gráfico se puede utilizar para personalizar el comportamiento del elemento cuando en la misma sección del reporte hay campos de texto que se extienden porque su contenido de texto es demasiado grande para caber en la altura original del campo de texto. Cuando campos de texto extensibles están presentes en una sección del reporte, la altura de la sección del reporte se ve afectada por el estiramiento de sus elementos.

Un elemento gráfico puede responder a la modificación del estiramiento de una sección del reporte de tres maneras:

- **No se estira.-** el elemento gráfico conserva la altura original especificada (*strechType = "NoStretch"*).
- **Estiramiento en relación a la altura de la banda.-** el elemento gráfico adaptará su altura de acuerdo a la nueva altura de la sección del reporte donde se encuentra y se ha visto afectada por el estiramiento (*stretchType = "RelativeToBandHeight"*).
- **Estiramiento en relación al elemento más alto del grupo:** Se tiene la posibilidad de agrupar los elementos de una sección del reporte en múltiples grupos, si se desea. La única razón que puede tener la agrupación de los elementos del reporte es que se puede personalizar su comportamiento cuando la sección del reporte se extiende. Los elementos pueden adaptar automáticamente su altura para acoplarse a la cantidad de

estiramiento sufrido por el elemento más alto del grupo al que pertenece. (*stretchType = "RelativeToTallestObject"*).

4.8.6.2. Espesor del borde

A diferencia de los elementos de texto, los elementos gráficos siempre tienen un borde. Pudiendo controlarse el tipo y espesor del mismo usando el atributo *pen*. Recuerde que el color del borde se establece el atributo *forecolor* presente cuando se describe la etiqueta *<reportElement>*.

Aquí se describen todos los tipos de bordes posibles para un elemento gráfico:

- **Sin borde.-** el elemento gráfico no despliega ningún borde a su alrededor (*pen="None"*).
- **Borde fino.-** el grueso del borde que rodea al elemento gráfico será de la mitad de un punto (*pen="Thin"*).
- **Borde de un punto.-** Borde normal (*pen="1Point"*).
- **Borde de dos puntos.-** Borde grueso (*pen="2Point"*).
- **Borde de cuatro puntos.-** Borde muy grueso (*pen="4Point"*).
- **Borde de puntos.-** el grosor del borde será de un punto y será de puntos continuos (*pen="Dotted"*).

El borde predeterminado para los elementos gráficos depende del tipo. Las líneas y los rectángulos tienen predeterminado un borde normal de un punto de espesor. Las imágenes sin embargo, no muestran ninguna borde, de manera predeterminada.

4.8.6.3. Estilo del rellenado

El atributo *fill* especifica el estilo del fondo de los elementos gráficos, pero el estilo sólo se admite cuando el estilo de relleno es sólido, que es la opción por defecto de los elementos gráficos (*fill = "Solid"*).

4.8.6.4. Líneas

Cuando se muestra un elemento línea, JasperReports dibuja una de las dos diagonales del rectángulo representado por los atributos *x*, *y*, *width* y *height* especificados para este elemento de línea en particular.

```
<!ELEMENT line (reportElement, graphicElement?)>
<!ATTLIST line
       direction (TopDown | BottomUp) "TopDown"
>
```

Dirección de la línea

Cuando una de las dos diagonales del rectángulo es dibujada puede establecerse la dirección de la misma utilizando el atributo *direction*:

- La diagonal que inicia en la esquina superior izquierda del rectángulo y termina en la esquina inferior izquierda se dibuja en caso de establecer *direction="TopDown"*.
- La línea que inicia en la esquina inferior izquierda y termina en la parte superior derecha se dibuja si se establece *direction="BottomUp"*.

Se puede dibujar líneas verticales especificando *width="0"* y líneas horizontales estableciendo *height = "0"*. Para estas líneas la dirección no es importante.

La dirección por defecto para una línea es *"TopDown".*

4.8.6.5. Rectángulos

Los rectángulos son los elementos gráficos básicos. Por esta razón no hay ninguna configuración adicional que hacer para la declaración de un elemento rectángulo, además de las que ya hemos visto en las etiquetas *<reportElement>* y *<graphicElement>*.

```
<!ELEMENT rectangle (reportElement, graphicElement?)>
```

4.8.6.6. Imágenes

Los elementos gráficos más complejos que puede tener un reporte son las imágenes.

Así como para los elementos de campo de texto, su contenido se evalúa dinámicamente durante el tiempo de ejecución, usando una expresión del reporte.

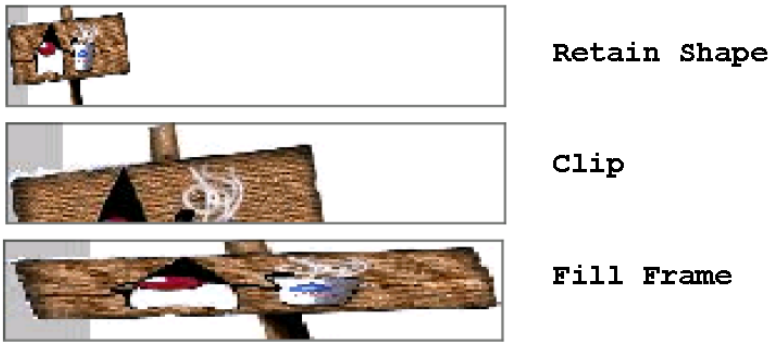
```
<!ELEMENT image (reportElement, graphicElement?, imageExpression?,
anchorNameExpression?, hyperlinkReferenceExpression?,
hyperlinkAnchorExpression?, hyperlinkPageExpression?)>
<!ATTLIST image
        scaleImage (Clip | FillFrame | RetainShape) "RetainShape"
        isUsingCache (true | false) "true"
        evaluationTime (Now | Report | Page | Column | Group) "Now"
        evaluationGroup CDATA #IMPLIED
        hyperlinkType (None | Reference | LocalAnchor | LocalPage |
        RemoteAnchor | RemotePage) "None"
>
<!ELEMENT imageExpression (#PCDATA)>
<!ATTLIST imageExpression
        class (java.lang.String | java.io.File | java.net.URL |
        java.io.InputStream | java.awt.Image) "java.lang.String"
>
```

Escala de imágenes

Teniendo en cuenta el hecho de que las imágenes se cargan en tiempo de ejecución, no hay manera de saber que su tamaño exacto, cuando creamos un diseño. Puede ser que las dimensiones del elemento imagen especificadas, durante el diseño no correspondan con las dimensiones de la imagen real cargada en tiempo de ejecución.

Es por esto que se debe decidir cómo se espera la imagen para cambiar su comportamiento con el fin de adaptarla a las dimensiones originales del elemento imagen que se especifica en el diseño del reporte. Existe el atributo *scaleImage* que permite hacer esto, estableciendo uno de los tres valores posibles:

- **Recortar la imagen.-** si el tamaño de la imagen real es mayor al tamaño del elemento imagen, esta será cortada de manera que quepa en la resolución original del elemento y sólo la región que se ajuste al tamaño especificado se mostrará (*scaleImage = "Clip"*).
- **Forzar el tamaño de la imagen.-** si las dimensiones de la imagen real no se ajustan a las especificadas para el elemento imagen que la muestra, las dimensiones de la imagen se deben forzar con el fin de adaptarla a las dimensiones establecidas en el elemento imagen de modo que quepa. Con esta opción la imagen necesariamente se deforma (*scaleImage = "FillFrame"*).
- **Preservando las proporciones de la imagen.-** si la imagen actual no encaja en el elemento imagen, se puede adaptar las dimensiones de la imagen de manera que no se deforme y manteniendo sus proporciones originales (*scaleImage = "RetainShape"*).



Fuente: [L01]

Figura 4.7. Tipos de escalamientos de imágenes.

Almacenamiento en caché de las imágenes

Todos los elementos imagen tienen contenido dinámico. No hay elementos especiales para presentar imágenes estáticas en los reportes, al igual que el elemento especial de texto estático. Sin embargo, la mayoría de veces, las imágenes en un reporte son estáticas y no necesariamente provienen de un origen de datos o de parámetros. En la mayoría de casos, se cargan desde archivos en disco y representan logotipos u otros recursos estáticos.

Si tenemos que mostrar la misma imagen varias veces en un informe, si se trata de un logotipo que aparece en el encabezado de la página, por ejemplo; no tiene ningún sentido cargar el archivo de imagen cada vez que tengamos que mostrarla. Podemos indicar al motor generador de reportes que almacene en caché esta imagen en particular. De esta manera nos estamos

[L01] The JasperReports Ultimate Guide, Teodor Danciu.

asegurando que la imagen se cargue desde disco o una ubicación en particular sólo una vez y luego será utilizada cada vez que tenga que mostrarse.

Al establecer el atributo *isUsingCache* a "*true*", el generador de reportes trata de identificar previamente la carga de imágenes utilizando el origen especificado. Por ejemplo, reconoce una imagen si en el origen de la imagen encuentra un archivo con el nombre de la imagen que tiene que cargar, o si es la misma URL.

Esta funcionalidad de almacenamiento en caché sólo está disponible para los elementos imagen que tienen expresiones que retornan objetos *java.lang.String* como el origen de la imagen, representada con un nombre de archivo, direcciones URL o classpath de recursos. Esto debido a que el generador de reporte usa la cadena de origen de la imagen como la clave para identificar que es la misma imagen que ya ha almacenado en caché.

Evaluando imágenes

Al igual que en los campos de texto, se tiene la posibilidad de posponer la evaluación de las expresión de un elemento imagen, que por defecto se realiza inmediatamente. Esto permitirá mostrar en alguna región del documento, las imágenes que se van a cargar o mostrar sólo después del proceso de llenado del reporte, debido a los complejos algoritmos o cualquier otra razón.

Los mismos atributos *evaluationTime* y *evaluationGroup*, que vimos en la sección de campos de texto están también disponibles para el elemento *<image>*. El atributo *evaluationTime* puede tener los siguientes valores:

- **Evaluación inmediata.-** la expresión de la imagen se evalúa cuando se llena la banda actual (*evaluationTime="Now"*).
- **Evaluación al final del reporte.-** la expresión de la imagen se evalúa al llegar al final del reporte (*evaluationTime="Report"*).
- **Evaluación al final de la página.-** la expresión de la imagen se evalúa al llegar al final de la página actual (*evaluationTime = "Page"*).
- **Evaluación al final de la columna.-** la expresión de la imagen se evalúa al llegar al final de la columna actual (*evaluationTime="Column"*).
- **Evaluación al final del grupo.-** la expresión de la imagen se evalúa cuando el grupo especificado por el atributo *evaluationGroup* cambia (*evaluationTime="Group"*).

El valor predeterminado para este atributo es "*Now*".

Expresión de la imagen

El valor retornado por la expresión de la imagen se utiliza como la fuente de la imagen que se va a mostrar. La expresión de la imagen es introducida por el elemento *<imageExpression>* y pueden retornar valores de un rango limitado de clases, que se listan a continuación:

- java.lang.String
- java.io.File
- java.net.URL
- java.io.InputStream
- java.awt.Image

Cuando la expresión de la imagen retorna un valor *java.lang.String*, el generador de reportes va a tratar de ver si el valor representa una URL para cargar la imagen. Si no es una representación de URL válida, tratará de localizar un archivo en disco y cargar la imagen, suponiendo que el valor representa un nombre de archivo. Si no encuentra el archivo, finalmente asume que el valor de la cadena representa la ubicación del recurso en el classpath y tratará de cargar la imagen desde ahí. Sólo si todo esto falla, se lanzará una excepción.

Si la clase de la expresión de la imagen no se especifica con el atributo *class*, se asume un *java.lang.String*, por defecto.

4.8.6.7. Cuadros y gráficos

La biblioteca JasperReports no produce cuadros y gráficos por sí misma. Esto no es uno de sus objetivos. Sin embargo, fácilmente puede integrar tablas y gráficos producidos por otras, bibliotecas Java más especializadas.

La gran mayoría de las bibliotecas Java disponibles que producen cuadros y gráficos producen archivos de imagen u objetos Java de imágenes en memoria. Es por esto que no resulta difícil colocar una tabla o un gráfico generado por una de estas bibliotecas en un documento JasperReports con un elemento imagen normal.

4.8.7. Hipervínculos

JasperReports permite crear reportes detallados, para introducir tablas de contenido en los documentos o para redireccionar a los espectadores a otros documentos externos usando el elemento especial de reporte llamado hipervínculo.

Los hipervínculos son elementos especiales que contienen una referencia a una ubicación local del documento actual o de un recurso externo al que será direccionado el espectador del documento si él hace clic en el elemento de enlace en particular.

Los hipervínculos no son los únicos actores en este escenario de redireccionamiento. Existe una manera para especificar cuáles son los destinos en un documento. Estos destinos locales se llaman anclas.

No hay elementos especiales del reporte que introduzcan hipervínculos o anclas en un diseño de reporte, sino más bien existe una configuración especial que hace que un elemento habitual del reporte sea comporte como hipervínculo o ancla.

En JasperReports, sólo los elementos de campo de texto e imagen pueden ser hipervínculos o anclas. Esto es porque en ambos tipos de elementos, hay opciones especiales que permiten especificar la referencia del hipervínculo pudiendo ser un enlace o un nombre de anclaje local.

```
<!ELEMENT anchorNameExpression (#PCDATA)>
<!ELEMENT hyperlinkReferenceExpression (#PCDATA)>
<!ELEMENT hyperlinkAnchorExpression (#PCDATA)>
<!ELEMENT hyperlinkPageExpression (#PCDATA)>
```

4.8.7.1. Tipos de hipervínculos

Cuando presentamos la sintaxis XML de los elementos de campo de texto y elementos de imagen, se pudo apreciar que existe un atributo llamado *hyperlinkType*. A continuación se detalla los posibles valores que este atributo puede tener, junto con su significado:

- **Sin hipervínculo.-** Por defecto, ni los campos de texto ni las imágenes representan hipervínculos, incluso si las expresiones especiales de hipervínculo está presente (*hyperlinkType ="None"*).
- **Referencia externa.-** El enlace actual señala un recurso externo especificado en el elemento correspondiente *<hyperlinkReferenceExpression>*, que por lo general es una URL (*hyperlinkType="Reference"*).
- **Vínculo local.-** El enlace actual señala a un vínculo local especificado en el elemento correspondiente *<hyperlinkAnchorExpression>* (*hyperlinkType = "LocalAnchor"*).
- **Página local.-** El enlace actual señala a un índice de la página basado en el documento actual especificado por el elemento correspondiente *<hyperlinkPageExpression>* (*hyperlinkType = "LocalPage"*).
- **Vínculo remoto.-** El enlace actual señala a un vínculo especificado por el elemento *<hyperlinkAnchorExpression>*, dentro del documento externo indicado en el elemento correspondiente *<hyperlinkReferenceExpression>* (*hyperlinkType = "RemoteAnchor"*).
- **Página remota.-** El enlace actual señala a un índice de la página especificado por el elemento *<hyperlinkPageExpression>*, dentro de un documento externo indicado por el elemento correspondiente *<hyperlinkReferenceExpression>* (*hyperlinkType = "RemotePage"*).

4.8.7.2. Expresión del vínculo

Si está presente en la declaración de un elemento campo de texto o imagen, la etiqueta *<anchorNameExpression>* va a transformar ese campo de texto o imagen en particular en un vínculo local del documento final, para que los hipervínculos puedan apuntarle. El vínculo llevará el nombre devuelto después de evaluar la expresión de nombre del vínculo, que siempre retorna un valor *java.lang.String*.

4.8.7.3. Expresiones del hipervínculo

Dependiendo del tipo de enlace actual, uno o dos de las siguientes expresiones serán evaluadas y utilizadas para construir la referencia a la que el elemento hipervínculo apuntará:

- *<hyperlinkReferenceExpression>*
- *<hyperlinkAnchorExpression>*

- *<hyperlinkPageExpression>*

Lo que es importante saber es que las dos primeras siempre deberán retornan un valor *java.lang.String* y la tercera deberá retornar un valor *java.lang.Integer*.

4.8.8. Grupos de elementos

Los elementos del reporte colocados en cualquier sección del reporte se pueden organizar en múltiples grupos. La única razón que se puede tener para agrupar los elementos es la habilidad para personalizar el comportamiento de los elementos gráficos.

El atributo *stretchType*, disponible para los elementos gráficos, tiene entre sus posibles valores uno llamado "*RelativeToTallestObject*". Al elegir esta opción, el generador de reportes tratará de identificar el objeto en el mismo grupo con el elemento gráfico actual, que sufrió la mayor cantidad de estiramiento. A continuación, se adaptará la altura del elemento gráfico actual a la altura de este elemento más alto del grupo.

Pero para que esto funcione, se tiene que agrupar los elementos. Esto se hace usando las etiquetas *<elementGroup>* y *</ elementGroup>* para marcar los elementos que forman parte de un mismo grupo.

```
<!ELEMENT elementGroup (line | rectangle | image | staticText | textField |
subreport | elementGroup)*>
```

Los grupos de elementos pueden contener otros grupos de elementos anidados y no hay límite en el número de grupos de elementos anidados.

4.9. SUBREPORTES

Los subreportes son una característica importante para una herramienta de generación de reportes. Estos permiten y facilitan la creación de reportes más complejos y simplifican el trabajo de diseño. Los subreportes son muy útiles cuando se crea un reporte tipo maestro-detalle o cuando la estructura de un reporte único no es suficiente para describir la complejidad del documento final.

Un subreporte es, de hecho, un reporte normal que se ha incorporado como parte de otro reporte. Se puede montar los subreportes y hacer que un subreporte contenga en sí otro subreporte, el nivel de anidamiento no está limitado.

Por otro lado, un subreporte es también un tipo especial de elemento del reporte que ayuda a introducir un subreporte en el reporte maestro o padre. No hay nada más que decir acerca de los subreportes, pues son vistos como reportes normales, debido a que se compilan y se llena al igual que los reportes normales. De hecho, cualquier diseño de reportes se puede utilizar como un subreporte cuando se lo incorpore en otro diseño de reportes, sin necesidad de cambiar nada en su interior.

Lo que vamos a ver ahora, son los detalles sobre el elemento *<subreport>* que se utiliza cuando introducimos subreportes en los reportes maestros.

```
<!ELEMENT subreport (reportElement, parametersMapExpression?,
subreportParameter*, (connectionExpression | dataSourceExpression)?,
subreportExpression?)>
<!ATTLIST subreport
       isUsingCache (true | false) "true"
>
<!ELEMENT parametersMapExpression (#PCDATA)>
<!ELEMENT subreportParameter (subreportParameterExpression?)>
<!ATTLIST subreportParameter
       name NMTOKEN #REQUIRED
>
<!ELEMENT subreportParameterExpression (#PCDATA)>
<!ELEMENT connectionExpression (#PCDATA)>
<!ELEMENT dataSourceExpression (#PCDATA)>
<!ELEMENT subreportExpression (#PCDATA)>
<!ATTLIST subreportExpression
class (java.lang.String | java.io.File | java.net.URL |
java.io.InputStream | dori.jasper.engine.JasperReport) "java.lang.String"
>
```

4.9.1. Expresión subreporte

Al igual que los diseños de reportes normales, los diseños de subreportes son de hecho objetos *dori.jasper.engine.JasperReport*. Estos se obtienen después de compilar un objeto *dori.jasper.engine.design.JasperDesign*.

Hemos visto que los elementos de campo de texto tienen una expresión que será evaluada para obtener el contenido del texto a mostrar. Los elementos imagen tienen una expresión que representa el origen de la imagen a mostrar. De la misma manera, los elementos subreporte tienen una expresión que es evaluada en tiempo de ejecución, para obtener el origen del objeto *dori.jasper.engine.JasperReport* a carga.

La expresión del subreporte es introducida por el elemento *<subreportExpression>* y puede retornar valores de las siguientes clases:

- java.lang.String
- java.io.File
- java.net.URL
- java.io.InputStream
- dori.jasper.engine.JasperReport

Cuando la expresión subreporte retorna un valor *java.lang.String*, el generador de reportes va a tratar de ver si el valor representa una URL desde donde se pueda cargar un objeto de diseño de subreporte. Si no es una URL válida, tratará de localizar un archivo en disco y cargará el diseño de subreporte desde ahí, asumiendo que el valor retornado representa un nombre de

archivo. Si no se encuentra el archivo, finalmente asume que el valor de cadena representa la ubicación de un recurso en el classpath y se intentará cargar el diseño de subreporte desde allí. Sólo si todos estos fallan, se lanzará una excepción.

Si la clase de la expresión del subreporte no se especifica usando el atributo *class,* este asume que es una *java.lang.String,* por defecto.

4.9.2. Almacenamiento en cache de subreportes

Un elemento subreporte puede cargar diferentes diseños de subreportes en cada evaluación, dándole una gran flexibilidad en la conformación de documentos. Sin embargo, la mayoría de veces, los elementos subreporte de un informe son de hecho estáticos y su origen no necesariamente cambia con cada nueva evaluación de la expresión subreporte. En la mayoría de casos, los diseños de subreportes se cargan de una ubicación fija: archivos en disco o URLs estáticas.

Si el mismo diseño de subreporte es llenado varias veces en un informe, no tiene ningún sentido cargar el objeto de diseño de subreporte desde el archivo de origen cada vez que tengamos que llenarlo con datos. Podemos indicar al generador de reportes que almacene en la caché este objeto de diseño de subreporte en particular. De esta manera nos aseguramos que el diseño de subreporte será cargado desde el disco o desde su particular ubicación sólo una vez y luego sólo se volverá a utilizar en el momento de llenado de este.

Al establecer el atributo *isUsingCache* a "*true*", el generador de reportes tratará de reconocer previamente a la carga los objetos de diseños de subreportes, utilizando la fuente especificada.

Esta funcionalidad de almacenamiento en caché sólo está disponible para los elementos subreporte que tienen expresiones que retornan objetos *java.lang.String*, como el origen de un diseño de subreporte, representado en nombres de archivo, direcciones URL's o recursos del classpath. Esto es porque el motor utiliza la cadena de origen de subreporte como la clave para reconocer que es el mismo diseño de subreporte que está almacenado en caché.

4.9.3. Parámetros

Puesto que los subreportes son reportes normales, que son compilados o llenados de la misma manera; esto significa que también requieren de una fuente de datos desde donde se pueda obtener los datos cuando estos sean llenados y que también pueden recibir parámetros, para recibir información adicional necesaria para cuando se inicie el proceso de llenado del informe.

Hay dos maneras de proporcionar valores a los parámetros de un subreporte y se pueden utilizar de forma simultánea, si se desea.

Se puede proporcionar un "map" que contenga los valores de los parámetros, como se lo hace cuando llenamos un informe normal con datos, utilizando uno de los métodos *fillReportXXX()* expuestos por la clase *dori.jasper.engine.JasperFillManager*. Esto se puede lograr si se utiliza el

elemento *<parametersMapExpression>*, que introduce la expresión que será evaluada para obtener el parámetro "map" especificado. Esta expresión siempre debe retornar un objeto *java.util.Map* en el que las claves son los nombres de los parámetros.

Además o en lugar de suministrar los valores de los parámetros en un "map", se puede proporcionar los valores de parámetros de forma individual, uno a uno, utilizando un elemento *<subreportParameter>* para cada parámetro que se desee, se tiene que especificar el nombre del parámetro correspondiente utilizando el atributo obligatorio *name* y proporcionar una expresión que será evaluada en tiempo de ejecución para obtener el valor de ese parámetro en particular, valor que será suministrado al proceso de llenado del subreporte.

Tenga en cuenta que se puede utilizar las dos formas para proporcionar valores a los parámetros del subreporte, al mismo tiempo. Cuando esto sucede, los valores de los parámetros especificados individualmente utilizando el elemento *<subreportParameter>*, reemplazarán los valores de los parámetros especificados en el parámetro "map", que corresponde al mismo parámetro del subreporte. Si el "map" no contiene los valores de los parámetros correspondientes, los valores de los parámetros especificados individualmente se añadirán al "map".

Cuando se proporcionan los valores de los parámetros del subreporte, hay que tener en cuenta que el generador de reportes afecta el objeto *java.util.Map* que recibe, añadiendo los valores de los parámetros incorporados que corresponden al subreporte. Este "map" también se ve afectado por los valores de los parámetros del subreporte especificados individualmente. Con el fin de evitar que se altere el objeto original *java.util.Map* que se envía, se lo puede incorporar en un objeto "map" diferente, antes de proporcionarlo al proceso de llenado del subreporte, de esta manera:

```
new HashMap(myOriginalMap)
```

Así, el objeto original "map" no se ve afectado y las modificaciones se realizan sobre el nuevo objeto "map". Esto es especialmente útil cuando se quiere proporcionar al subreporte el mismo conjunto de parámetros que recibió el reporte maestro, esto se logra usando el parámetro del reporte incorporado en el reporte maestro llamado *REPORT_PARAMETERS_MAP*. Sin embargo, no se quiere afectar el valor original de este parámetro incorporado y se lo realiza de la siguiente manera:

```
<parametersMapExpression>
        new HashMap($P{REPORT_PARAMETERS_MAP})
</parametersMapExpression>
```

4.9.4. Fuentes de datos para subreportes

Los subreportes necesitan de una fuente de datos para generar su contenido, al igual que un reporte normal. En la sección de llenado de reportes hemos visto, que cuando se llena un

reporte se tiene que la proporcionar un objeto de origen de datos o un objeto de conexión, dependiendo del tipo de reporte. Es decir, si tiene una consulta SQL interna y se desea ejecutarla para obtener los datos del reporte o se proporciona los datos del reporte directamente.

Los subreportes se comportan de la misma manera que los reporte y esperan recibir los mismos tipos de objetos cuando se van a llenar.

A un subreporte podemos proporcionarle un origen de datos usando el elemento <*dataSourceExpression*> o una conexión JDBC al generador de reportes para ejecutar la consulta SQL interna del subreporte usando el elemento <*connectionExpression*>. Estos dos elementos XML no pueden estar presentes al mismo tiempo en la declaración de un elemento tipo subreporte, esto es porque no se puede proporcionar tanto una fuente de datos y una conexión para el subreporte, entonces se deberá decidir que opción utilizar.

El generador de reportes espera que la expresión del origen de datos retorne un objeto *dori.jasper.engine.JRDataSource* y que la expresión de la conexión retorne un objeto *java.sql.Connnection*.

4.10. JASPERREPORTS AVANZADA

En las secciones anteriores se ha presentado la funcionalidad principal que la mayoría de usuarios llega a usar cuando trabajan con la librería JasperReports. Sin embargo, algunos requerimientos complejos de aplicaciones específicas podrían obligar a profundizar en la funcionalidad de la librería JasperReports a fin de adaptarla a sus necesidades.

A continuación vamos a echar un vistazo más de cerca a aquellos aspectos que pudieran interesarle si se quiere aprovechar al máximo la utilización de la librería JasperReports.

4.10.1. Cargando y editando diseños XML

Hemos explicado cómo los diseños de informes pasan de su forma inicial en XML al formato compilado, antes de ser utilizado en la generación de documentos listos para imprimir.

El primer paso del generador de reportes es analizar el diseño XML y crea la representación del mismo en memoria mediante la instanciación y preparación de un objeto *dori.jasper.engine. design.JasperDesign*. Este objeto es sometido a diferentes controles de validación y al proceso de compilación para producir su correspondiente objeto *dori.jasper.engine.JasperReport*.

Pero en algunos casos, tal vez se quiera cargar manualmente el diseño XML en un objeto *dori.jasper.engine.design.JasperDesign*, sin compilarlo inmediatamente. Este tipo de escenario se presenta en aplicaciones que mediante programación crean diseños y utilizan el formato XML para almacenarlos temporal o permanentemente.

Cargar un objeto *dori.jasper.engine.design.JasperDesign* desde un diseño XML puede resultar fácil si se usa uno de los métodos públicos estáticos *load ()* expuestos por la clase

dori.jasper.engine.xml.JRXmlLoader. De esta manera se pueden cargar los objetos de diseño desde el contenido XML almacenado en archivos o que puede ser leído desde algún flujo de entrada. El proceso opuesto al proceso de carga de diseño XML es la generación del formato XML para un objeto de diseño dado.

Como hemos visto, algunas veces los diseños se crean mediante programación, utilizando el API de JasperReports. Los objetos de diseño obtenidos de esta manera se pueden serializar, para almacenarlos en disco o transferirlos a través de la red, pero también pueden ser almacenados en formato XML.

Se puede obtener la representación XML de un objeto de diseño dado, mediante el uso de uno de los métodos públicos estáticos *writeReport ()* expuestos por la clase de utilidad *dori.jasper.engine.xml.JRXmlWriter*.

4.10.2. Implementando fuentes de datos

La librería JasperReports viene con diversas implementaciones predefinidas de la interfaz *dori.jasper.engine.JRDataSource*. Esta interfaz se utiliza para suministrar los datos al reporte cuando se invoca al proceso de llenado del reporte. Estas implementaciones predefinidas permiten generar reportes con los datos de bases de datos relacionales recuperados a través de un JDBC, de tablas Swing de Java o desde colecciones y arrays de objetosJavaBeans.

Pero quizás, los datos de la aplicación que se tratan de mostrar en el reporte tienen una estructura especial o están organizados de una manera particular que impiden el uso de una de las implementaciones predeterminadas de la interfaz origen de datos que viene con la librería. En tales situaciones, se puede crear implementaciones personalizadas de la interfaz *dori.jasper.engine.JRDataSource*, a fin de ajustar los datos especiales del reporte de modo que el generador de reportes pueda interpretarlos y usarlos para generar los reportes.

La creación de una implementación personalizada de la interfaz *dori.jasper.engine. JRDataSource* no es muy complicada, ya que se debe implementar sólo dos métodos. El primero es el método *next ()*, invocado por el motor generador de reportes cada vez que el puntero actual avanza al siguiente registro virtual del origen de datos. El otro, es el método *getFieldValue ()*, invocado por el motor generador de reportes con cada iteración del origen de datos para recuperar el valor de cada campo del reporte.

4.10.3. Personalizando visualizadores

La librería JasperReports viene con visualizadores incorporados que permiten visualizar los reportes almacenados en el formato propio de la librería o para previsualizar los diseños cuando se crean. Estos visualizadores están representados por las dos clases siguientes:

- ***dori.jasper.view.JasperViewer:*** Esta clase se usa para visualizar los reportes generados, ya sea como objetos en memoria u objetos serializados en el disco, incluso almacenados en formato XML.
- ***dori.jasper.view.JasperDesignViewer:*** Esta clase se puede utilizar para visualizar los diseños, ya sea en formato XML o de forma compilada.

Pero estos visualizadores incorporados, no pueden satisfacer las necesidades de todos y por lo tanto, se podría considerar su personalización; para que estos se adapten a ciertos requerimientos de las aplicaciones. Para hacer esto, se debe tener en cuenta que estos visualizadores hacen uso de componentes visuales más básicos que vienen con la librería JasperReports.

Los visualizadores de reportes antes mencionados utilizan el componente visual representado por la clase *dori.jasper.view.JRViewer* y sus compañeros. De hecho, es un componente especial *javax.swing.JPanel* que es capaz de visualizar reportes generados y puede fácilmente incorporarse en otras aplicaciones Java basadas en swing o applets. Si la funcionalidad de este componente visual básico no es suficiente para las necesidades de la aplicación, podría adaptarse mediante subclases. Por ejemplo, si se quiere tener un botón extra en la barra de herramientas de este visualizador, se podría considerar extender el componente y añadir el botón el diseñador mismo en el nuevo componente visual que se obtiene por medio de las subclases.

Otra funcionalidad muy importante, es que el visualizador de reportes predeterminado que viene con la librería no sabe cómo tratar con los hipervínculos de los documentos que apuntan a recursos externos. Este trata solamente con referencias locales y puede redireccionar el visualizador a las referencias locales correspondiente. Sin embargo, JasperReports ofrece la posibilidad de manipular los clics realizados en los hipervínculos de los documentos que apuntan a documentos externos y a referencias no locales.

La única cosa que se debe hacer para lograr esto, es implementar la interfaz *dori.jasper.view.JRHyperlinkListener* y añadir el registro con el componente visualizador una instancia de esta clase "listener", con el método *addHyperlinkListener ()* expuesto por la clase *dori.jasper.view.JRViewer*. De esta manera, el visor invocará la implementación del método *gotoHyperlink ()* en donde se direccionan las referencias externas.

4.10.4. Exportando a nuevos formatos

La librería JasperReports continuamente evoluciona y mejora. Entre las características que podrían introducirse con el tiempo está la factibilidad de exportar a otros nuevos formatos de documentos, adicionales a los ya conocidos como son: PDF, HTML y XML.

Con el fin de ampliar la diversificación de formatos de exportación, JasperReports ofrece la posibilidad de implementar una interfaz para ampliar la variedad de formatos de exportación, sin afectar su funcionalidad; con el fin de crear clases exportadoras que transformen los

documentos generados en nuevos formatos de salida. De esta manera, si se necesita exportar los reportes a un formato de salida especial que aún no está disponible en el núcleo de la librería, se puede implementar la interfaz *dori.jasper.engine.JRExporter*.

Antes de implementar esta interfaz, es importante saber cómo se espera que funcione la implementación.

Todos los datos de entrada que el exportador pueda necesitar deben ser suministrados por los llamados parámetros exportador, antes de que el proceso de exportación sea iniciado. Esto es porque el proceso de exportación siempre será invocado al llamar al método *ExportReport ()* de la interfaz *dori.jasper.engine.JRExporter*, y este método no recibe ningún parámetro en sí, cuando se lo llama. Los parámetros exportador tienen que ser establecidos usando el método *setParameter ()* en la instancia del exportador que se esté trabajando, antes de iniciar la tarea de exportación.

Para establecer los valores de la mayoría de los parámetros exportador se puede optar por el uso del método *setParameters ()*, que recibe un objeto *java.util.Map* que contiene los valores de los parámetros. Las entradas en el mapa deben ser instancias de la clase *ori.jasper.engine.JRExporterParameter*, como cuando se proporcionan individualmente llamando al método *setParameter ()* para cada uno de los parámetros exportador.

Nótese que no importa qué tipo de salida el exportador produce, se usarán parámetros para indicar al exportador dónde ubicar o enviar estas salidas. Dichos parámetros podrían ser llamados parámetros *OUT*.

Por ejemplo, si queremos que el exportador envíe la salida que produce a un flujo de salida, se deberá suministrar una referencia del objeto *java.io.OutputStream* al exportador utilizando un parámetro, probablemente identificado por la constante *dori.jasper.engine.JRExporter Parameter.OUTPUT_STREAM*. Se recomienda el uso de las constantes públicas de la clase *dori.jasper.engine.JRExporterParameter* para identificar los parámetros que se establecen en los exportadores y sólo si no encuentra un disponible para un contexto en particular se tiene que extender esta clase en el exportador, para añadir nuevas constantes. Esto puede ser apreciado en la clase *dori.jasper.engine.export.JRXmlExporter*, donde un identificador especial de parámetro especial crea subclases de la clase *dori.jasper.engine.JRExporterParameter* en la clase *dori.jasper.engine.export.JRXmlExporterParameter*.

No se tiene que empezar de cero cuando se implementa la interfaz exportadora, ya que existe una clase abstracta llamada *dori.jasper.engine.JRAbstractExporter* que se encarga de gestionar los parámetros.

CAPÍTULO V

BDD

Bases de Datos

INTRODUCCIÓN

SISTEMAS DE GESTIÓN DE BASES DE DATOS

OBJETIVOS DE LOS SGBD

ARQUITECTURA DE LOS SGBD

LENGUAJE SQL

ADMINISTRACIÓN DE BASES DE DATOS

ADAPTIVE SERVER ENTERPRISE

5.1. INTRODUCCIÓN

Las aplicaciones informáticas de los años sesenta acostumbraban a darse totalmente por lotes y estaban pensadas para una tarea muy específica relacionada con muy pocas entidades tipo.

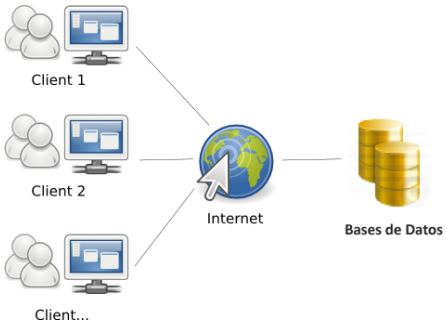
Cada aplicación utilizaba ficheros de movimientos para actualizar y/o para consultar uno o dos ficheros maestros o, excepcionalmente, más de dos. Cada programa trataba como máximo un fichero maestro, que solía estar sobre cinta magnética y, en consecuencia, se trabajaba con acceso secuencial. Cada vez que se le quería añadir una aplicación que requería el uso de algunos de los datos que ya existían y de otros nuevos, se diseñaba un fichero nuevo con todos los datos necesarios para evitar que los programas tuviesen que leer muchos ficheros.

A medida que se fueron introduciendo las líneas de comunicación, los terminales y los discos, se fueron escribiendo programas que permitían a varios usuarios consultar los mismos ficheros on-line y de forma simultánea. Más adelante fue surgiendo la necesidad de hacer las actualizaciones también on-line.

A medida que se integraban las aplicaciones, se tuvieron que interrelacionar sus ficheros y fue necesario eliminar la redundancia. El nuevo conjunto de ficheros se debía diseñar de modo que estuviesen interrelacionados; al mismo tiempo, la información redundante que figuraba en los ficheros de más de una de las aplicaciones, debía estar ahora en un solo lugar.

Estos conjuntos de ficheros interrelacionados con estructuras complejas y compartidas por varios procesos de forma simultánea recibieron al principio el nombre de Data Banks, y después, a inicios de los años setenta, el de Data Bases, cuya abreviatura es DB. El software de gestión de ficheros era demasiado elemental para dar satisfacción a todas estas necesidades.

La utilización de estos conjuntos de ficheros por parte de los programas de aplicación era excesivamente compleja, de modo que, especialmente durante la segunda mitad de los años setenta, fue saliendo al mercado software más sofisticado, los Data Base Management Systems DBMS (*Sistemas de Gestión de Bases de Datos*).



Fuente: [W01]

Figura 5.1. Esquema general de un sistema de base de datos.

En conclusión, podemos decir que una base de datos es un conjunto estructurado de datos que representa entidades y sus interrelaciones. La representación será única, integrada y que permita utilizaciones varias y simultáneas. El concepto de base de datos está relacionado con

[W01] Shelter Manager, http://www.sheltermanager.com/es_about.html

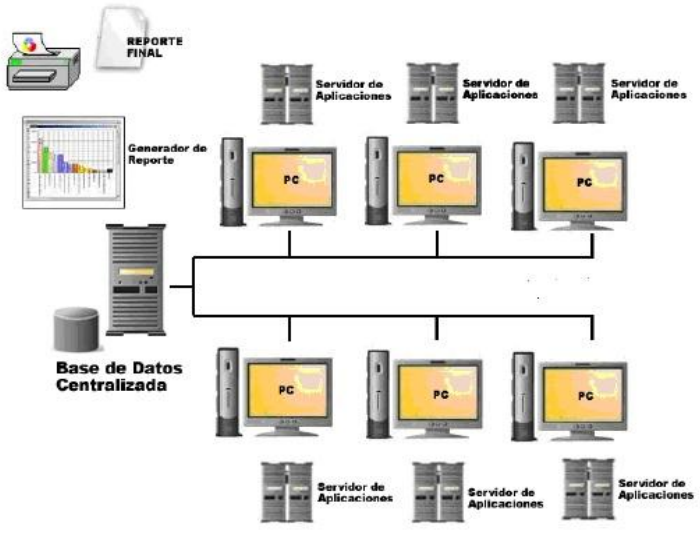
la red, ya que la información debe ser compartida y accesible a través de la red. El término utilizado para describir la estructura global que incluye todos los mecanismos para compartir los datos que conforman este sistema es "Sistema de Información".

5.2. SISTEMAS DE GESTIÓN DE BASES DE DATOS

Los primeros Sistemas de Gestión de Bases de Datos (*SGBD*), por los años sesenta, todavía no se les denominaba así estaban orientados a facilitar la utilización de grandes conjuntos de datos en los que las interrelaciones eran complejas. Estos sistemas trabajaban exclusivamente por lotes (*batch*).

Al aparecer los terminales de teclado, conectados al ordenador central mediante una línea telefónica, se empiezan a construir grandes aplicaciones on-line transaccionales. Los SGBD estaban íntimamente ligados al software de comunicaciones y de gestión de transacciones. Aunque para escribir los programas de aplicación se utilizaban lenguajes de alto nivel como Cobol; se disponía también de instrucciones y de subrutinas especializadas para tratar las BD, que requerían que el programador conociese muchos detalles del diseño físico, y que hacían que la programación fuese muy compleja. Puesto que los programas estaban relacionados con el nivel físico, se debían modificar continuamente cuando se hacían cambios en el diseño y la organización de la BD. La preocupación básica de estos sistemas era maximizar el rendimiento.

Los ordenadores minis, y después los ordenadores micros, extendieron la informática a prácticamente todas las empresas e instituciones. Esto exigía que el desarrollo de aplicaciones fuese más sencillo. Los SGBD de los años setenta eran demasiado complejos, inflexibles y sólo los podía utilizar un personal muy cualificado.



Fuente: [W02]

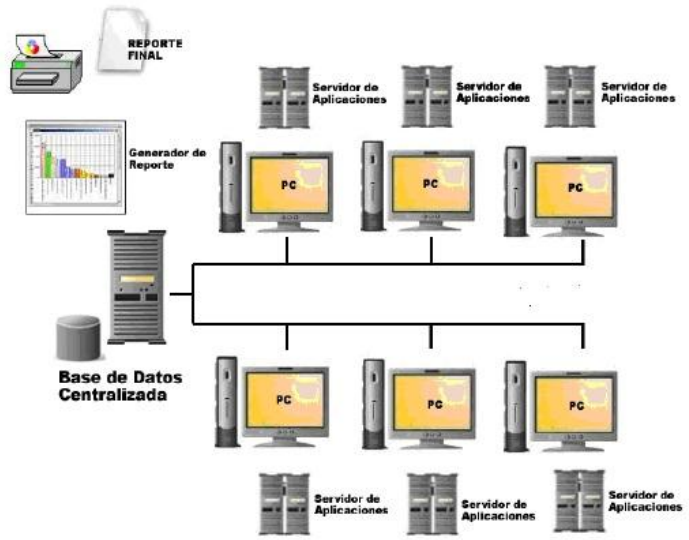
Figura 5.2. Esquema de una base de datos centralizada.

[W02] Bases de datos relacionales, http://mglopezc.blogspot.com/2011/04/aplicacion-de-computing-en-bases-de.html

Todos estos factores hacen que se extienda el uso de los SGBD. La estandarización, en el año 1986, del lenguaje SQL produjo una auténtica explosión de los SGBD relacionales.

Al acabar la década de los ochenta, los SGBD relacionales ya se utilizaban prácticamente en todas las empresas. A finales de los ochenta y principios de los noventa, las empresas se han encontrado con el hecho de que sus departamentos han ido comprando ordenadores departamentales y personales, y han ido haciendo aplicaciones con BD. El resultado ha sido que en el seno de la empresa hay numerosas BD y varios SGBD de diferentes tipos o proveedores. Este fenómeno de multiplicación de las BD y de los SGBD se ha visto incrementado por la fiebre de las fusiones de empresas.

Esta distribución ideal se consigue cuando las diferentes BD son soportadas por una misma marca de SGBD, es decir, cuando hay homogeneidad. Sin embargo, esto no es tan sencillo si los SGBD son heterogéneos. En la actualidad, gracias principalmente a la estandarización del lenguaje SQL, los SGBD de marcas diferentes pueden darse servicio unos a otros y colaborar para dar servicio a un programa de aplicación.



Fuente: [W02]

Figura 5.3. Esquema de una base de datos distribuida.

Además de esta distribución "impuesta", al querer tratar de forma integrada, distintas BD preexistentes, también se puede hacer una distribución “deseada”, diseñando una BD distribuida físicamente, y con ciertas partes replicadas en diferentes sistemas. Las razones básicas por las que interesa esta distribución son las siguientes:

- **Disponibilidad:** La disponibilidad de un sistema con una BD distribuida puede ser más alta, porque si queda fuera de servicio uno de los sistemas, los demás seguirán funcionando. Si los datos residentes en el sistema no disponible están replicados en

[W02] Bases de datos relacionales, http://mglopezc.blogspot.com/2011/04/aplicacion-de-computing-en-bases-de.html

otro sistema, continuarán estando disponibles. En caso contrario, sólo estarán disponibles los datos de los demás sistemas.

- **Coste:** Una BD distribuida puede reducir el coste. En el caso de un sistema centralizado, todos los equipos usuarios, que pueden estar distribuidos por distintas y lejanas áreas geográficas, están conectados al sistema central por medio de líneas de comunicación. El coste total de las comunicaciones se puede reducir haciendo que un usuario tenga más cerca los datos que utiliza con mayor frecuencia.

La tecnología que se utiliza habitualmente para distribuir datos es la que se conoce como arquitectura "cliente/servidor[1]". Todos los SGBD relacionales del mercado han sido adaptados a este entorno.

Fuente: [Propia]

Figura 5.4. SGBD en un entorno cliente servidor.

La facilidad para disponer de distribución de datos no es la única razón, ni siquiera la básica, del gran éxito de los entornos cliente/servidor en los años noventa. Tal vez el motivo fundamental ha sido la flexibilidad para construir y hacer crecer la configuración informática global de la empresa, así como de hacer modificaciones en ella, mediante hardware y software muy estándar y barato.

El éxito de las BD, incluso en sistemas personales, ha llevado a la aparición de los *Fourth Generation Languages* (4GL), lenguajes muy fáciles y potentes, especializados en el desarrollo de aplicaciones fundamentadas en BD. Proporcionan muchas facilidades en el momento de definir, generalmente de forma visual, diálogos para introducir, modificar y consultar datos en entornos cliente/servidor.

[1] La arquitectura cliente-servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes.

5.3. OBJETIVOS Y SERVICIOS DE LOS SGBD

Los Sistemas de Gestión de Bases de Datos que existen actualmente pretenden satisfacer los siguientes objetivos:

5.3.1. Realizar consultas no predefinidas y complejas

El objetivo fundamental de los SGBD es permitir que se hagan consultas no predefinidas y complejas, de manera sencilla y rápida. Los usuarios podrán realizar consultas de cualquier tipo y complejidad directamente al SGBD. El usuario debe formular la consulta con un lenguaje sencillo, que el sistema debe interpretar directamente.

El SGBD tendrá que responder inmediatamente sin que estas consultas estén preestablecidas; es decir, sin que se tenga que escribir, compilar y ejecutar un programa específico para cada consulta. Sin embargo, esto no significa que no se puedan escribir programas con consultas incorporadas, por ejemplo: procedimientos almacenados. La solución estándar para alcanzar este doble objetivo es el lenguaje SQL.

5.3.2. Flexibilidad e independencia

La complejidad de las bases de datos y la necesidad de irlas adaptando a la evolución de los sistemas de información hacen que un objetivo básico de los SGBD sea dar flexibilidad a los cambios.

Entonces, se debe obtener la máxima independencia posible entre los datos y los procesos de los usuarios para que se pueda llevar a cabo todo tipo de cambios tecnológicos y variaciones en la descripción de la base de datos, sin que se deban modificar los programas de aplicación ya escritos ni cambiar la forma de escribir las consultas o actualizaciones directas.

5.3.3. Eliminar la redundancia

En el mundo de los ficheros tradicionales, cada aplicación utilizaba su fichero. Sin embargo, puesto que se daba mucha coincidencia de datos entre aplicaciones, se producía también mucha redundancia entre los ficheros. De ahí que, uno de los objetivos de los SGBD es la eliminación de la redundancia.

Tal vez se piense que el problema de la redundancia es el espacio perdido por el almacenamiento de los datos. Pero, el verdadero inconveniente de la redundancia se evidencia cuando tenemos un mismo dato almacenado en dos lugares diferentes; no pasará demasiado tiempo hasta que las dos anotaciones dejen de ser coherentes, porque habremos modificado la anotación en uno de los lugares y nos habremos olvidado de hacerlo en el otro. Así pues, el verdadero problema es el grave riesgo de inconsistencia o incoherencia de los datos; es decir, la pérdida de integridad que las actualizaciones pueden provocar cuando existe redundancia.

Por lo tanto, convendría evitar la redundancia. En principio, nos conviene hacer que un dato sólo figure una vez en la base de datos. Sin embargo, esto no siempre será cierto. Por ejemplo, para representar una interrelación entre dos entidades, se suele repetir un mismo atributo en

las dos, para que una haga referencia a la otra. Los SGBD deben permitir que el diseñador defina datos redundantes, pero entonces tendría que ser el mismo SGBD el que hiciese automáticamente la actualización de los datos en todos los lugares donde estuviesen repetidos.

La duplicación de datos es el tipo de redundancia más habitual; pero también tenemos redundancia cuando guardamos en la BD, datos derivados o calculados a partir de otros datos de la misma BD. De este modo podemos responder rápidamente a consultas globales, ya que nos ahorramos la lectura de gran cantidad de registros.

En los casos de datos derivados, para que el resultado del cálculo se mantenga consistente con los datos elementales, es necesario rehacer el cálculo cada vez que éstos se modifican. El usuario puede olvidarse de hacer el nuevo cálculo; por ello convendrá que el mismo SGBD lo haga automáticamente.

5.3.4. Integridad de los datos

Nos interesará que los SGBD aseguren el mantenimiento de la calidad de los datos en cualquier circunstancia. Acabamos de ver que la redundancia puede provocar pérdida de integridad de los datos, pero no es la única causa posible. Se podría perder la corrección o la consistencia de los datos por muchas otras razones: errores de programas, errores de operación humana, avería de disco, transacciones incompletas por corte de alimentación eléctrica, etc.

En el punto anterior, hemos visto que podemos decirle al SGBD que nos lleve el control de las actualizaciones en el caso de las redundancias, para garantizar la integridad. Del mismo modo, podemos darle otras reglas o restricciones de integridad para asegurarse que los programas las cumplan cuando efectúen actualizaciones.

Al diseñar una base de datos para un sistema de información específico, no sólo definiremos los datos, sino también las reglas de integridad que queremos que el SGBD haga cumplir. Cuando el SGBD detecte que un programa quiere hacer una operación que va contra las reglas establecidas en la base de datos, no se lo deberá permitir, y tendrá que devolver un estado de error.

Aparte de las reglas de integridad que el diseñador de la base de datos puede definir y que el SGBD entenderá y hará cumplir; el mismo SGBD tiene reglas de integridad inherentes al modelo de datos que utiliza y que siempre se cumplirán. Son las denominadas reglas de integridad del modelo. Las reglas definibles por parte del usuario son las reglas de integridad del usuario. El concepto de integridad de los datos va más allá de prevenir que los programas usuarios almacenen datos incorrectos. En casos de errores o desastres, también podríamos perder la integridad de los datos. El SGBD nos debe brindar las herramientas necesarias para reconstruir o restaurar los datos estropeados.

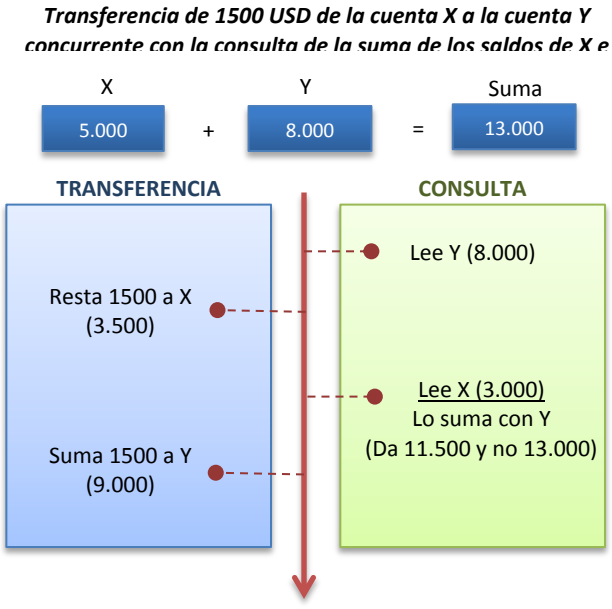
Los procesos de restauración (recovery) de los SGBD pueden reconstruir las bases de datos y darle el estado consistente y correcto anterior al incidente. Esto se acostumbra a hacer gracias a la obtención de copias periódicas de los datos (backups) y mediante el mantenimiento continuo de un diario (log) donde el SGBD va anotando todas las escrituras que se hacen en la base de datos.

5.3.5. Concurrencia de usuarios

Un objetivo fundamental de los SGBD es permitir que varios usuarios puedan acceder concurrentemente a la misma base de datos.

Cuando los accesos concurrentes son todos de lectura, es decir, cuando sólo se consulta; el problema que se produce es simplemente de rendimiento, causado por las limitaciones de los soportes que se dispone. Cuando un usuario o más de uno están actualizando los datos, se pueden producir problemas de interferencia que tengan como consecuencia la obtención de datos erróneos y la pérdida de la integridad de la base de datos.

Para tratar los accesos concurrentes, los SGBD utilizan el concepto de "transacción de BD", concepto de especial utilidad para todo aquello que hace referencia a la integridad de los datos. Denominamos transacción de BD a un conjunto de operaciones simples que se ejecutan como una unidad. Los SGBD deben conseguir que el conjunto de operaciones de una transacción nunca se ejecute parcialmente. Para indicar al SGBD que damos por acabada la ejecución de una transacción, el programa utilizará la operación de COMMIT. Si el programa no puede acabar normalmente, es decir, si el conjunto de operaciones se ha hecho sólo de forma parcial; el SGBD tendrá que deshacer todo lo que la transacción ya haya hecho. Esta operación se denomina ROLLBACK.



Fuente: [Propia]

Figura 5.5. Concurrencia de transacciones en un SGBD.

Acabamos de observar la utilidad del concepto de transacción para el mantenimiento de la integridad de los datos en caso de interrupción de un conjunto de operaciones lógicamente unitario. Sin embargo, entre transacciones que se ejecutan concurrentemente se pueden producir problemas de interferencia que hagan obtener resultados erróneos o que comporten la pérdida de la integridad de los datos.

Nos interesará que el SGBD ejecute las transacciones de forma que no se interfieran; es decir, que queden aisladas unas de otras. Para conseguir que las transacciones se ejecuten como si estuviesen aisladas, los SGBD utilizan distintas técnicas. La más conocida es el bloqueo (lock).

El bloqueo de unos datos en beneficio de una transacción consiste en poner limitaciones a los accesos que las demás transacciones podrán hacer a estos datos. Cuando se provocan bloqueos, se producen esperas, retenciones y, en consecuencia, el sistema es más lento. Los SGBD se esfuerzan en minimizar estos efectos negativos.

5.3.6. Seguridad

El término seguridad se ha utilizado en diferentes sentidos a lo largo de la historia de la informática. Actualmente, en el campo de los SGBD, el término seguridad se suele utilizar para hacer referencia a los temas relativos a la confidencialidad, las autorizaciones, los derechos de acceso, etc.

Los SGBD permiten definir autorizaciones o derechos de acceso a diferentes niveles: al nivel global de toda la base de datos, al nivel entidad y al nivel atributo. Estos mecanismos de seguridad requieren que el usuario se deba identificar; se acostumbra a utilizar códigos de usuarios y/o grupos de usuarios acompañados de contraseña, pero también se utilizan tarjetas magnéticas, identificación por reconocimiento de voz, etc.

También, nos puede interesar almacenar la información con una codificación secreta; es decir, con técnicas de encriptación, como mínimo se deberían encriptar las contraseñas. Prácticamente, todos los SGBD actuales, nos proveen de una gran variedad de herramientas para administrar la seguridad de los datos.

5.3.7. Otros objetivos

Acabamos de describir los objetivos fundamentales de los SGBD. Sin embargo, a medida que los SGBD evolucionan, se imponen nuevos objetivos adaptados a las nuevas necesidades y las nuevas tecnologías. Por lo tanto, podríamos citar también como objetivos de los SGBD, los siguientes:

- Servir eficientemente los *Data Warehouse*.
- Adaptarse al desarrollo orientado a objetos.
- Incorporar el tiempo como un elemento de caracterización de la información.
- Adaptarse al mundo de Internet.

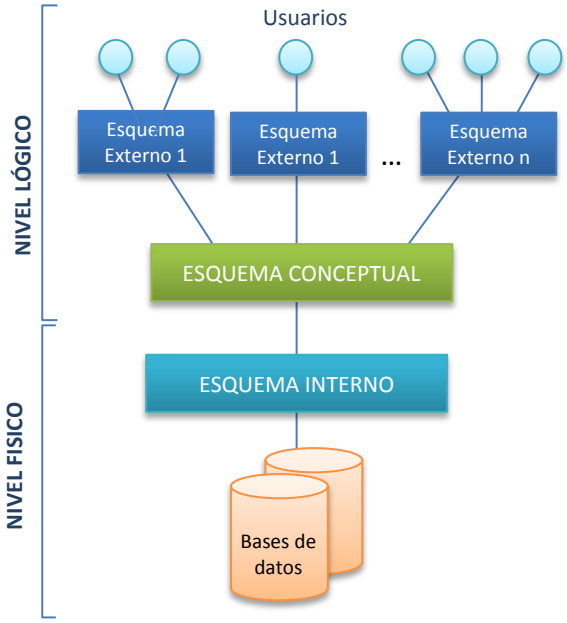
5.4. ARQUITECTURA DE LOS SGBD

5.4.1. Esquemas y niveles

Para trabajar con nuestras bases de datos, los SGBD necesitan conocer su definición o estructura. Esta descripción recibe el nombre de esquema de la base de datos, y los SGBD la tendrán continuamente a su alcance.

El esquema de la base de datos es un elemento fundamental de la arquitectura de un SGBD que permite independizar el SGBD de la base de datos; de este modo, se puede cambiar el diseño de la base de datos (su esquema) sin tener que hacer ningún cambio en el SGBD.

En el periodo 1975-1982, ANSI intentaba establecer las bases para crear estándares en el campo de las Bases de Datos. El comité conocido como ANSI/SPARC recomendó que la arquitectura de los SGBD previese tres niveles de descripción de la Base de Datos.



Fuente: [Propia]

Figura 5.6. Esquemas y niveles de un SGBD.

De este modo, y de acuerdo con ANSI/SPARC, habría los tres niveles de esquemas que mencionamos a continuación:

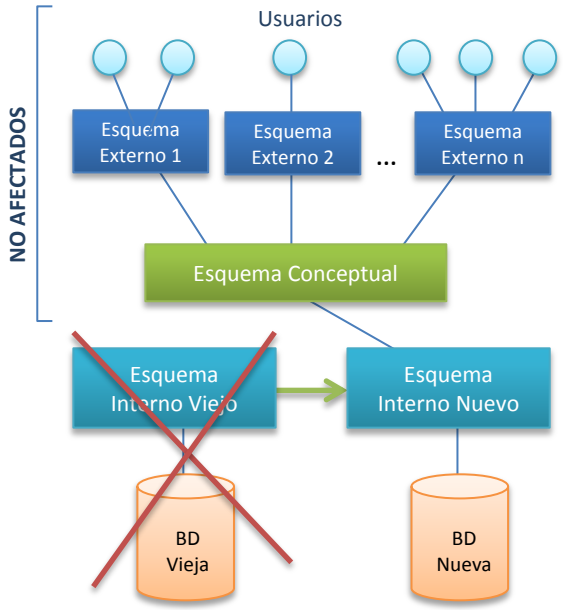
- **Nivel externo.-** En este nivel se sitúan las diferentes visiones lógicas que los procesos usuarios (programas de aplicación y usuarios directos) tendrán de las partes de la Base de Datos. Al definir un esquema externo, se citarán sólo aquellos atributos y aquellas entidades que interesen; los podremos redenominar, podremos definir datos derivados o redefinir una entidad para que las aplicaciones que utilizan este esquema externo crean que son dos, definir combinaciones de entidades para que parezcan una sola, etc. Todas estas visiones se denominan “esquemas externos”.

- **Nivel conceptual.-** En este nivel hay una sola descripción lógica básica, única y global, que denominamos "esquema conceptual", y que sirve de referencia para el resto de los esquemas. En el esquema conceptual se describirán las entidades tipo, sus atributos, las interrelaciones y las restricciones o reglas de integridad.
- **Nivel físico.-** En este nivel hay una sola descripción física, que denominamos "esquema interno". El esquema interno o físico contendrá la descripción de la organización física de la Base de Datos: caminos de acceso (índices, hashing, apuntadores, etc.), codificación de los datos, gestión del espacio, tamaño de la página, etc. El esquema de nivel interno responde a las cuestiones de rendimiento planteadas al hacer el diseño físico de la Base de datos y al ajustarlo posteriormente a las necesidades cambiantes.

5.4.2. Independencia de los datos

En este punto describiremos cómo la arquitectura de tres niveles nos proporciona los dos tipos de independencia de los datos: la física y la lógica.

Hay independencia física cuando los cambios en la organización física de la Base de Datos no afectan al mundo exterior; es decir, los programas usuarios o los usuarios directos. De acuerdo con la arquitectura ANSI/SPARC, habrá independencia física cuando los cambios en el esquema interno no afecten al esquema conceptual ni a los esquemas externos.



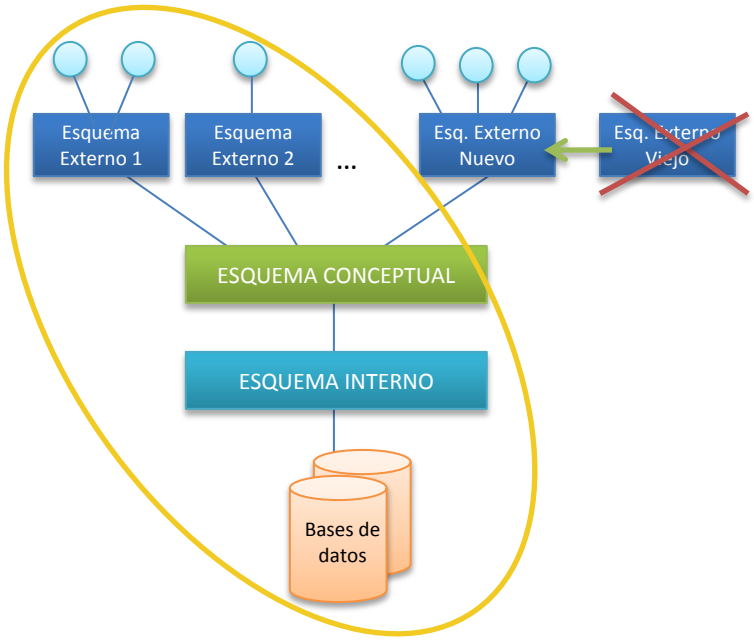
Fuente: [Propia]

Figura 5.7. Independencia Física.

Es obvio que cuando cambiemos unos datos de un soporte a otro, o los cambiemos de lugar dentro de un soporte, no se verán afectados ni los programas de aplicación ni los usuarios directos, ya que no se modificará el esquema conceptual ni el externo. Tampoco tendrían que verse afectados si cambiamos, por ejemplo, el método de acceso a unos registros

determinados, el formato o la codificación, etc. Ninguno de estos casos debería afectar al mundo exterior, sino sólo a la Base de Datos física, el esquema interno, etc. Si hay independencia física de los datos, lo único que variará al cambiar el esquema interno son las correspondencias entre el esquema conceptual y el interno. Obviamente, la mayoría de los cambios del esquema interno obligarán a rehacer la BD real (la física).

Hay **independencia lógica** cuando los programas de aplicación o usuario directos no se ven afectados por los cambios en el nivel lógico.



Fuente: [Propia]

Figura 5.8. Independencia Lógica.

Dados los dos niveles lógicos de la arquitectura ANSI/SPARC, diferenciaremos las dos situaciones siguientes:

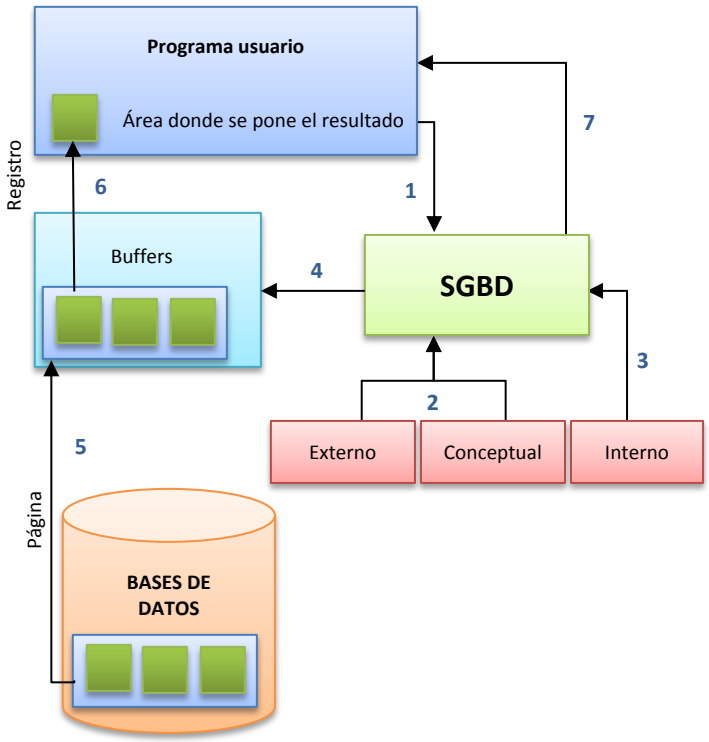
1. **Cambios en el esquema conceptual.-** Un cambio de este tipo no afectará a los esquemas externos que no hagan referencia a las entidades o a los atributos modificados.
2. **Cambios en los esquemas externos.-** Efectuar cambios en un esquema externo afectará a los usuarios que utilicen los elementos modificados. Sin embargo, no debería afectar a los demás usuarios ni al esquema conceptual; y tampoco, al esquema interno y a la Base de Datos física.

Los SGBD actuales proporcionan bastante independencia lógica, pero menos de la que haría falta, ya que las exigencias de cambios constantes en el Sistema de Información piden grados muy elevados de flexibilidad. Los sistemas de ficheros tradicionales, en cambio, no ofrecen ninguna independencia lógica.

5.4.3. Flujo de datos y de control

Para entender el funcionamiento de un SGBD, a continuación veremos los principales pasos de la ejecución de una consulta sometida al SGBD por un programa de aplicación. Explicaremos las líneas generales del flujo de datos y de control entre el SGBD, los programas de usuario y la Base de Datos.

Supongamos que la consulta pide los datos de un alumno que tiene un determinado número de identidad. Por lo tanto, la respuesta que el programa obtendrá será un solo registro y lo recibirá dentro de un área de trabajo propia (una variable).



Fuente: [Propia]

Figura 5.9. Flujo de datos y control.

El proceso que se sigue es el siguiente:

1. Empieza con una llamada del programa al SGBD, en la que se le envía la operación de consulta. El SGBD debe verificar que la sintaxis de la operación recibida sea correcta, que el usuario del programa esté autorizado a hacerla, etc.
2. Para poder llevar a cabo todo esto, el SGBD se basa en el esquema externo con el que trabaja el programa y en el esquema conceptual.
3. Si la consulta es válida, el SGBD determina, consultando el esquema interno, qué mecanismo debe seguir para responderla. Ya sabemos que el programa usuario no dice nada respecto a cómo se debe hacer físicamente la consulta. Es el SGBD el que lo debe determinar. Casi siempre hay varias formas y diferentes caminos para responder a una consulta. Supongamos que ha elegido aplicar un hashing al valor del número de

identidad, que es el parámetro de la consulta, y el resultado es la dirección de la página donde se encuentra (entre muchos otros) el registro del alumno buscado.

4. Cuando ya se sabe cuál es la página, el SGBD comprobará si por suerte esta página ya se encuentra en aquel momento en el área de los buffers (tal vez como resultado de una consulta anterior de este usuario o de otro).
5. Si no está, el SGBD, con la ayuda del Sistema Operativo, la busca en disco y la carga en los buffers. Si ya está, se ahorra el acceso a disco.
6. Ahora, la página deseada ya está en la memoria principal. El SGBD aplica a los datos las eventuales transformaciones lógicas que implica el esquema externo y las lleva al área de trabajo del programa.
7. Por último, el SGBD retorna el control al programa y da por terminada la ejecución de la consulta.

5.5. EL LENGUAJE SQL

5.5.1. Definición

El lenguaje de consulta estructurado o SQL (*Structured Query Language*) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en éstas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo efectuar consultas con el fin de recuperar, de una forma sencilla, información de interés de una base de datos, así como también hacer cambios sobre ella.

5.5.2. Orígenes y evolución

Los orígenes del lenguaje SQL están ligados a las de las bases de datos relacionales. En 1970, *Edgar Frank Codd*[2] propone el modelo relacional y asociado a éste un sublenguaje de acceso a los datos basado en el cálculo de predicados. Basándose en estas ideas, los laboratorios de IBM definen el lenguaje SEQUEL (*Structured English QUEry Language*) que más tarde sería ampliamente implementado por el Sistema de Gestión de Bases de Datos (SGBD) experimental System R, desarrollado en 1977 también por IBM[3]. Sin embargo, fue Oracle[4] quien lo introdujo por primera vez en 1979 en un programa comercial.

Al final de la década de los setenta y al principio de la de los ochenta, una vez finalizado el proyecto System R, IBM[3] y otras empresas empezaron a utilizar el SQL en sus SGBD relacionales, con lo que este lenguaje adquirió una gran popularidad.

En 1982, ANSI[5] encargó a uno de sus comités la definición de un lenguaje de bases de datos relacionales. Este comité, después de evaluar diferentes lenguajes, y ante la aceptación

[2] *Edgar Frank Codd* (23 de agosto de 1923 - 18 de abril de 2003), Científico informático inglés, conocido por sus aportes a la teoría de bases de datos relacionales.
[3] *International Business Machines*, empresa multinacional estadounidense de tecnología con sede en Armonk, Nueva York.
[4] *Oracle Corporation* es una de las mayores compañías de software del mundo.
[5] *American National Standards Institute*, es una organización sin fines de lucro que supervisa el desarrollo de estándares para productos, servicios, procesos y sistemas en los Estados Unidos.

comercial del SQL, eligió un lenguaje estándar que estaba basado en éste prácticamente en su totalidad. El SQL se convirtió oficialmente en el lenguaje estándar de ANSI[5] en el año 1986, y de ISO[6] en 1987. También ha sido adoptado como lenguaje estándar por FIPS[7], Unix X/Open[8] y SAA (*Systems Application Architecture*) de IBM[3].

En el año 1989, el estándar fue objeto de una revisión y una ampliación que dieron lugar al lenguaje que se conoce con el nombre de SQL89 o SQL1. En el año 1992 el estándar volvió a ser revisado y ampliado considerablemente para cubrir carencias de la versión anterior. Esta nueva versión del SQL, que se conoce con el nombre de SQL92 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 general y muy amplio.

El ANSI SQL sufrió varias revisiones y agregados a lo largo del tiempo:

AÑO	NOMBRE	ALIAS	COMENTARIOS
1986	SQL-86	SQL-87	Primera publicación hecha por ANSI. Confirmada por ISO en 1987.
1989	SQL-89		Revisión menor.
1992	SQL-92	SQL2	Revisión mayor.
1999	SQL:1999	SQL2000	Se agregaron expresiones regulares, consultas recursivas, triggers y algunas características orientadas a objetos.
2003	SQL:2003		Introduce algunas características de XML, cambios en las funciones, estandarización del objeto sequence y de las columnas autonumericas.
2006	SQL:2006		ISO/IEC 9075-14:2006 Define las maneras en las cuales el SQL se puede utilizar conjuntamente con XML. Define maneras importar y guardar datos XML en una base de datos SQL, manipulándolos dentro de la base de datos y publicando el XML y los datos SQL convencionales en forma XML. Además, proporciona facilidades que permiten a las aplicaciones integrar dentro de su código SQL el uso de XQuery, lenguaje de consulta XML publicado por el W3C (World Wide Web Consortium) para acceso concurrente a datos ordinarios SQL y documentos XML.
2008	SQL:2008		Permite el uso de la cláusula ORDER BY fuera de las definiciones de los cursores. Incluye los disparadores del tipo INSTEAD OF. Añade la sentencia TRUNCATE.

Fuente: [L01]

Tabla 5.1. Versiones del lenguaje SQL distribuidas a través del tiempo.

[6] *Organización Internacional de Normalización*, (23 de febrero de 1947), es el organismo encargado de promover el desarrollo de normas internacionales de fabricación, comercio y comunicación a excepción de la electrónica.

[7] *Federal Information Processing Standards*, son estándares desarrollados por el gobierno de los Estados Unidos para la utilización por parte de todas las agencias del gobierno no militares y por los contratistas del gobierno.

[8] *X/Open Company, Ltd*. fue un consorcio fundado por varios manufacturadores europeos de UNIX en 1984 para identificar y promover el estándar abierto en el campo de la tecnología de la información.

[L01] *Introducción a las bases de datos*, Rafael Camps Paré.

5.5.3. Características generales del SQL

El SQL es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales permitiendo gran variedad de operaciones en éstos últimos.

Es un lenguaje declarativo de "alto nivel" o "de no procedimiento", que gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros, y no a registros individuales, permite una alta productividad en codificación y la orientación a objetos. De esta forma una sola sentencia puede equivaler a uno o más programas que se utilizarían en un lenguaje de bajo nivel orientado a registros.

Como suele ser común en los lenguajes de acceso a bases de datos de alto nivel, el SQL es un lenguaje declarativo. O sea, que especifica qué es lo que se quiere y no cómo conseguirlo, por lo que una sentencia no 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 se hace necesario que éste lleve a cabo una optimización antes de su ejecución. Muchas veces, el uso de índices acelera una instrucción de consulta, pero ralentiza la actualización de los datos. Dependiendo del uso de la aplicación, se priorizará el acceso indexado o una rápida actualización de la información. La optimización difiere sensiblemente en cada motor de base de datos y depende de muchos factores.

5.5.4. Lenguaje de definición de datos (DDL)

El lenguaje de definición de datos (*Data Definition Language, o DDL*), es el que se encarga de la modificación de la estructura de los objetos de la base de datos. Existen cuatro operaciones básicas: *CREATE*, *ALTER*, *DROP* y *TRUNCATE*.

5.5.4.1. *CREATE*

Este comando crea un objeto dentro de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier otro objeto que el motor de la base de datos soporte. Por ejemplo:

```
CREATE OR REPLACE FUNCTION 'nombre_funcion' ('parametros')
RETURNS 'tipo_retorno' AS
$BODY$
BEGIN
       -- Instrucciones SQL
       -- Por ejemplo:
       DELETE FROM empleado WHERE idempleado = 'ANY' (ids);
END;
$BODY$
LANGUAGE 'plpgsql';
```

5.5.4.2. *ALTER*

Este comando permite modificar la estructura de un objeto. Se pueden agregar o quitar campos a una tabla, modificar el tipo de un campo, agregar o quitar índices a una tabla, modificar un trigger, etc. Por ejemplo:

```
ALTER TABLE 'nombre_tabla' (
        ADD nuevo_campo INT UNSIGNED meel
)
```

5.5.4.3. *DROP*

Este comando elimina un objeto de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento, etc. Se puede combinar con la sentencia ALTER. Por ejemplo:

```
ALTER TABLE 'nombre_tabla' (
          DROP COLUMN 'nombre_campo'
)
```

5.5.4.4. *TRUNCATE*

Este comando trunca todo el contenido de una tabla. La ventaja sobre el comando DROP, es que si se quiere borrar todo el contenido de la tabla, es mucho más rápido, especialmente si la tabla es muy grande. La desventaja es que TRUNCATE sólo sirve cuando se quiere eliminar absolutamente todos los registros, ya que no se permite la cláusula WHERE. Si bien, en un principio, esta sentencia parecería ser DML (Lenguaje de Manipulación de Datos), es en realidad una DDL, ya que internamente, el comando TRUNCATE borra la tabla y la vuelve a crear y no ejecuta ninguna transacción. Ejemplo:

```
TRUNCATE TABLE 'nombre_tabla'
```

5.5.5. Lenguaje de manipulación de datos (DML)

Un lenguaje de manipulación de datos (*Data Manipulation Language, o DML*) es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado.

El lenguaje de manipulación de datos más popular hoy en día es SQL, usado para recuperar y manipular datos en una base de datos relacional.

5.5.5.1. *INSERT*

Una sentencia INSERT de SQL agrega uno o más registros a una tabla en una base de datos relacional. La estructura básica de una sentencia INSERT es:

```
INSERT INTO 'tabla' ('columna1', ['columna2',...]) VALUES ('valor1',
['valor2',...])
```

Las cantidades de columnas y valores deben ser iguales. Si una columna no se especifica, le será asignado el valor por omisión. Los valores especificados (o implícitos) por la sentencia INSERT deberán satisfacer todas las restricciones aplicables. Si ocurre un error de sintaxis o si alguna de las restricciones es violada, no se agrega la fila y se devuelve un error. Por ejemplo:

```
INSERT INTO agenda (nombre, telefono) VALUES ('Wilson', '062612277')
```

Cuando se especifican todos los valores de una tabla, se puede utilizar la sentencia acortada, de la siguiente manera:

```
INSERT INTO 'tabla' VALUES ('valor1', ['valor2',...])
```

Asumiendo que 'nombre' y 'telefono' son las únicas columnas de la tabla 'agenda', la sentencia SQL quedaría de la siguiente manera:

```
INSERT INTO agenda VALUES ('Wilson', '062612277')
```

FORMAS AVANZADAS

Inserciones en múltiples filas

Una característica de SQL (desde SQL-92) es el uso de constructores de filas para insertar múltiples filas a la vez, con una sola sentencia SQL.

```
INSERT INTO 'tabla' ('columna1', ['columna2',...])
VALUES ('valor1a', ['valor1b',...]),
       ('valor2a', ['valor2b',...]), ...
```

Por ejemplo, asumiendo que 'nombre' y 'telefono' son las únicas columnas en la tabla 'agenda', podríamos insertar datos de la siguiente manera:

```
INSERT INTO agenda VALUES ('Wilson', '062612277'), ('Carlos', '062907342')
```

Que podía haber sido realizado por las sentencias

```
INSERT INTO agenda VALUES ('Wilson', '062612277')
INSERT INTO agenda VALUES ('Carlos', '062907342')
```

Notar que las sentencias separadas pueden tener semántica diferente, especialmente con respecto a los triggers; y puede tener diferente rendimiento que la sentencia de inserción múltiple.

Copia de filas de otras tablas

Un INSERT también puede utilizarse para recuperar datos de otros, modificarla si es necesario e insertarla directamente en la tabla. Todo esto se hace en una sola sentencia SQL que no implica ningún procesamiento intermedio en la aplicación cliente. Un SUBSELECT se utiliza en lugar de la cláusula VALUES. El SUBSELECT puede contener JOIN, llamadas a funciones, y puede incluso consultar en la misma TABLA los datos que se inserta. Lógicamente, el SELECT se evalúa

antes que la operación INSERT esté iniciada. A continuación un ejemplo de este tipo de instrucciones:

```
INSERT INTO agenda2 SELECT * FROM agenda WHERE nombre IN ('Wilson', 'Carlos')
```

Una variación es necesaria cuando algunos de los datos de la tabla fuente se están insertando en la nueva tabla, pero no todo el registro o cuando los esquemas de las tablas no son iguales.

```
INSERT INTO agenda2 (name, number)
SELECT nombre, telefono FROM agenda WHERE name IN ('Wilson', 'Carlos')
```

El SELECT produce una tabla temporal y el esquema de la tabla temporal debe coincidir con el esquema de la tabla donde los datos son insertados.

5.5.5.2. UPDATE

Una sentencia UPDATE de SQL es utilizada para modificar los valores de un conjunto de registros existentes en una tabla, la forma básica de la sentencia es:

```
UPDATE nombre_tabla SET campo1 = 'nuevo_valor' WHERE campo2 = 'valor'
```

Por ejemplo:

```
UPDATE agenda SET telefono = '062965462' WHERE nombre = 'Carlos'
```

5.5.5.3. DELETE

Una sentencia DELETE de SQL borra uno o más registros existentes en una tabla. La forma básica de esta sentencia, es:

```
DELETE FROM nombre_tabla WHERE 'columna1' = 'valor'
```

Por ejemplo:

```
DELETE FROM agenda WHERE nombre = 'Carlos'
```

5.6. ADMINISTRACIÓN DE BASES DE DATOS

En los sistemas de gestión de bases de datos existe un tipo de usuario especial que realiza las tareas de administración y control de la base de datos. Una empresa o institución que tenga uno o varios sistemas de información construidos en torno a bases de datos, necesita que alguien lleve a cabo una serie de funciones centralizadas de gestión y administración, para asegurar que la explotación de la base de datos es la correcta. Este conjunto de funciones se conoce con el nombre de administración de base de datos (*ABD*), y los usuarios que hacen este tipo especial de trabajo se denominan administradores de base de datos.

Los administradores de bases de datos son los responsables del correcto funcionamiento de la base de datos y velan para que siempre se mantenga útil. Intervienen en situaciones problemáticas o de emergencia, pero su responsabilidad fundamental es velar para que no se

produzcan incidentes. A continuación, enumeramos una lista de tareas típicas del administrador de bases de datos:

1. Mantenimiento, administración y control de los esquemas. Comunicación de los cambios a los usuarios.
2. Asegurar la máxima disponibilidad de los datos; por ejemplo, haciendo copias (*back-ups*), administrando diarios (*logs*), reconstruyendo la base de datos, etc.
3. Resolución de emergencias.
4. Vigilancia de la integridad y de la calidad de los datos.
5. Diseño físico, estrategia de caminos de acceso y reestructuraciones.
6. Control del rendimiento y decisiones relativas a las modificaciones en los esquemas y/o en los parámetros del sistema de gestión de bases de datos y del sistema operativo, para mejorarlo.
7. Normativa y asesoramiento a los programadores y a los usuarios finales sobre la utilización de la base de datos.
8. Control y administración de la seguridad: autorizaciones, restricciones, etc.

Los sistemas de gestión de bases de datos del mercado procuran reducir al mínimo el volumen de estas tareas, pero en sistemas muy grandes y críticos se llega a tener grupos de administradores de bases de datos de más de diez personas. Buena parte del software que acompaña el SGBD está orientado a facilitar la gran diversidad de tareas controladas por el administrador de bases de datos: monitores del rendimiento, monitores de la seguridad, verificadores de la consistencia entre índices y datos, reorganizadores, gestores de las copias de seguridad, etc. La mayoría de estas herramientas tienen interfaces visuales para facilitar la tarea del administrador de bases de datos.

5.7. ADAPTIVE SERVER ENTERPRISE

Adaptive Server Enterprise (*ASE*) es el motor de bases de datos (*SGBD*) insignia de la compañía Sybase. ASE es un sistema de gestión de bases de datos, altamente escalable, de alto rendimiento, con soporte a grandes volúmenes de datos, transacciones y usuarios, y de bajo costo, que permite:

- Almacenar datos de manera segura
- Tener acceso y procesar datos de manera inteligente
- Movilizar datos

5.7.1. Historia

ASE es directo descendiente de Sybase SQL Server (lanzada al mercado en 1988 como la primera base de datos con arquitectura cliente/servidor[1]) y su cambio de nombre se produjo a partir de la versión 11.5, en 1996, para evitar confusiones con Microsoft SQL Server, con el que

comparte un origen común (Sybase licenció el código a Microsoft[9] para el sistema operativo Windows). En 1998, se lanzó ASE 11.9.2, con soporte al bloqueo a nivel de registro y rendimiento mejorado en ambientes SMP. ASE 12.0 fue liberado en 1999, brindando soporte para Java en la base de datos, alta disponibilidad y gestión de transacciones distribuidas. En 2001, ASE 12.5 fue lanzada, con características tales como asignación dinámica de memoria, soporte para XML en la base de datos y conexiones seguras con SSL, entre otros. En septiembre de 2005, Sybase lanzó al mercado ASE 15.

5.7.2. Características en Adaptive Server Enterprise 12.5

La versión 12.5 de ASE incluye características como:

5.7.2.1. Instalación y configuración más fáciles

ASE 12.5 usa el instalador Install Shield, el cual le simplifica de gran manera el proceso de instalación y configuración de los componentes de ASE.

5.7.2.2. Caché de datos dinámico

ASE 12.5 le permite cambiar dinámicamente el caché de datos sin tener que reiniciar el servidor, contrario a lo que se requería en versiones anteriores. Usando acciones dinámicas de sp_cacheconfig, usted puede liberar memoria de tal manera que sea reasignada inmediatamente.

Algunas de las acciones que usted puede llevar a cabo dinámicamente con sp_cacheconfig son:

- Agregar un nuevo caché
- Agregar memoria a un caché existente
- Borrar un caché, de log o mixto
- Cambiar el tipo de un caché (mixto/log)

Si ASE no puede asignar toda la memoria requerida, asigna la memoria disponible. La memoria adicional es asignada al reiniciar de ASE.

Reducir el tamaño de un caché no es una acción dinámica. En vez de usar sp_cacheconfig para reducir el tamaño de un caché como una acción estática, remueva el caché y vuélvalo a crear con el tamaño correcto.

5.7.2.3. Expansión automática de bases de datos

ASE 12.5 permite la expansión automática de bases de datos y dispositivos. Las bases de datos pueden ser configuradas para expandirse automáticamente al quedarse sin espacio.

El procedimiento almacenado del sistema sp_dbextend le permite instalar umbrales que identifican cuales dispositivos tienen espacio y luego alterar de manera apropiada la base de

[9] *Microsoft Corporation*, es una empresa multinacional fundada el 4 de abril de 1975 por Bill Gates, Steve Ballmer y Paul Allen. Dedicada al sector de la informática, con sede en Redmond, Washington, Estados Unidos.

datos sobre esos dispositivos; sp_dbextend es la interfaz de usuario para automatizar el proceso de expansión. El administrador de la base de datos lo puede usar para:

- Definir reglas para segmentos y dispositivos individuales.
- Cambiar las políticas definidas por los usuarios y el comportamiento predeterminado del sistema.
- Examinar el estado actual de las reglas.

Los administradores también pueden usar esta interfaz para ejecutar simulaciones del proceso de expansión y, de ser necesario, llevar a cabo una expansión manual de una base de datos sin la necesidad de esperar a que el proceso automático de expansión sea ejecutado.

Después de configurar una base de datos para que sea expandida automáticamente, cuando la base de datos crece hasta su umbral de espacio libre, mecanismos internos se disparan, aumentando el tamaño de la base de datos en la cantidad especificada por las políticas de expansión. El proceso automático de expansión mide la cantidad de espacio libre sobre todos los dispositivos asociados a la base de datos. Si hay suficiente espacio en el dispositivo, la base de datos continua creciendo. Si hay dispositivos configurados para expansión, los dispositivos son expandidos a continuación, continuando con la expansión de la base de datos sobre esos dispositivos. El proceso automático de expansión corre como una tarea en background y genera mensajes informativos sobre su progreso en el log de errores de ASE.

El proceso automático de expansión requiere de nuevos procedimientos de instalación. Se puede instalar usando el script installdbextend, el cual agrega filas a master.dbo. sysattributes; filas que describen valores por defecto para la auto expansión en una base de datos o en un dispositivo. El proceso de expansión automático se puede instalar sobre una o más bases de datos o más dispositivos.

Podemos configurar la expansión automática para que se ejecute con políticas predeterminadas a nivel de servidor, o se puede personalizarla para segmentos específicos en bases de datos especificadas. También, se puede instalar umbrales sobre segmentos críticos sobre los que residan tablas con datos críticos, permitiéndole tener un muy detallado grado de control sobre cómo ASE responde a los requerimientos de espacio para diferentes clases de tablas. Si su sitio tiene tablas importantes con grandes volúmenes de insert, el ASE puede asociar estas tablas a segmentos específicos, con reglas específicas para extender ese segmento; esto permite evitar quedarse sin espacio en un ambiente productivo debido a grandes cargas sobre ese tipo de tablas críticas.

Los procedimientos de expansión automáticas permiten simular el proceso de expansión basándose en las políticas elegidas. Esta simulación permite verificar que el proceso de expansión trabaja adecuadamente, sin arriesgar el ambiente de producción. El procedimiento automático de expansión no crea nuevos dispositivos, sólo altera el tamaño de la base de datos y segmento sobre dispositivos existentes a los cuales el segmento esté asociado.

5.7.2.4. Mejoras al Plug-In de ASE para Sybase Central

El plug-in de ASE para Sybase Central ha sido mejorado en 12.5. El plug-in de ASE, como el de Sybase IQ y Adaptive Server Anywhere, ahora corre sobre Sybase Central 4.1. Esta versión de Sybase Central brinda mejor rendimiento, requiere menos memoria y mejora la interfaz de usuario.

El plug-in de ASE 12.5.1:

- Permite ver y exportar datos desde tablas o vistas usando el panel "Data".
- Permite ver el código SQL de objetos compilados (procedimientos, vistas, reglas, etc.), usando el panel "Code".
- Permite registrar comandos SQL en una ventana o archivo de tal manera que exista un registro de las acciones llevadas a cabo por el plug-in.
- Brinda un robusto soporte al archivo interfaces.
- Brinda un editor mejorado de tablas para soportar propiedades de columnas y asociación de restricciones.
- Permite manejar múltiples bases de datos tempdb.
- Incluye asistentes para los comandos quiesce y mount/unmount database.
- Incluye un asistente mejorado para tablas proxy creadas sobre fuentes de datos diferentes a ASE.
- Permite manejar el Job Scheduler.
- Permite bajar ASE.
- Permite cambiar el tamaño de dispositivos.
- Permite usar la interfaz gráfica de Monitor Server en Windows.

5.7.2.5. Puertos de Red (Listeners) Dinámicos

Esta versión de ASE agrega funcionalidad a *sp_listener*, la cual permite manejar los puertos de red, *sp_listener* permite:

- Arrancar puertos adicionales.
- Parar puertos.
- Suspender puertos.
- Reanudar puertos suspendidos.

La sintaxis de sp_listener es:

```
sp_listener "comando", "servidor", "engine" | remaining
```

o:

```
sp_listener "comando", "[protocolo:]máquina:puerto", "engine" | remaining
```

dónde:

- **comando:** start, stop, suspend, resume o status

- **servidor:** es el nombre de ASE.
- **engine:** especifica el número del engine afectado por este comando (este parámetro es ignorado en Windows). engine puede ser un sólo número entre comillas ("2"), una lista ("3,5,6"), un rango ("2-5") o una mezcla ("2,3-5,7").
- **remaining:** especifica que el comando debe entrar en efecto sobre todos los engines en los que pueda ser aplicado (o sea, donde el puerto esté en un estado en el cual el comando pueda tener efecto).
- **protocolo:** es el protocolo a ser utilizado (tcp, tli, ssltcp, ssltli, winsock, sslnlwnsck o sslwinsock).
- **máquina:** puerto es el nombre de máquina y puerto en el cual el puerto escucha (tal y como se especifica en el archivo interfaces).

El número de puertos es determinado por el parámetro dinámico de configuración number of network listeners. El valor predeterminado de éste parámetro es 1. La semántica para sp_listner es atómica, si un comando no se puede completar exitosamente, es abortado.

5.7.2.6. Mejor soporte a datos globales

ASE 12.5 soporta el modo de ordenamiento UTF-8 y el analizador Unicode.

Modo de Ordenamiento UTF-8

En versiones de ASE anteriores a la 12.5, al usar UTF-8, las dos únicas opciones eran "binary" y "no-case" (ASCII[10]). ASE 12.5 brinda la habilidad de llevar a cabo ordenamientos no binarios en UTF-8. De ésta manera, todos los modos de ordenamiento disponibles para los tipos de dato unichar y univarchar pueden ser usados para datos char y varchar cuando el modo de ordenamiento predeterminado sea UTF-8.

Analizador Unicode

En versiones de ASE anteriores a 12.5, se tenía que definir el juego de caracteres predeterminado en UTF-8 para poder usar los tipos de dato unichar y univarchar.

Esta restricción fue eliminada en la versión 12.5. Ahora se puede usar tipos de dato unichar y univarchar con cualquier juego de caracteres predeterminado sin tener que primero configurar su juego de caracteres predeterminado a UTF-8.

Al incorporar la sintaxis del nuevo estándar SQL2002, se puede usar secuencias de escape para especificar cualquier caracter Unicode en un literal tipo string. Si un literal tipos string contiene caracteres que no pueden ser representados con el juego de caracteres predeterminado del servidor, el literal es promovido de tipo de dato varchar a tipo de dato univarchar.

[10] *American Standard Code for Information Interchange*, creado en 1963, es un código de caracteres basado en el alfabeto latino, tal como se usa en inglés moderno y en otras lenguas occidentales

5.7.2.7. Tablas SQL Derivadas

Una tabla derivada es creada con una sentencia select anidada, como en el siguiente ejemplo:

```
SELECT * FROM (SELECT * FROM table_1) derived_table_1
```

Un grupo equivalente de sentencias SQL que usan una vista en vez de una tabla derivada requiere tanto la sentencia create como la sentencia drop:

```
CREATE VIEW view_1 AS SELECT * FROM table_1
SELECT * FROM view_1
DROP VIEW view_1
```

El grupo de sentencias SQL que usan una vista es más complicado, requiriendo tres sentencias. Así mismo, los beneficios de crear una vista para sólo una consulta, superan el costo adicional asociado a crear una entrada en los catálogos del sistema. Las tablas derivadas eliminan este costo adicional permitiendo que las consultas espontáneamente creen tablas no persistentes sin la necesidad de borrar tablas o realizar inserciones en los catálogos del sistema. Esto hace que una tabla derivada sea más deseable que un grupo de sentencias SQL que utilicen una vista para consultas ad-hoc. Para consultas repetitivas, una tabla derivada usada múltiples veces se comporta, desde el punto de vista de rendimiento, de manera comparable a una sentencia SQL que use una vista con una definición en caché. Una tabla derivada difiere de una tabla temporal en que la tabla derivada existe sólo por la duración de la consulta, mientras que una tabla temporal existe hasta que el servidor es reiniciado.

Una entrada en los catálogos del sistema ocurre para una vista que use una tabla derivada:

```
CREATE VIEW view_1 AS
SELECT column_1 FROM
(SELECT column_1 FROM table_1) derived_table_1
```

Esto también es cierto para un procedimiento almacenado que utilice una tabla derivada:

```
CREATE PROC sp_foo @name VARCHAR(40) AS
SELECT column_1 FROM
(SELECT column_1 FROM table_1) derived_table_1
```

5.7.2.8. Tipos de Datos *Date* y *Time*

ASE tiene varias maneras de identificar fechas y horas. Antes de la versión 12.5, sólo los tipos de datos datetime y smalldatetime estaban disponibles. A partir de la versión 12.5, los tipos de datos date y time fueron agregados como tipos de datos separados.

Los tipos de datos date y time soportan los siguientes rangos:

- date de enero 1, 0001 a diciembre 31, 9999
- time de 12:00:00:000 AM a 11:59:59.999 PM

5.7.2.9. Asociación de datos SQL a XML

La cláusula for xml de la sentencia select y la función forxmlj asocian resultados SQL a documentos SQLX-XML, usando el formato SQLX-XML definido por el estándar ANSI SQLX. Por ejemplo:

```
SELECT name, type FROM systypes WHERE name LIKE "%var%"
for xml
```

```
<resultset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<row>
        <name>nvarchar</name>
        <type>39</type>
</row>
<row>
        <name>univarchar</name>
        <type>155</type>
</row>
<row>
        <name>varbinary</name>
        <type>37</type>
</row>
<row>
        <name>varchar</name>
        <type>39</type>
</row>
</resultset>
```

5.7.2.10. Procesador Nativo de XML

El procesador nativo de XML soporta extensiones SQL para llevar a cabo funciones de consulta sobre documentos XML. El procesador nativo XML puede llevar a cabo consultas sobre documentos XML almacenados y sobre documentos XML asociados a resultados SQL con la cláusula for xml o la función forxmlj.

La integración de un procesador nativo de XML en ASE 12.5 brinda mejoras radicales en rendimiento sobre el procesador XQL basado en Java de versiones anteriores de ASE.

El procesador nativo de XML soporta documento XML estándares y consultas XPath estándares, que son un subconjunto del nuevo lenguaje XQuery.

Las extensiones SQL soportadas por el procesador nativo de XML son:

- xmlextract - una función del sistema que aplica una expresión de consulta XML a un documento XML y retorna el resultado del select.
- xmltest - un predicado SQL que aplica una expresión de consulta XML a un documento XML a retorna un resultado booleano.
- xmlparse - una función del sistema que analiza un documento XML para un procesamiento más eficaz.
- xmlrepresentation - una función que determina si una columna tipo image contiene un documento XML analizado.

5.7.2.11. Autenticación de Usuarios con LDAP

LDAP es un estándar de la industria para el acceso a servicios de directorio a través de una red. Con la versión 12.5, ASE extiende su soporte a LDAP para incluir el almacenamiento de información de usuarios.

Con los servicios LDAP habilitados:

- ASE autentica clientes con datos del servidor LDAP. Los usuarios son autenticados con contraseñas almacenadas en el servidor LDAP, y no en la tabla syslogins. El servidor LDAP brinda una ubicación centralizada para las cuentas de los usuarios-tanto nombres como contraseñas.
- Los servidores ASE comparten la información de los usuarios almacenada en el servidor LDAP. La información que antes se almacenaba en syslogins ahora será manipulada y almacenada en un servidor LDAP. Está información será ubicada en cachés locales para preservar la integridad referencial y para otros usos, específicos de la base de datos.
- Con LDAP habilitado, los usuarios tienen un único nombre de usuario y contraseña en la empresa.

El soporte de LDAP en ASE requiere la licencia ASE_DIRS. Para definir nuevas cuentas de usuario, se debe:

- Agregar las cuentas de usuario al servidor LDAP.
- Construir un URL de LDAP con la cadena de búsqueda para llevar a cabo la autenticación con ASE usando sp_ldapadmin.
- Definir el parámetro de configuración enable ldap user auth para autorizar el uso de LDAP.

Después de que una cuenta se agrega al servidor LDAP, ASE puede modificar características locales de esa cuenta. Un SA o un SSO puede agregar una fila en syslogins usando sp_addlogin para definir valores específicos del login, como las base de datos predeterminada o asociarle roles.

Para asistir en la migración de cuentas de usuario al servidor LDAP, el parámetro enable ldap user auth provee opciones que permiten autenticación a través del servidor LDAP o con syslogins.

5.7.2.12. Agente HA sobre VCS 3.5

ASE 12.5.1 incorpora un nuevo agente HA que agrega soporte para esquemas de alta disponibilidad activoactivo en Veritas Cluster 3.5.

El nuevo agente tiene estos componentes:

- Archivo de definición de recursos
- Agente binario con función de monitoreo
- Scripts para el agente
- Herramienta de instalación del agente

El nuevo agente HA brinda varios beneficios:

- Brinda una solución completa e independiente y requiere solo de la licencia HA de Sybase. El Enterprise Agent para Sybase, desarrollado por Veritas, no es necesario.
- Es fácilmente instalado usando la herramienta de instalación incluida.
- Elimina la necesidad de sobrescribir scripts de Enterprise Agent para productos Sybase.
- Está estrechamente integrado con ASE.

5.7.2.13. Recuperación Rápida

Durante el reinicio del servidor después de una bajada planeada o no planeada, o durante un failover HA, una cantidad significativa de tiempo puede ser gastada en la recuperación de las bases de datos. La característica de recuperación rápida permite que las bases de datos estén disponibles más temprano y minimiza el tiempo fuera de línea. Puede ayudar a la recuperación después de bajadas planeadas, no planeadas y failover HA.

La recuperación rápida introduce los siguientes cambios en ASE:

- Pone en línea lo engines más temprano después de que todas las bases de datos del sistema han sido recuperadas.
- Mejora el rendimiento de la recuperación de las bases de datos.
- Recupera múltiples bases de datos en paralelo haciendo uso de los recursos disponibles en el servidor. Esto es controlado por el parámetro de configuración max concurrently recovered db.
- Permite que se ejecuten múltiples tareas de checkpoint en tiempo de ejecución que pueden correr concurrentemente para minimizar el trabajo que se requiere ser llevado a cabo en el momento de la recuperación. Esto es controlado por el parámetro de configuración number of checkpoint tasks.
- La nueva variable global @@recovery_state determina si ASE está en recuperación.

- Se introdujo la nueva opción strict | relax al procedimiento almacenado sp_dbrecovery_order.

5.7.2.14. Comandos mount unmount database

El propósito de los comandos mount y unmount es hacer posible transportar bases de datos de un servidor ASE origen a un servidor ASE destino.

Para mover una base de datos

Una base de datos puede ser movida entre servidores ASE "desmontando" la base de datos del primer ASE y "montándola" sobre el segundo ASE. Esta operación es llevada a cabo a través de los comandos mount database y unmount database.

El comando unmount database desvincula la base de datos y sus dispositivos de ASE. Las descripciones de la base de datos y sus dispositivos son escritas a un archivo de manifiesto, especificado por el usuario. La base de datos y sus dispositivos no serán accesibles desde ese ASE de ahí en adelante, pero los datos permanecerán intactos sobre los dispositivos de datos. Este es un comando dinámico, y puede ser ejecutado aun cuando la base de datos esté en uso. Todos los clientes que estén usando la base de datos desmontada serán desconectados. Hasta ocho bases de datos pueden ser desmontadas con un solo comando unmount database.

Para copiar una base de batos

Una base de datos puede ser copiada de un ASE a otro usando los comandos quiesce database y mount database.

El comando quiesce database ha sido extendido con una opción que crea un archivo de manifiesto para la base de datos que se va a copiar. Cuando el archivo de manifiesto es especificado en el comando quiesce database hold, ASE creará un archivo de manifiesto, en adición a llevar a cabo la operación de quiesce sobre la base de datos. Una copia de los dispositivos de la base de datos puede ser llevada a cabo con comandos del sistema operativo (por ejemplo dd en Unix), mientras que la base de datos permanece bloqueada. La copia de los dispositivos de la base de datos puede ser luego montada sobre un segundo ASE usando el archivo de manifiesto.

5.7.2.15. Programación de tareas (Job Scheduler)

El Job Scheduler de ASE facilita la administración de ASE brindando la habilidad de definir y programar tareas de administración de base de datos. Con el Job Scheduler, las tareas que normalmente requieren interacción de un administrador de la base de datos pueden ser programadas para ser ejecutadas a horas apropiadas, permitiendo que el administrador atienda otros asuntos.

El Job Scheduler le permite crear y programar tareas y también compartir tareas y programaciones. Un administrador de base de datos puede crear una tarea y otro

administrador de base de datos puede luego programar y ejecutar esa tarea en otro servidor. Las tareas pueden ser creadas de cero, con línea de comandos o desde Sybase Central, cargadas a partir de un archivo batch SQL o generadas a partir de plantillas predefinidas.

El Job Scheduler captura los resultados y salida de las tareas y registra esa información en tablas de log. El Job Scheduler manutiene una historia de las tareas programadas. Sin embargo, para mantener un límite en el tamaño de la tabla histórica, el Job Scheduler mismo lleva a cabo un monitoreo y remueve registros viejo innecesarios.

El Job Scheduler consta de los siguientes componentes:

- Una tarea interna de ASE.
- Un proceso externo llamado el Agente Job Scheduler.
- La base de datos sybmgmtdb y procedimientos almacenados.
- La interfaz gráfica de usuario.
- Plantiíllas predefinidas a partir de las cuales el administrador de la base de datos puede crear y programar tareas.

La tarea interna de ASE determina cuándo las tareas programadas deben ejecutarse y crea un registro histórico de tareas que son ejecutadas. Arranca el proceso del Agente Job Scheduler y lo alimenta con la información necesaria para recuperar información de tareas y ejecutar la tarea en el ASE especificado.

El Agente Job Scheduler recupera información de la base de datos del Job Scheduler, llamada sybmgmtdb. Luego la registra en el ASE destino y ejecuta los comandos de la tarea. Cuando la tarea se completa, el Agente Job Scheduler registra cualquier resultado o salida en tablas de log de la base de datos sybmgmtdb.

Toda la información de tareas, programación, tareas programadas y datos necesaria para el procesamiento interno de la tarea Job Scheduler es almacenada en la base de datos sybmgmtdb. La mayor parte del acceso a datos de la base de datos sybmgmtdb es llevada a cabo vía procedimientos almacenados. Los procedimientos almacenados hacen que la información esté disponible para la interfaz gráfica, el Agente Job Scheduler y la interfaz de línea de comandos. Sólo la tarea Job Scheduler tiene acceso directo a los datos de la base de datos sybmgmtdb.

La interfaz gráfica asiste al usuario en la creación y programación de tareas, visualización del estado de las tareas y de la historia de las tareas, y en el control de las tareas. Las interfaz gráfica también provee una característica de administración para habilitar o deshabilitar la tarea interna de ASE y en consecuencia la habilidad del Job Scheduler de ejecutar tareas programadas.

Las plantillas son una herramienta importante al definir tareas para la auto-administración de la base de datos, tales como copias de respaldo de las bases de datos, reorganización, modificación de parámetros de configuración, actualización de estadísticas y monitoreo. Ellas

están implementadas como comandos Transact-SQL en batch para los cuales se dan valores paramétricos. Los administradores de la base de datos pueden usar las plantillas para generar tareas, que pueden ser programadas para ser ejecutadas en horas específicas.

5.7.2.16. Nueva Sección de Asistente de Caches en sp_sysmon

El sp_sysmon para ASE 12.5.1 ha sido mejorado. El nuevo parámetro cache wizard ayuda en el monitoreo y configuración de cachés de datos para un mejor rendimiento. El asistente para cachés le permite identificar:

- Objetos "críticos" (objetos que son frecuentemente utilizados). La salida es calificada por el númerode lecturas lógicas en el cahcé con nombre o en el default data cache.
- El uso de cachés y pools de buffers.
- El porcentaje de aciertos en un caché, pool de buffers y a nivel de objeto.
- La efectividad de operaciones de lectura/escritura de tamaño grande.
- La efectividad del APF (Asynchronous Prefetch).
- El nivel de ocupación de cachés por parte de diferentes objetos.

La sección del asistente de cachés aparece en la salida del sp_sysmon sólo cuando usted incluye el parámetro cache wizard. Usted puede incluir dos parámetros con cache wizard, topN y filter. topN califica los objetos e imprime el reporte y filter imprime l0os nombres de los cachés que contengan este patrón.

5.7.2.17. Servicios Web para ASE

Un servicio Web es una aplicación auto-contenida y modular a la cual se puede tener acceso y puede ser usada sobre una conexión de red usando los estándares abiertos SOAP, WSDL y XML. Los Servicios Web de ASE permiten que ASE provea y use servicios Web.

Los servicios Web de ASE consisten de componente Productor, el cual permite que aplicaciones cliente tengan acceso a SQL y procedimientos almacenados en ASE usando SOAP.

5.7.2.18. Mayor Compatibilidad con las Extensiones de Microsoft SQL

ASE 12.5 brinda mayor compatibilidad con las extensiones de Microsoft SQL.

Las nuevas características le permiten:

- Usar default en el INSERT. Por ejemplo,

```
CREATE TABLE test table(col1 tinyint, col2 tinyint default 10, col3
tinyint)
INSERT test_table VALUES(11, default, 43)
```

- Usar set para asignar valores a variables. Por ejemplo:

```
DECLARE @a tinyint
SET @a = 10
```

- Usar paréntesis cuadrados alrededor de identificadores. Por ejemplo:

```
CREATE TABLE [mytable] (a tinyint)
```

- Usar las nueva función del sistema cast() y aquellas agregadas en versión 12.5.0.3: len(), left(), day(), month(), year(), str_replace(), newid(), square().
- Usar tablas derivadas, las cuales pueden ser usadas en donde esté permitido el uso de una vista y las cuales pueden ser usadas en la lista from de las sentencias select, select into, create view e insert.
- Migrar más fácilmente de MS-SQL Server a Sybase ASE.

5.7.3. Ediciones

En la actualidad Sybase ofrece ASE en cinco ediciones diferentes:

- **ASE Enterprise Edition.-** No tiene límites desde el punto de vista de escalabilidad y puede correr todas las opciones que se adquieren por separado.
- **ASE Small Business Edition.-** Tiene algunos límites en escalabilidad y puede correr un conjunto limitado de las opciones que se adquieren por separado.
- **ASE Developer's Edition.-** Tiene límites de escalabilidad e incluye un número mínimo de opciones, edición gratuita para desarrolladores.
- **ASE Express Edition para Linux.-** Tiene algunos límites de escalabilidad y almacenamiento, pero se puede usar libremente para desarrollo y producción.
- **ASE Cluster Edition.-** Permite aprovechar los recursos computacionales, brindando escalabilidad horizontal en clusters de hasta 32 nodos. A la fecha, ASE Cluster Edition está disponible sólo para las plataformas Linux y Sun Solaris.

PAQUETES	Enterprise Edition	Small Business Edition	Developer Edition	Express Edition
Security and directory services	Opcional	No disponible	Incluido	No disponible
High Availability	Opcional	No disponible	Incluido	No disponible
Distributed Transaction Management (DTM)	Opcional	No disponible	Incluido	No disponible
SQL Expert	Opcional	Opcional	No disponible	No disponible
BMC DBXray	Opcional	Opcional	No disponible	No disponible
Disaster Recovery option	Opcional	No disponible	No disponible	No disponible
Partitions	Opcional	No disponible	Incluido	No disponible
Enhanced Full Text Search	Opcional	Opcional	No disponible	No disponible
LIMITES				
Número de usuarios:	Ilimitado	256	25	Ilimitado
Número de CPUs:	Ilimitada	4	1	1
Capacidad máxima de almacenamiento:	Ilimitada	Ilimitada	Ilimitada	5 GB
Memoria máxima:	Ilimitada	Ilimitada	Ilimitada	2 GB

Fuente: [L01]

Tabla 5.2. Componentes disponibles para las ediciones de ASE y algunos de los límites de escalabilidad.

[L01] *Introducción a las bases de datos*, Rafael Camps Paré.

5.7.4. Plataformas soportadas

ASE está soportado para la mayoría de plataformas comerciales, incluyendo:

- Windows
- Linux
- Solaris
- IBM AIX
- HP-UX
- Mac OS
- Silicon Graphics IRIX
- Silicon Graphics IRIX2

5.7.5. Instalación y configuración de Sybase 12.5

5.7.5.1. Roles de Usuario

El proceso de instalación y configuración de Adaptive Server define varios roles de usuario. Cada uno de los roles de usuario está asociado a distintas responsabilidades y privilegios. Estos roles de usuario determinan como el servidor de Adaptive Server se integra en el sistema.

- **Administrador del sistema operativo:** usuario responsable del mantenimiento del sistema operativo. Este tipo de usuario tiene privilegios de superusuario o "root".
- **Administrador del sistema Sybase:** usuario responsable de la Administración del sistema del servidor Adaptive Server, creación de cuentas de usuario, asignación de permisos en las bases de datos y creación de bases de datos nuevas. En el proceso de instalación, el nombre de inicio de sesión del Administrador del sistema es "sa". Este nombre ("sa") no es válido para el inicio de sesión en UNIX. El nombre de inicio de sesión "sa" es específico del servidor de Adaptive Servery se utiliza para iniciar en éste la sesión mediante el comando isql.
- **Inicio de sesión Sybase:** el nombre de inicio de sesión "sybase" es válido para UNIX y tiene privilegios sobre todos los archivos y directrorios de instalación de Sybase, define los permisos en esos directorios y archivos, y realiza la instalación y actualización del servidor Adaptive Server.

5.7.5.2. Descripción de productos

PRODUCTO	DESCRIPCION
Adaptive Server	Servidor de base de datos relacional. El proceso de descarga predeterminado incluye: ▪ Adaptive Server ▪ Backup Server ▪ Monitor Server ▪ XP Server ▪ Utilidades de Adaptive Server ▪ Archivos de secuencias de comandos y de configuración
Plugin de Adaptive Server para Sybase Central Java Edition	Sybase central es un sistema de marco común para la administración de servidores. Permite administrar las instalaciones de Adaptive Server mediante la herramienta gráfica de administración de Sybase Central.

Sybase Monitor Server para Adaptive Server	Aplicación Open Server que obtiene estadísticas de rendimiento sobre el servidor Adaptive Server y que las pone a disposición de las aplicaciones cliente Monitor Server. Sybase Monitor Server incluye: ▪ **Monitor Server para Adaptive Server Enterprise 12.5:** Aplicación Open Server que obtiene estadísticas de rendimiento del servidor Adaptive Server que se pueden consultar con los monitores Monitor Historical Server y las aplicaciones creadas con la biblioteca Monitor Client-Library. ▪ **Monitor Client-Library:** interfaz de programación que proporciona acceso a los datos de rendimiento de Adaptive Server. ▪ **Monitor Historical Server:** aplicación Open Server que obtiene estadísticas de rendimiento de varios servidores Adaptive Server a través de servidores Monitor Server y que registra los datos en las ubicaciones de archivo especificadas.
Backup Server	Backup Server es una aplicación basada en Open Server que gestiona todas las operaciones de copia de seguridad (volcado) y restauración (carga) de las bases de datos del servidor Adaptive Server. Backup Server. ▪ Permite utilizar un número ilimitado de dispositivos de volcado (lo que se denomina volcado fragmentado) de forma paralela para el volcado o carga de un diario de transacciones o bases de datos. ▪ Permite realizar volcados comprimidos y cargas en el disco local. ▪ Permite realizar un volcado en varias cintas o transferir varios volcados a una sola cinta. ▪ Permite operaciones bidireccionales de volcado y carga en la red con un dispositivo situado en el otro equipo. ▪ Permite determinar automáticamente con comandos del sistema operativo las características de los dispositivos de cinta para una operación de volcado. ▪ Admite la especificación de sintaxis de comandos de volcado y carga para la asignación de nombres de volúmenes, el control de desmontar y cargar, la densidad de cinta, el tamaño de bloques, la asignación de nombres de archivos a volúmenes de varios volcados y la presentación de información de archivos de cabecera. Instalar Backup Server si se tiene previsto realizar una copia de seguridad y restaurar las bases de datos de Adaptive Server. Backup Server se instala de forma predeterminada a instalar el software del servidor de Adaptive Server.
Módulos de Idioma (Servidor)	Proporciona mensajes del sistema y formatos de fecha y hora útiles para localizar las aplicaciones. La instalación predeterminada incluye el módulo de idioma us_english y los siguientes juegos de caracteres: ▪ cp437: IBM CP437, conjunto de códigos EE.UU ▪ cp850: IBM CP850, conjunto de códigos para Europa ▪ iso_1: 8859-1, Latin-1 ▪ mac: Códigos estándar para Macintosh ▪ roman8: HP Roman-8 Los productos cliente de Adaptive Server y Sybase están también disponibles en francés, alemán y japonés. Hay también módulos de idioma en español, coreano, portugués brasileño y chino simplificado, pero sólo para Adaptive Server.
Módulos de idioma (Conectividad)	Proporciona mensaje y archivos de compatibilidad para la ejecución de aplicaciones Open Client en varios idiomas.
jConnect 4.5 y j Connect 5.5	Proporcionan un controlador de conectividad de bases de datos Java (JDBC) que funciona en máquinas virtuales (VM) Sun y Microsoft. Proporcionan compatibilidad para límites extendidos de Adaptive Server 12.5 al solicitar compatibilidad para tablas más anchas. Esta petición sólo la admiten servidores de Adaptive Server 12.5 y versiones superiores. Para obtener más información sobre jConnect para JDBC, consulte la página del producto jConnect en http://www.sybase.com/support/manuals/.
Documentación de jConnect	Contiene el manual Sybase jConnect for JDBC Programmers Reference.

Utilidades Java	Incluye: ▪ **Cascade Gateway:** pasarela que actúa como proxy para proporcionar una ruta al servidor de bases de datos si se ejecuta en un servidor distinto al servicio web. ▪ **Jisql:** editor gráfico Transact-SQL desarrollado en Java y que sustituye a SQL Advantage. ▪ **Ribo:** utilidad que captura, convierte y muestra los flujos del protocolo TDS (Tabular Data Stream, Flujo de datos en tablas) entre un cliente TDS y un servidor TDS.
Controlador ODBC	Permite a las aplicaciones cliente Windows NT tener acceso a los datos del servidor Adaptive Server.
Open Client	Contiene bibliotecas y utilidades para desarrollar aplicaciones Open Client.
XP Server	Aplicación Open Server que gestiona y ejecuta procedimientos almacenados extendidos (ESP) desde el servidor Adaptive Server. Los procedimientos ESP proporcionan un método de llamada para las funciones de lenguaje de procedimientos desde Adaptive Server.XP Server se descarga de forma predeterminada al descargar el software del servidor de Adaptive Server desde el soporte de distribución. Use la utilidad srvbuild para configurar XP Server y para conectar XP Server y Adaptive Server mediante el archivo de interfaces.

Fuente: [L02]

Tabla 5.3. Descripción de los productos de servidor incluidos en el paquete estándar de Adaptive Server.

5.7.5.3. Directorio de instalación Sybase

La estructura del directorio de instalación de Sybase se crea con el proceso de instalación. Adaptive Server se instala en el directorio indicado durante la ejecución de Studio Installer. En la siguiente tabla se muestra los directorios de instalación a los que probablemente se tendrá acceso durante la instalación, configuración y administración del servidor de Adaptive Server.

El directorio de instalación del servidor de Adaptive Server contiene los archivos ejecutables y las herramientas de administración que se añadieron al instalar los productos.

DIRECTORIO DE COMPONENTES	SUBDIRECTORIO	SUBDIRECTORIO
/ASE-12_5	/bin	
	/certificates	
	/debugger	
	/include	
	/init	/auditinit
		/bsrv
		/logs
		/sqlsrv
	/install	/spr
	/lib	
	/sample	/esp
		/JavaSql
		/server
	/scripts	
	/sybhelp	
	/upgrade	
	/xappdefaults	
/ASEP-1_0		
/CFG-10		
/charsets		

[L02] Documentación de Adaptive Server Enterprise 12.5, Sun Solaris, Manual de instalación.

/collate	/Unicode	
/configed		
/docs-45_55		
/EFTS-12_5		
/EJB-12_5		
/HOST-1_0		
/installed		
/Installer	/bin	
	/lib	
	/classes	/gateway2
		/sample2
	/devclasses	
	/docs	/en
	/gateway2	
	/sample2	
	/sp	
	/tools	
	/cascade	/uk
/jutils-2_0	/jisql	/doc
		/HelpFiles
	/ribo	/doc
/locales	/us_english	
	/message	
	/unicode	
/OCS-12_5	/bin	
	/config	
	/devblib	
	/include	
	/lib	
	/lib3p(/lib3p64 para plataformas de 64 bits)	
	/locales	
	/sample	
	/sybhelp	
/shared-1_0		
/SQLRemote	/bin	
	/scripts	
/Sybcent32		
/SYSAM-1_0	/bin	
	/licenses	

Fuente: [L02]

Tabla 5.4. Directorio de instalación para Sybase

5.7.5.4. Instalación de Adaptive Server

Métodos de instalación

Se puede utilizar cualquiera de los siguientes métodos para instalar servidores de Sybase:

- **Studio Installer:** utilizar este programa para instalar servidores y adaptarlos a un entorno de producción. La adaptación de un servidor durante la instalación reduce la necesidad de modificaciones posteriores. Se utiliza este método para gestionar la licencia de las funciones opcionales de Adaptive Server durante el proceso de instalación.
- **Archivo de recursos:** se utiliza Studio Installer para instalar los componentes del servidor en el disco duro y, a continuación, se ejecuta la utilidad srvbuildres para instalar y configurar Adaptive Server y Backup Server en otras ubicaciones que

[L02] Documentación de Adaptive Server Enterprise 12.5, Sun Solaris, Manual de instalación.

requieran servidores idénticos. El archivo de recursos también permite realizar la instalación de entornos donde no está disponible X-Windows. No puede instalar Monitor Server y XP Server mediante archivos de recursos.

- **La utilidad de instalación *asecfg*:** ubicada en el directorio *$SYBASE/$SYBASE_ASE/bin* sirve para configurar un servidor nuevo, actualizar un servidor existente y modificar el idioma predeterminado del servidor.

Instalación de componentes con Studio Installer

Studio Installer crea un directorio de destino (si es necesario) e instala todos los componentes seleccionados en dicho directorio.

Puede verificar la instalación de los productos al final de la misma. Puede que tenga que llevar a cabo procedimientos de configuración adicionales para utilizar algunos productos.

Durante el proceso de instalación, Studio Installer define la mayoría de las variables de entorno necesarias para los productos Adaptive Server. No obstante, debe definir otras variables de entorno mediante la ejecución del archivo de procedimiento de comandos *SYBASE.csh* o *SYBASE.sh*, cuando haya salido de Studio Installer.

Para instalar componentes del servidor:

1. Inserte el CD del servidor en la unidad de CD-ROM
2. El sistema operativo Solaris monta el CD automáticamente. Haga doble clic en el icono Instalar para iniciar el proceso de instalación.

   Si Studio Installer no se inicia automáticamente, escriba:

   ```
   cd /cdrom/sybasecd
   ./install
   ```

   Donde *cdrom* corresponde al directorio especificado al montar la unidad de CD-ROM e *./install* inicia Studio Installer.

3. Seleccione el tipo de instalación que desea realizar.
   - **Standard Install (Instalación estándar):** instala los componentes predeterminados que los usuarios necesitan.
   - **Full Install (Instalación Completa):** instala todos los componentes del CD.
   - **Customized Install (Instalación Personalizada):** permite seleccionar los componentes que se van a instalar. Algunos componentes se instalan automáticamente si son necesarios para ejecutar otros componentes seleccionados.

   Los servidores Backup Server, Monitor Server y XP Server se instalan de forma predeterminada con Adaptive Server.

Si no selecciona ningún componente para instalación, Studio Installer genera un mensaje de error y detiene la instalación.

Haga clic en atrás para seleccionar componentes o en Cancelar para cancelar el proceso de instalación.

4. Haga clic en siguiente.
5. Especifique el directorio de destino y haga clic en siguiente para continuar.
   Si selecciona Instalación personalizada, la siguiente ventana es la pantalla de selección de componentes, donde puede especificar que componentes desea instalar.
   Los componentes de la instalación estándar tienen marcada la casilla de verificación que aparece a la izquierda del nombre del producto. Puede seleccionar o anular la selección de los componentes de esta lista. Los componentes formados por subcomponentes tienen activado el botón Más. Haga clic en este botón para seleccionar o anular la selección de subcomponentes.
6. La pantalla de resumen muestra todos los componentes que va a instalar Studio Installer, el espacio en disco necesario y el espacio en disco disponible.

   Si el directorio de destino no dispone de espacio suficiente, el espacio disponible aparece en rojo. Si pulsa Siguiente y no hay espacio suficiente en el disco duro, se producirá un error y la instalación se detendrá.

   Haga clic en Siguiente.

7. Si no existe el directorio de destino, Studio Installer le pedirá q lo genere.

   Haga clic en Sí para continuar.

   También puede seleccionar Guardar en la pantalla de resumen para guardar toda la información de la instalación en un archivo *cdmfile*, y realizar la instalación de forma silenciosa y no interactiva.

   A continuación, Studio Installer instala los componentes en el disco duro y muestra un indicador de progreso.

Si se interrumpe el proceso de instalación, se deberá eliminar manualmente todos los archivos de Adaptive Server y los archivos relacionados y reiniciar la instalación de un entorno nuevo.

Configuración de Servidores.

Studio Installer define todas las variables de entorno necesarias para configurar los servidores de Sybase. Si se producen errores en el entorno, Studio Installer genera archivos de procedimientos de comandos que puede ejecutar para corregir los errores después de la Instalación. Si fuera necesario corregir el entorno, ejecutar los siguientes procedimientos de comandos como usuario "root".

- Si se utiliza el *sheell Bourne*, ejecute:

```
$ .SYBASE.sh
```

- Si utiliza el *Shell C*, ejecute:

```
% source SYBASE.csh
```

Cuando se inicie la sesión como usuario "sybase" por primera vez después de la instalación, comprobar que la variable de entorno SYBASE se ha definido correctamente y que *$SYBASE/$SYBASE_ASE/bin* y *$SYBASE/$SYBASE_OCS/bin* forman parte de la búsqueda.

CAPÍTULO VI

DISEÑO E IMPLEMENTACIÓN DEL SISTEMA DE ADMINISTRACIÓN Y CONTROL DE RECURSOS TECNOLÓGICOS PARA LA COAC "ATUNTAQUI" LTDA.

INTRODUCCIÓN

METODOLOGÍA

REQUISITOS TECNOLÓGICOS

PLAN DE DESARROLLO

ANÁLISIS DEL PROYECTO

DISEÑO DEL PROYECTO

IMPLEMENTACIÓN DEL PROYECTO

6.1. INTRODUCCIÓN

Actualmente, las instituciones financieras brindan una gran variedad de servicios a sus socios y clientes, mismos que deben estar disponibles las veinte y cuatro horas del día; convirtiendo a la informática en una herramienta indispensable para la disponibilidad de estos servicios. Implicando que inviertan en una importante cantidad de recursos informáticos, para de esta manera mantener actualizado su parque tecnológico y estar siempre a la vanguardia en la atención a sus socios. Razón por la cual, estas instituciones deben tener un adecuado proceso de administración y control de los recursos, de manera que prolonge su vida útil y disminuya los costos de mantenimiento.

La Cooperativa de Ahorro y Crédito "Atuntaqui" Ltda., es una institución financiera ubicada en la provincia de Imbabura y al norte de Pichincha, ofreciendo a la comunidad una variedad de productos y servicios. Con la finalidad de brindar una adecuada atención a sus socios y clientes, el Departamento Informático de la Cooperativa Atuntaqui Ltda., constantemente actualiza su tecnología, provocando una variedad de cambios y modificaciones en los recursos tecnológicos.

Durante mi permanencia laboral y profesional en la Cooperativa de Ahorro y Crédito "Atuntaqui" Ltda., he podido constatar que en la institución existe una variedad y cantidad considerable de recursos tecnológicos que facilitan el desempeño de las actividades cotidianas de sus empleados; cuya administración y control de forma manual resulta tediosa y compleja, razón por la cual el departamento informático no cuenta con la información clara, precisa y oportuna, complicando de esta manera la obtención de datos.

Es por esta razón que, en colaboración con el área informática de la Cooperativa de Ahorro y Crédito "Atuntaqui" Ltda., se ha decidido plantear el presente proyecto con la implementación de una aplicación web que optimice el proceso de administración y control de los recursos tecnológicos existentes.

6.1.1. Descripción del problema

En la actualidad, la informática cumple un rol fundamental en el desarrollo de nuestras actividades diaras, desde las actividades más simples hasta las más complejas necesitan de la informática; es así como, desde el envío de una carta o hasta la administración de un sistema financiero están relacionados con la informática. La informática nos brinda un sin número de ventajas, tales como: disminución de tiempo y trabajo, minimización de errores, automatización de tareas repetitivas, almacenamiento de gran cantidad de información, entre otras.

A todo esto se suma que durante los últimos años las aplicaciones web han tenido una mejor acogida por parte de los usuarios, no solo por la interfaz a la que están familiarizados; sino también a la posibilidad que nos brindan de poder subir nuestra aplicación al internet y poder acceder a esta desde cualquier parte del mundo. Pero existen inconvenientes en el desarrollo de estas aplicaciones cuando se necesita visualizar e imprimir la información de nuestro sistema de una manera específica y/o gráfica, debiendo los desarrolladores invertir gran parte de su tiempo en la generación de estos informes o a su vez adquirir alguna herramienta de desarrollo que facilite su implementación incrementando el costo del proyecto.

Una de las tecnologías más sustentables en el desarrollo de aplicaciones web es JSP, pues esta nos brinda una variedad de alternativas al momento de diseñar nuevos proyectos. Pero, ¿Existen herramientas OpenSource que nos ayuden con la generación de reportes en nuestras aplicaciones web con tecnología JSP?, la respuesta tal vez es obvia y es que si existen estas herramientas, una de ellas y tal vez la más robusta es JasperReports; el inconveniente surge cuando no existe la documentación necesaria para la utilización de esta herramienta en nuestros proyectos, pues la documentación existente se limita a explicar su utilización de forma general con ejemplos básicos que no satisfacen la necesidad de los programadores, viéndose relegada por estos. Es por esta razón que es necesario realizar un estudio detallado de esta herramienta para presentar una nueva alternativa a los desarrolladores de software.

Por otra parte, podemos decir que en la actualidad en todas las empresas e instituciones está presente la informática, quienes se ven en la necesidad de actualizar su tecnología constantemente para garantizar su permanencia y crecimiento en el mercado competitivo de hoy en día; pero en la mayoría de estas empresas no existe un proceso adecuado de administración y control de los recursos tecnológicos, debido muchas veces a la falta de recursos económicos o que los mismos se invierten totalmente en el negocio, sin considerar posibles eventualidades. Es por ello que en las empresas e instituciones que invierten en la adquisición de nueva tecnología, es indispensable y necesario contar con una herramienta informática de apoyo que permita realizar la administración de los recursos tecnológicos de manera adecuada, eficaz y eficiente, optimizando recursos y disponiendo de información actualizada que contribuya al normal desarrollo de las actividades.

6.2. METODOLOGÍA

La metodología a utilizarse en el desarrollo del presente proyecto será la metodología RUP (*Rational Unified Process*), pues está se constituye como la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos; se

caracteriza por ser iterativa e incremental, está centrada en la arquitectura y guiada por los casos de uso. La metodología RUP constituye las etapas:

- Iniciación, incepción o concepción.
- Elaboración.
- Construcción, desarrollo o implementación.
- Transición o cierre.

6.2.1. Iniciación (Concepción)

Esta etapa tiene como propósito definir y acordar el alcance del proyecto, para lo cual es muy importante la definición de los requerimientos del negocio además de identificación de riesgos, los cuales deben ser direccionados antes de que el proyecto siga adelante. Para proyectos enfocados en mejoras de sistemas existentes, esta fase será corta pero estará enfocada en asegurar que el proyecto vale la pena su realización y se lo puede hacer.

6.2.2. Elaboración

En esta fase se establece una línea base para la arquitectura del sistema que provea estabilidad en la construcción del sistema, se realiza la especificación de los casos de uso seleccionados y el primer análisis del dominio del problema, se diseña la solución preliminar tomando en cuenta los riesgos más significativos.

6.2.3. Construcción (Implementación)

Esta etapa consiste en completar el desarrollo del sistema basado en la arquitectura definida, poniendo énfasis en la administración de recursos y controlando las operaciones optimizando costos, tiempos y calidad; este procedimiento consiste de una serie de iteraciones que culminan con la operación inicial del sistema.

6.2.4. Transición (Cierre)

El propósito de esta fase es asegurar que el software esté disponible para los usuarios finales, ajustar los errores y defectos encontrados en las pruebas de aceptación, capacitar a los usuarios y proveer el soporte técnico necesario. Se debe verificar que el producto cumpla con las especificaciones entregadas por las personas involucradas en el proyecto. En este proceso el usuario deberá centrarse en las mejoras del producto, configuración, y problemas de uso. Todos los problemas estructurales mayores se resolvieron anteriormente.

6.3. REQUISITOS TECNOLÓGICOS

Para la implementación del sistema se utilizó una variedad de herramientas de desarrollo, indispensables para la finalización de del proyecto, herramientas que se pasan a detallar a continuación con sus costos aproximados de inversión.

DESCRIPCION	COSTO ACTUAL (USD)	COSTO REAL (USD)
S O F T W A R E		
JASPERREPORTS 3.7	0.00	0.00
iREPORT 3.7	0.00	0.00
NETBEANS 7	0.00	0.00
ADAPTIVE SYBASE ENTERPRISE 12.5	5, 000.00	0.00
TOMCAT 7	0.00	0.00
H A R D W A R E		
COMPUTADOR PORTATIL	900.00	0.00
IMPRESORA A TINTA	60.00	0.00
M A T E R I A L E S		
MATERIALES DE OFICINA	100.00	100.00
IMPRESIÓN DE DOCUMENTOS	200.00	200.00
OTROS	200.00	200.00
B I B L I O G R A F Í A		
LIBROS Y REVISTAS	200.00	200.00
INTERNET	300.00	300.00
CAPACITACIÓN	500.00	500.00
S U B T O T A L	7,460.00	1,500.00
IMPREVISTOS	150.00	150.00
T O T A L	7,610.00	1,650.00

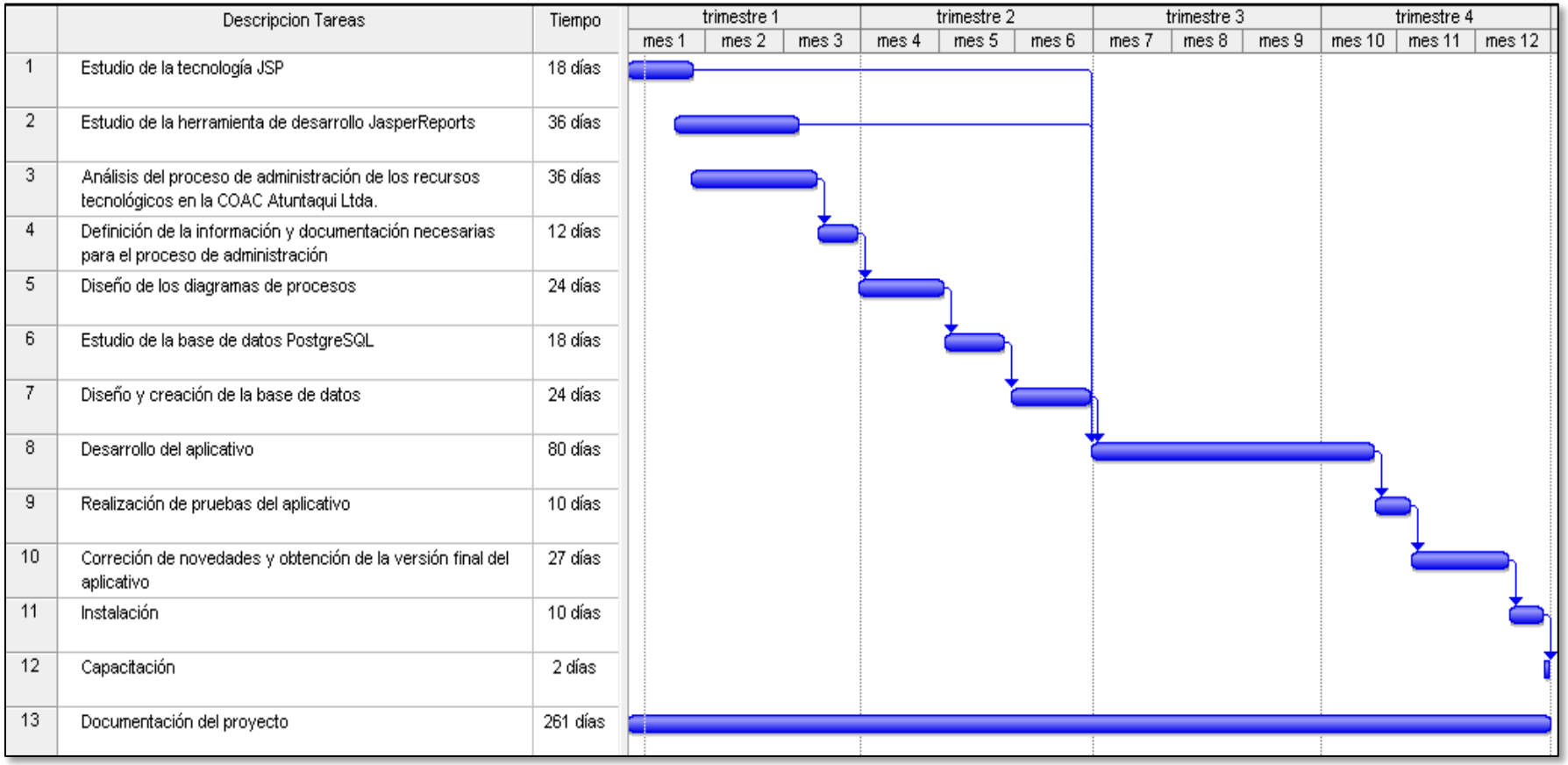
Fuente: [Propia]

Tabla 6.1. Cuadro de recursos tecnológicos y costo aproximados a utilizarse en el proyecto.

6.4. PLAN DE DESARROLLO

En este punto, se detalla las actividades a realizarse durante todo el proceso de diseño e implementación del proyecto, mismas que tendrán una duración total aproximada de un año calendario. Además, cabe indicar que el levantamiento de información y documentación de la investigación se realizará durante todo el tiempo que dure el desarrollo del proyecto.

La siguiente figura muestra el cronograma de actividades a realizarse en el presente proyecto:



	Descripcion Tareas	Tiempo
1	Estudio de la tecnología JSP	18 días
2	Estudio de la herramienta de desarrollo JasperReports	36 días
3	Análisis del proceso de administración de los recursos tecnológicos en la COAC Atuntaqui Ltda.	36 días
4	Definición de la información y documentación necesarias para el proceso de administración	12 días
5	Diseño de los diagramas de procesos	24 días
6	Estudio de la base de datos PostgreSQL	18 días
7	Diseño y creación de la base de datos	24 días
8	Desarrollo del aplicativo	80 días
9	Realización de pruebas del aplicativo	10 días
10	Correción de novedades y obtención de la versión final del aplicativo	27 días
11	Instalación	10 días
12	Capacitación	2 días
13	Documentación del proyecto	261 días

Fuente: [Propia]

Figura 6.1. Cronograma de actividades del proyecto.

6.5. ANÁLISIS DEL PROYECTO

Después de haber realizado un estudio de los procesos involucrados en la administración y control de los recursos tecnológicos de la Cooperativa de Ahorro y Crédito Atuntaqui Ltda., se ha visto conveniente clasificados, en cinco módulos principales. Mismo que fueron optimizados para una mejor administración de tecnología.

Fuente: [Propia]

Figura 6.2. Esquema general del sistema (SARTE).

6.5.1. Descripción general del sistema

A continuación se describe brevemente cada uno de los procesos involucrados:

Administración

El sistema está basado en su mayoría en parámetros establecidos por parte del administrador del sistema; mismos que pueden ser configurables a través del sistema de información. Las principales opciones del módulo son:

- Parametrización de grupos de recursos tecnológicos
- Parametrización de categorías de recursos.
- Parametrización de subcategorías de recursos.
- Parametrización de marcas de recursos.
- Parametrización de modelos de recursos.
- Parametrización de áreas.
- Parametrización de los tipos de mantenimientos.
- Parametrización de motivos de traslados de recursos.

Ingreso de datos

Una vez realizada la configuración de los parámetros, esenciales para el correcto funcionamiento del sistema, estos nos permite ingresar, recompilar y organizar la información de una variedad de recursos tecnológicos, conforme la categorización establecida por el administrador del sistema. Las opciones principales de este módulo son:

- Ingreso de datos de los recursos tecnológicos.
- Ingreso de las características principales de los recursos y establecidas en la parametrización.

Mantenimiento

Uno de las necesidades principales y motivos por el cual se planteó la implementación del sistema de administración de recursos tecnológicos, fue el poder registrar y llevar un control de todos los mantenimientos y novedades suscitadas durante el tiempo de vida útil de los recursos tecnológicos. Las actividades más trascendentales de este módulo son:

- Registrar los mantenimientos preventivos y correctivos realizados a los recursos tecnológicos.
- Ingresar detalladamente las modificaciones y alteraciones de componentes realizadas en un determinado recurso.

Traslados

Durante el estudio del proceso de control de los recursos y el proceso de recopilación de la información, pudimos determinar que existe carencia de información en lo que respecta al traslado y custodios de equipos, dificultándose el proceso de control de los mismos. Es por esto que se planteó la implementación de un módulo de traslado de equipos que nos ayuden con esta tarea. Las opciones principales del módulo son:

- Registro del cambio de custodio de los recursos
- Registro del cambio de ubicación y/u oficina de los recursos.

Reportes

La razón principal por la que se planteó la implementación de este sistema en la cooperativa es el poder disponer de información clara, precisa y oportuna de todas las novedades presentadas con los recursos tecnológicos, con el fin de conocer la realidad institucional en lo que respecta a la tecnología y poder tomar con prontitud las decisiones más acertadas dentro de la administración de estos recursos.

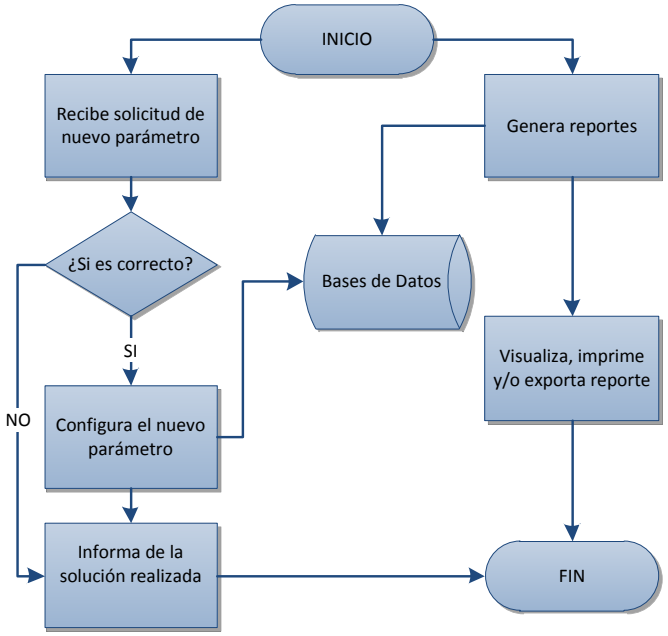
El módulo de reportes presenta una variedad de informes que pueden ser generados instantáneamente.

6.5.2. Flujo de trabajo

Durante ekl proceso de estudio e investigación del proceso de administración de la tecnología, se pudo establecer el flujo de trabajo y de información de cada uno de los módulos establecidos para el sistema. A continuación se describe cada uno de ellos:

Administración

El administrador del sistema es el encargado de realizar toda la configuración de los parámetros, de igual manera el administrador del sistema podrá generar cualquier tipo de reporte, con la finalidad de verificar la funcionalidad del sistema.

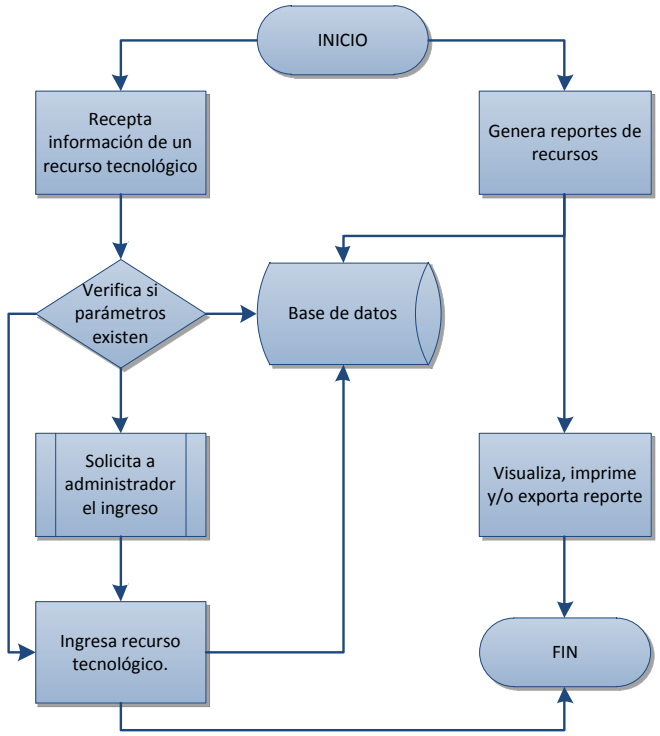


Fuente: [Propia]

Figura 6.3. Flujo de trabajo del módulo de Administración.

Ingreso de datos

El módulo de ingreso de datos, esta encargado de realizar todo el ingreso de información referente a los recursos tecnológicos, basándose en los parámetros configurados por parte del administrador.

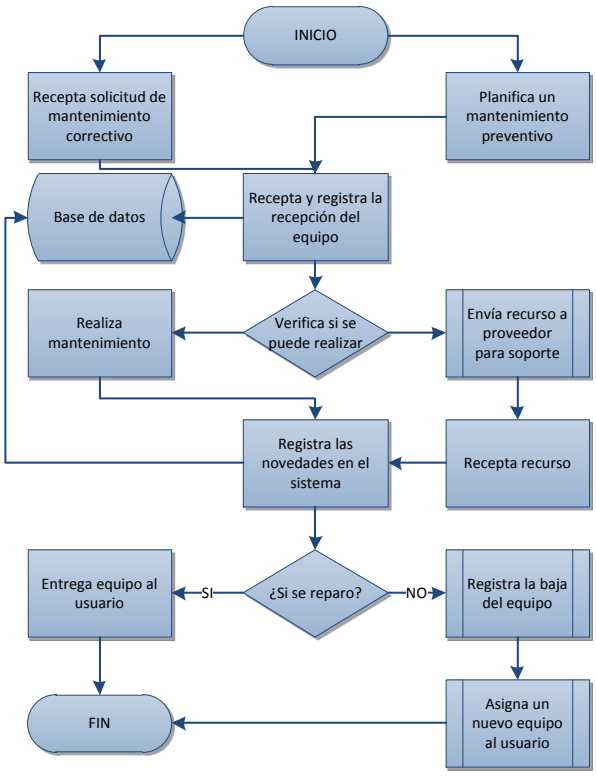


Fuente: [Propia]

Figura 6.4. Flujo de trabajo del módulo de Ingreso de datos.

Mantenimiento

El módulo de mantenimiento es el encargado de registrar todas las modificaciones y novedades surgidas con todos y cada uno de los recursos registrados en el sistema, para su control y administración de manera más eficiente.

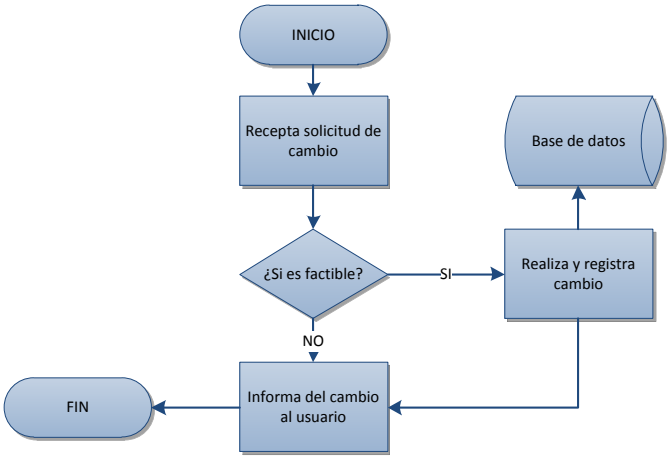


Fuente: [Propia]

Figura 6.5. Flujo de trabajo del módulo de Mantenimiento.

Traslados

En la administración de recursos tecnolçogicos es muy común realizar el cambio de custodios de los equipos, por lo tanto el sistema presenta un módulo para registrar todos estos cambios.

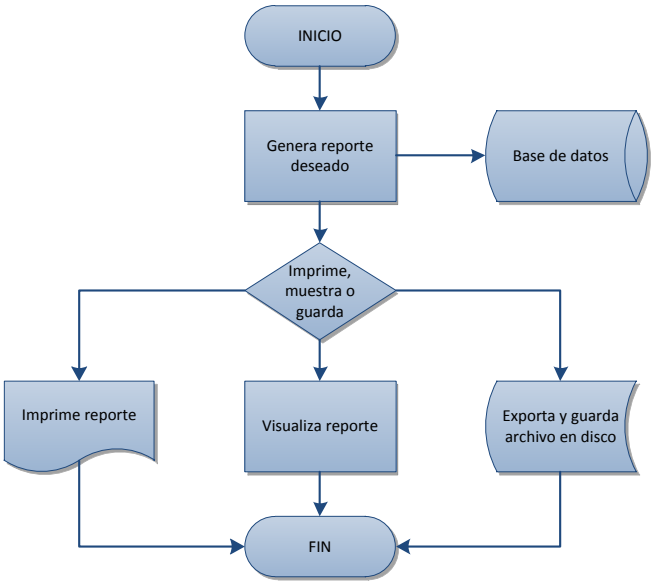


Fuente: [Propia]

Figura 6.6. Flujo de trabajo del módulo de Traslados.

Reportes

El módulo de reportes presenta una variedad de reportes que pueden ser generados.



Fuente: [Propia]

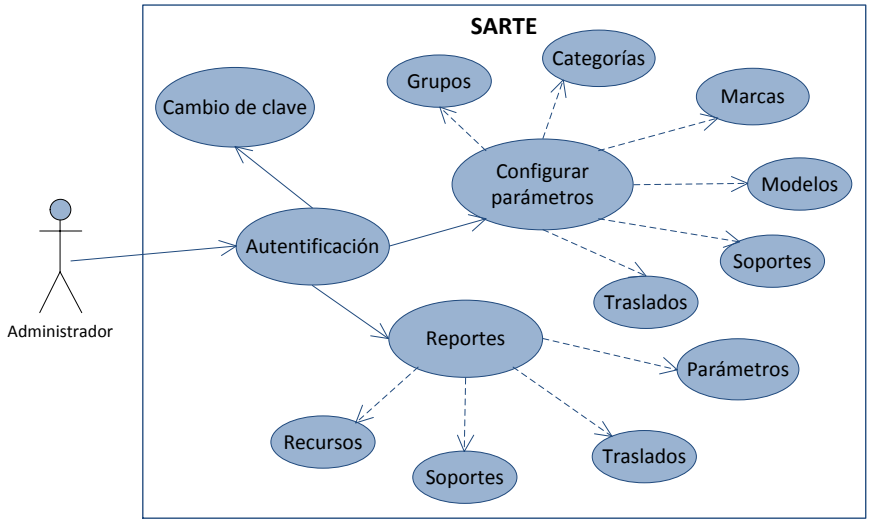
Figura 6.7. Flujo de trabajo del módulo de Reportes.

6.5.3. Diagramas de Casos de Uso

En ingeniería del software, es necesario identificar mediante casos de usos los actores y actividades que están inmersos en el proceso. Una vez realizado este análisis se pudo determinar que existen tres actores principales: Administrador, Soporte y Verificador:

Administrador

Las actividades que el administrador puede realizar en el sistema implementado, se describen en la siguiente figura:

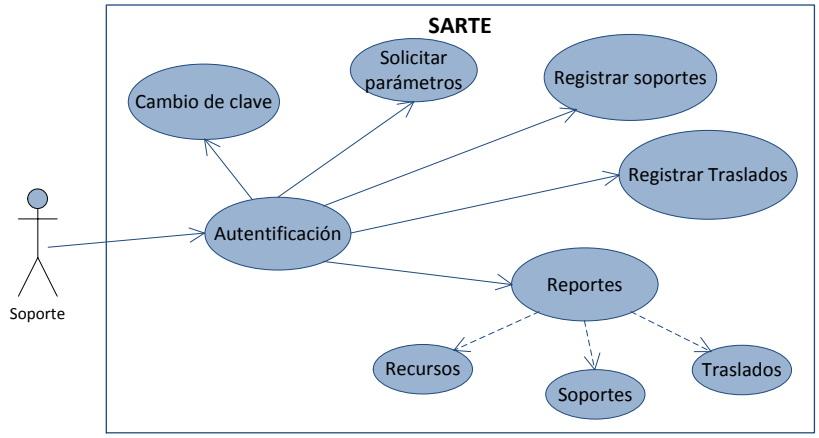


Fuente: [Propia]

Figura 6.8. Diagrama de caso de uso del Administrador.

Soporte

Las actividades que el personal de soporte técnico podrá realizar en el sistema se describen en la siguiente figura:

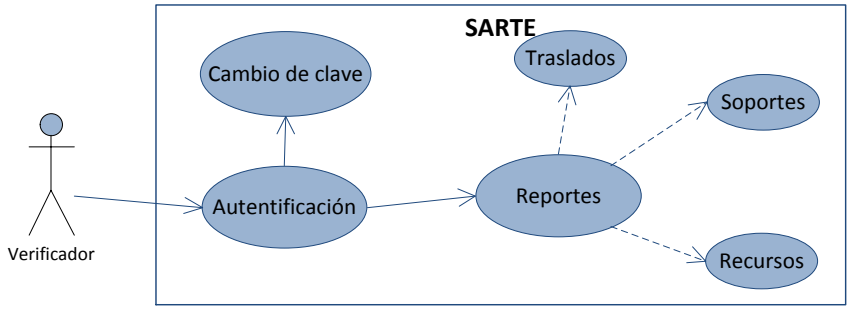


Fuente: [Propia]

Figura 6.9. Diagrama de casos de uso de Soporte.

Verificador

El verificador de información tendrá la posibilidad de generar cualquier tipo de reporte referente a los recursos tecnológicos.



Fuente: [Propia]

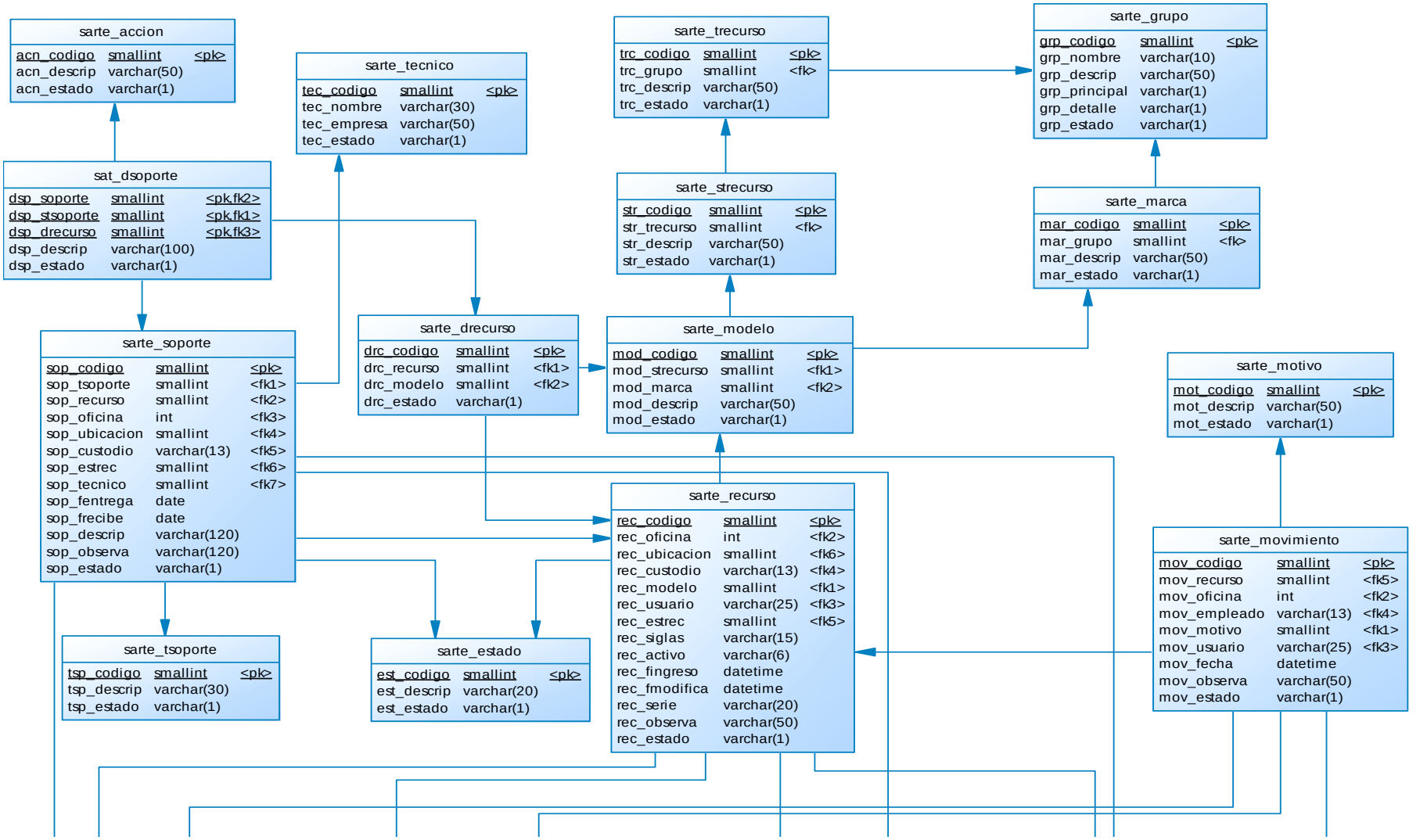
Figura 6.10. Diagrama de casos de uso de Verificador.

6.6. DISEÑO DEL PROYECTO

Durante la fase de análisis pudimos determinar el flujo de información existente durante el proceso de Administración y Control de Recursos Tecnológicos; para posteriormente realizar el diseño de la base de datos que se adapte a las necesidades y requerimientos de los usuarios, diseño en el que se implementó toda la información necesaria y útil para los usuarios y por ende para el desarrollo del presente aplicativo.

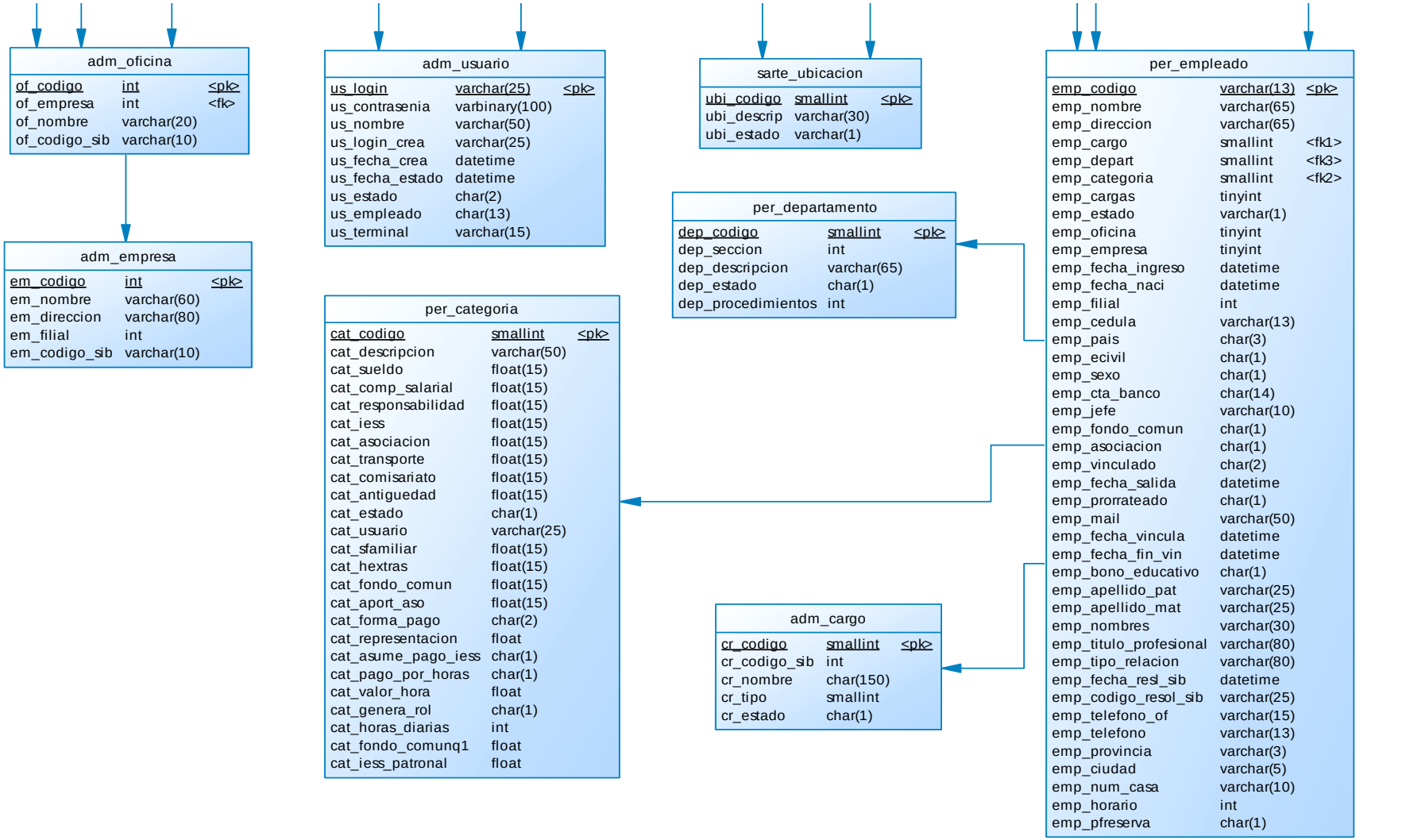
6.6.1. Modelo de Datos

La elaboración del Sistema de Administración y Control de Recursos Tecnológicos para la Cooperativa de Ahorro y Crédito Atuntaqui Ltda., necesita de una variedad de objetos de la base de datos para su funcionalidad y operatividad mismos que se presentan en el siguiente diagrama de entidad-relación para su mayor comprensión y entendimiento.



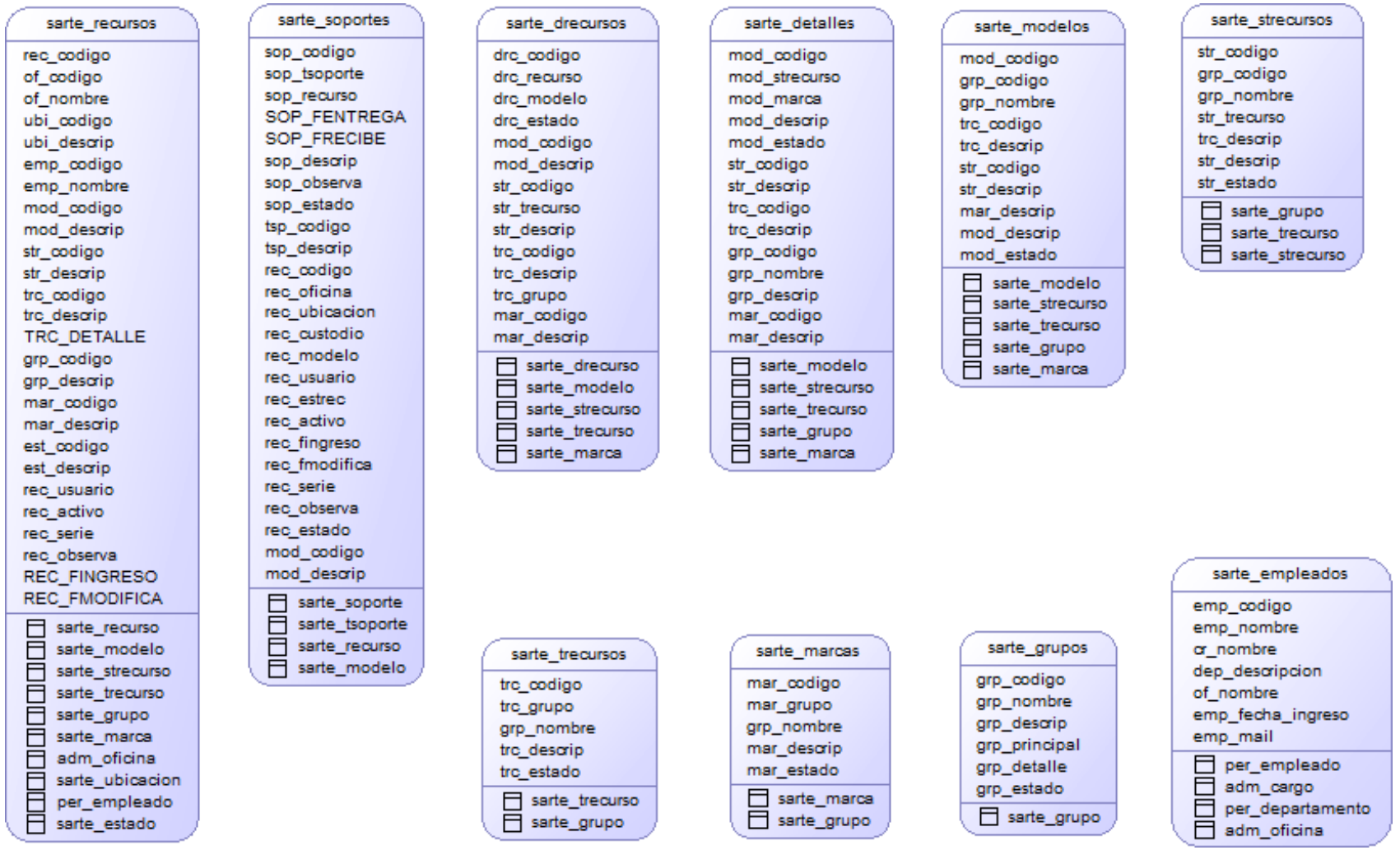
Fuente: [Propia]

Figura 6.11. Diagrama entidad relación de la base (1ra parte).



Fuente: [Propia]

Figura 6.11. Diagrama entidad relación de la base (2da parte).



Fuente: [Propia]

Figura 6.12. Diagrama vistas implementadas en el proyecto.

6.6.2. Diccionario de datos

Durante el proceso de diseño del sistema se crearon una variedad de objetos en el sistema de bases de datos de Sybase existente en la Cooperativa de Ahorro y Crédito Atuntaqui Ltda., específicamente en la base de datos "*atun_riesgos*" que está destinada para la creación de los objetos de los sistemas diseñados e implementados en y para la cooperativa; y nos permitirán almacenar y manipular toda la información suministrada al sistema. Debido a que la cooperativa dispone de una variedad de sistemas enfocados al ámbito administrativo y financiero fue necesario diferenciar todos los objetos del sistema desarrollado, de los ya existentes; para ello se definieron todos los objetos con la palabra inicial clave "*sarte_*" que identificará los objetos pertenecientes al sistema.

A continuación se detalla la estructura y funcionalidad de todos y cada uno de los objetos implementados durante la elaboración del proyecto.

6.6.2.1. TABLAS

Tabla sarte_grupo

Tabla destinada a la parametrización y agrupación de recursos tecnológicos.

CAMPO	TIPO	KEY	DESCRIPCION
grp_codigo	SMALLINT	PK	Identificador único de la tabla grupo.
grp_nombre	VARCHAR(10)		Nombre del grupo de recursos.
grp_descrip	VARCHAR(50)		Descripción larga del grupo de recursos.
grp_principal	VARCHAR(1)		Valor para determinar si el nivel del grupo de recursos es primario.
grp_detalle	VARCHAR(1)		Valor para determinar si el grupo de recursos describe las características de otro grupo de recursos.
grp_estado	VARCHAR(1)		Estado actual del registro.

Fuente: [Propia]

Tabla 6.2. Estructura de la tabla sarte_grupo.

Tabla sarte_trecurso

Permite realizar una clasificación por tipo de los recursos tecnológicos existentes en un grupo.

CAMPO	TIPO	KEY	DESCRIPCION
trc_codigo	SMALLINT	PK	Identificador único de la tabla tipo de recurso (categoría).
trc_grupo	SMALLINT	FK	Grupo al que pertenece la categoría de recursos.
trc_descrip	VARCHAR(50)		Descripción de la categoría de recursos.
trc_estado	VARCHAR(1)		Estado actual del registro.

Fuente: [Propia]

Tabla 6.3. Estructura de la tabla sarte_trecurso.

Tabla sarte_strecurso

Permite realizar una clasificación por subtipo de los recursos tecnológicos existentes en una categoría de tipos de recursos para una mejor organización.

CAMPO	TIPO	KEY	DESCRIPCION
str_codigo	SMALLINT	PK	Identificador único de la tabla subtipo de recurso (subcategoría).
str_trecurso	SMALLINT	FK	Categoría a la que pertenece la subcategoría de recursos.
str_descrip	VARCHAR(50)		Descripción de la subcategoría de recursos.
str_estado	VARCHAR(1)		Estado actual del registro.

Fuente: [Propia]

Tabla 6.4. Estructura de la tabla sarte_strecurso.

Tabla sarte_marca

Almacena la información correspondiente a las marcas de los recursos, clasificadas de acuerdo a los grupos de recursos parametrizados en la tabla sarte_grupo.

CAMPO	TIPO	KEY	DESCRIPCION
mar_codigo	SMALLINT	PK	Identificador único de la tabla marca.
mar_grupo	SMALLINT	FK	Grupo al que pertenece la marca.
mar_descrip	VARCHAR(50)		Nombre de la marca.
mar_estado	VARCHAR(1)		Estado actual del registro.

Fuente: [Propia]

Tabla 6.5. Estructura de la tabla sarte_marca.

Tabla sarte_modelo

Almacena la información correspondiente a los modelos de los recursos tecnológicos, dependiendo del subtipo de recurso y la marca a la que pertenece.

CAMPO	TIPO	KEY	DESCRIPCION
mod_codigo	SMALLINT	PK	Identificador único de la tabla modelo.
mod_strecurso	SMALLINT	FK	Subcategoría de recursos al que pertenece el modelo.
mod_marca	SMALLINT	FK	Marca del modelo.
mod_descrip	VARCHAR(50)		Descripción del modelo.
mod_estado	VARCHAR(1)		Estado actual del registro.

Fuente: [Propia]

Tabla 6.6. Estructura de la tabla sarte_modelo.

Tabla sarte_estado

Permite almacenar la parametrización de los posibles estados en los que puede encontrarse un recurso tecnológico.

CAMPO	TIPO	KEY	DESCRIPCION
est_codigo	SMALLINT	PK	Identificador único de la tabla estado.
est_descrip	VARCHAR(50)		Descripción del estado.
est_estado	VARCHAR(1)		Estado actual del registro.

Fuente: [Propia]

Tabla 6.7. Estructura de la tabla sarte_estado.

Tabla sarte_ubicacion

Guarda la información de las diferentes áreas o ubicaciones donde se puede encontrar un recurso, para su localización.

CAMPO	TIPO	KEY	DESCRIPCION
ubi_codigo	SMALLINT	PK	Identificador único de la tabla ubicación.
ubi_descrip	VARCHAR(50)		Descripción de la ubicación o área.
ubi_estado	VARCHAR(1)		Estado actual del registro.

Fuente: [Propia]

Tabla 6.8. Estructura de la tabla sarte_ubicacion.

Tabla sarte_recurso

Almacena la información correspondiente a los diferentes tipos de recursos tecnológicos, categorizándolos de acuerdo a la parametrización establecida en las tablas anteriores.

CAMPO	TIPO	KEY	DESCRIPCION
rec_codigo	SMALLINT	PK	Identificador único de la tabla recurso.
rec_oficina	INT	FK	Oficina donde se encuentra el recurso.
rec_ubicacion	SMALLINT	FK	Área donde se encuentra el recurso.
rec_custodio	VARCHAR(13)	FK	Custodio o responsable del recurso.
rec_modelo	SMALLINT	FK	Modelo del recurso tecnológico.
rec_usuario	VARCHAR(25)	FK	Usuario que ingreso el recurso tecnológico.
rec_estrec	SMALLINT	FK	Estado actual del recurso tecnológico.
rec_activo	VARCHAR(15)		Código de activo fijo, de no existir este campo se queda en NULL.
rec_fingreso	DATE		Fecha cuando se realizó el ingreso del recurso, esta fecha nunca cambia después del ingreso.
rec_fmodifica	DATE		Fecha de la última modificación del recurso.
rec_serie	VARCHAR(20)		Número de serie del recurso.
rec_observa	VARCHAR(50)		Breve observación referente al recurso.
rec_estado	VARCHAR(1)		Estado actual del registro.

Fuente: [Propia]

Tabla 6.9. Estructura de la tabla sarte_recurso.

Tabla sarte_drecurso

Permite detallar las características o componentes principales de los recursos tecnológicos.

CAMPO	TIPO	KEY	DESCRIPCION
drc_codigo	SMALLINT	PK	Identificador único de la tabla detalle del recurso.
drc_recurso	SMALLINT	FK	Código del recurso al que pertenece el detalle.
drc_modelo	SMALLINT	FK	Modelo del detalle del recurso.
drc_estado	VARCHAR(1)		Estado actual del registro.

Fuente: [Propia]

Tabla 6.10. Estructura de la tabla sarte_drecurso.

Tabla sarte_tsoporte

Registra los tipos de mantenimientos que se pueden realizar a los recursos tecnológicos.

CAMPO	TIPO	KEY	DESCRIPCION
tsp_codigo	SMALLINT	PK	Identificador único de la tabla tipo de soporte.
tsp_descrip	VARCHAR(50)		Descripción del tipo de soporte.
tsp_estado	VARCHAR(1)		Estado actual del registro.

Fuente: [Propia]

Tabla 6.11. Estructura de la tabla sarte_tsoporte.

Tabla sarte_tecnico

Almacena los datos del técnico que realizo el mantenimiento de un recurso.

CAMPO	TIPO	KEY	DESCRIPCION
tec_codigo	SMALLINT	PK	Identificador único de la tabla técnico.
tec_nombre	VARCHAR(30)		Nombre del técnico.
tec_empresa	VARCHAR(50)		Nombre de la empresa del técnico.
tec_estado	VARCHAR(1)		Estado actual del registro.

Fuente: [Propia]

Tabla 6.12. Estructura de la tabla sarte_tecnico.

Tabla sarte_accion

Almacena la parametrización de las tareas o acciones a realizarse durante el mantenimiento.

CAMPO	TIPO	KEY	DESCRIPCION
acn_codigo	SMALLINT	PK	Identificador único de la tabla acción.
acn_descrip	VARCHAR(20)		Descripción de la acción o tarea.
acn_estado	VARCHAR(1)		Estado actual del registro.

Fuente: [Propia]

Tabla 6.13. Estructura de la tabla sarte_accion.

Tabla sarte_soporte

Tabla en la cual se registran todos los mantenimientos brindados a determinado recurso.

CAMPO	TIPO	KEY	DESCRIPCION
sop_codigo	SMALLINT	PK	Identificador único de la tabla soporte.
sop_tsoporte	SMALLINT	FK	Código perteneciente al tipo de mantenimiento.
sop_recurso	SMALLINT	FK	Recurso al que se realizó el mantenimiento.
sop_oficina	SMALLINT	FK	Oficina donde se encontraba el recurso al momento del mantenimiento.
sop_ubicacion	SMALLINT	FK	Ubicación o área donde se encontraba el recurso al momento del mantenimiento.
sop_custodio	VARCHAR(13)	FK	Custodio del recurso al momento de realizar el mantenimiento.
sop_estrec	SMALLINT	FK	Estado del recurso al momento de realizar el mantenimiento.
sop_tecnico	SMALLINT	FK	Técnico que brindo el mantenimiento.
sop_fentrega	DATE		Fecha de entrega del recurso para realizar el mantenimiento.
sop_frecibe	DATE		Fecha de recepción del recurso después de realizar el mantenimiento.
sop_descrip	VARCHAR(200)		Descripción del problema o inconveniente presentado con el recurso.
sop_observa	VARCHAR(200)		Detalle de lo realizado por parte del técnico durante el proceso de mantenimiento.
sop_estado	VARCHAR(1)		Estado actual del registro.

Fuente: [Propia]

Tabla 6.14. Estructura de la tabla sarte_soporte.

Tabla sarte_dsoporte

Permite detallar todas las acciones tomadas durante el proceso de mantenimiento.

CAMPO	TIPO	KEY	DESCRIPCION
dsp_soporte	SMALLINT	PK FK	Número de mantenimiento realizado.
dsp_drecurso	SMALLINT	PK FK	Características o componentes del recurso afectados durante el proceso de mantenimiento.
dsp_accion	SMALLINT	FK	Modelo del detalle del recurso.
dsp_estado	VARCHAR(1)		Estado actual del registro.

Fuente: [Propia]

Tabla 6.15. Estructura de la tabla sarte_dsoporte.

Tabla sarte_motivo

Registra los motivos por los cuales se puede realizar el traslado de un recurso.

CAMPO	TIPO	KEY	DESCRIPCION
mot_codigo	SMALLINT	PK	Identificador único de la tabla motivo.
mot_descrip	VARCHAR(50)		Descripción del motivo de los traslados.
mot_estado	VARCHAR(1)		Estado actual del registro.

Fuente: [Propia]

Tabla 6.16. Estructura de la tabla sarte_motivo.

Tabla sarte_movimiento

Almacena todos los movimientos realizados con los recursos tecnológicos para registrar sus custodios y ubicaciones tenidos durante la vida útil del recurso.

CAMPO	TIPO	KEY	DESCRIPCION
mov_codigo	SMALLINT	PK	Identificador único de la tabla movimiento.
mov_recurso	SMALLINT	FK	Código del recurso que cambio de ubicación y/o custodio.
mov_oficina	INT	FK	Oficina destino donde se trasladó el recurso.
mov_ubicacion	SMALLINT	FK	Área destino donde se ubicó el recurso.
mov_custodio	VARCHAR(13)	FK	Custodio nuevo al cual se asignó el recurso.
mov_motivo	SMALLINT	FK	Motivo por el cual se realizó el traslado.
mov_estrec	SMALLINT	FK	Estado del recurso al momento de realizar el traslado.
mov_usuario	VARCHAR(25)	FK	Usuario que registro el traslado del recurso.
mov_fecha	DATE		Fecha cuando se realizó el movimiento.
mov_observa	VARCHAR(50)		Almacena alguna observación sobre el movimiento.
mov_estado	VARCHAR(1)		Estado actual del registro.

Fuente: [Propia]

Tabla 6.17. Estructura de la tabla sarte_movimiento.

Además, para la implementación del presente proyecto fue necesario utilizar algunas tablas existentes en la base de datos "atun_riesgos", las mismas que se detallan a continuación:

Tabla adm_empresa

Tabla en donde se detalla los datos principales de la cooperativa.

CAMPO	TIPO	KEY	DESCRIPCION
em_codigo	INT	PK	Identificador único de la tabla empresa.
em_nombre	VARCHAR(60)		Nombre de la empresa.
em_direccion	VARCHAR (80)		Dirección de la empresa.
em_filial	INT		Filial a la que pertenece la empresa.
em_codigo_sib	VARCHAR (10)		Código de la empresa asignado por la Superintendencia de Bancos.

Fuente: [Propia]

Tabla 6.18. Estructura de la tabla adm_empresa.

Tabla adm_oficina

Tabla en donde se encuentran los datos de las oficinas existentes de la cooperativa, para ubicar el lugar donde se encuentran los recursos.

CAMPO	TIPO	KEY	DESCRIPCION
of_codigo	INT	PK	Identificador único de la tabla oficina.
of_empresa	INT	FK	Código de la empresa a la cual pertenece la oficina.
of_nombre	VARCHAR (20)		Nombre de la oficina.
of_codigo_sib	VARCHAR (10)		Código de la oficina asignado por la SBS.

Fuente: [Propia]

Tabla 6.19. Estructura de la tabla adm_oficina.

Tabla adm_usuario

Tabla en la que se almacenan el login y la contraseña de los usuarios para ingresar a los diferentes sistemas de la cooperativa.

CAMPO	TIPO	KEY	DESCRIPCION
us_login	VARCHAR (25)	PK	Identificador único de la tabla usuario Login de acceso al sistema.
us_contrasenia	VARBINARY(100)		Contraseña cifrada de acceso al sistema.
us_nombre	VARCHAR (50)		Nombre del usuario.
us_login_crea	VARCHAR (25)		Login del usuario que ingreso el usuario.
us_fecha_crea	DATETIME		Fecha cuando se ingresó el usuario.
us_fecha_estado	DATETIME		Fecha de la última modificación del usuario.
us_estado	CHAR (2)		Estado actual del usuario.
us_empleado	CHAR (13)		Empleado al que pertenece el login.
us_terminal	VARCHAR (15)		Estación de trabajo a la que pertenece el usuario.

Fuente: [Propia]

Tabla 6.20. Estructura de la tabla adm_usuario.

Tabla per_departamento

Especifica los datos de los departamentos existentes en la cooperativa.

CAMPO	TIPO	KEY	DESCRIPCION
dep_codigo	SMALLINT	PK	Identificador único de la tabla departamento, código del departamento.
dep_seccion	INT		Código de la sección a la que pertenece el departamento.
dep_descripcion	VARCHAR (65)		Descripción del departamento.
dep_estado	CHAR (1)		Estado actual del departamento.
dep_procedimientos	INT		Código de los procedimientos correspondientes al departamento.

Fuente: [Propia]

Tabla 6.21. Estructura de la tabla per_departamento.

Tabla adm_cargo

Almacena la parametrización correspondiente a los cargos de los empleados de la cooperativa.

CAMPO	TIPO	KEY	DESCRIPCION
of_codigo	INT	PK	Identificador único de la tabla oficina.
of_empresa	INT	FK	Código de la empresa a la cual pertenece la oficina.
of_nombre	VARCHAR (20)		Nombre de la oficina.
of_codigo_sib	VARCHAR (10)		Código de la oficina asignado por la Superintendencia de Bancos.

Fuente: [Propia]

Tabla 6.22. Estructura de la tabla adm_cargo.

Tabla per_categoria

Tabla donde se ingresan las categorías de los empleados y se establecen los parámetros correspondientes para la gestión de las mismas.

CAMPO	TIPO	KEY	DESCRIPCION
cat_codigo	SMALLINT	PK	Identificador único de la tabla categoría.
cat_descripcion	VARCHAR (50)		Descripción o nombre de la categoría.
cat_sueldo	FLOAT		Valor de la remuneración mensual correspondiente a la categoría.
cat_responsabilidad	FLOAT		Cantidad asignada por responsabilidad.
cat_iess	FLOAT		Porcentaje de aportación al IESS.
cat_asociacion	FLOAT		Identifica las categorías que pertenecen a la asociación de empleados.
cat_estado	CHAR (1)		Estado actual de la categoría.
cat_usuario	VARCHAR (25)		Usuario que ingreso la categoría.
cat_fondo_comun	FLOAT		Valor debitado para el fondo común.
cat_forma_pago	CHAR (2)		Indica si la forma de pago es Quincenal o Mensual.
cat_representacion	FLOAT		Valor asignado por representación.
cat_asume_pago_iess	CHAR (1)		Indica si el patrono asume el pago al IESS.
cat_valor_hora	FLOAT		Indica el valor de la hora extra.
cat_genera_rol	CHAR (1)		Indica si la categoría general rol de pagos.
cat_horas_diarias	INT		Indica el número de horas diarias a laborar por parte del empleado.
cat_fondo_comunq1	FLOAT		Valor debitado para el fondo común en la primera quincena.
cat_iess_patronal	FLOAT		Porcentaje del IESS correspondiente al empleador.

Fuente: [Propia]

Tabla 6.23. Estructura de la tabla per_categoria.

Tabla per_empleado

Tabla donde se registran todos los datos correspondientes a los empleados de la cooperativa, y utilizados para asignar los custodios responsables de los recursos.

CAMPO	TIPO	KEY	DESCRIPCION
emp_codigo	VARCHAR (13)	PK	Identificador único de la tabla empleado.
emp_nombre	VARCHAR (65)		Nombre completo del empleado.
emp_direccion	VARCHAR (65)		Dirección domiciliaria del empleado.
emp_cargo	SMALLINT	FK	Cargo actual del empleado.
emp_depart	SMALLINT	FK	Departamento actual del empleado.
emp_categoria	SMALLINT	FK	Categoría actual del empleado.
emp_cargas	TINYINT		Número de cargar familiares.
emp_estado	VARCHAR (1)		Estado actual del empleado.
emp_oficina	TINYINT		Oficina a la que pertenece el empleado.
emp_empresa	TINYINT		Empresa a la que pertenece el empleado.
emp_fecha_ingreso	DATETIME		Fecha de ingreso a la cooperativa.
emp_fecha_naci	DATETIME		Fecha de nacimiento del empleado.
emp_filial	INT		Filial a la que pertenece el empleado.
emp_cedula	VARCHAR (13)		Número de cédula de identidad.
emp_pais	CHAR (3)		Nacionalidad del empleado.
emp_ecivil	CHAR (1)		Estado civil actual del empleado.
emp_sexo	CHAR (1)		Genero del empleado.
emp_cta_banco	CHAR (14)		Número de cuenta del empleado.
emp_jefe	VARCHAR (10)		Código del jefe inmediato.
emp_fondo_comun	CHAR (1)		Indica si aporta a no al fondo común.
emp_asociacion	CHAR (1)		Indica si es o no parte de la asociación de empleados.
emp_vinculado	CHAR (2)		Indica si el empleado es vinculado.
emp_fecha_salida	DATETIME		Fecha de salida de la institución.
emp_mail	VARCHAR (50)		Correo electrónico del empleado.
emp_fecha_vincula	DATETIME		Fecha de vinculación del empleado.
emp_fecha_fin_vin	DATETIME		Fecha de finalización de la vinculación.
emp_apellido_pat	VARCHAR (25)		Apellido paterno del empleado.
emp_apellido_mat	VARCHAR (25)		Apellido materno del empleado.
emp_nombres	VARCHAR (30)		Nombres del empleado.
emp_titulo_ profesional	VARCHAR (80)		Título profesional que tiene el empleado.
emp_telefono_of	VARCHAR (15)		Teléfono de la oficina.
emp_telefono	VARCHAR (13)		Teléfono personal.
emp_provincia	VARCHAR (3)		Provincia donde vive el empleado.
emp_ciudad	VARCHAR (5)		Ciudad donde vive el empleado.
emp_num_casa	VARCHAR (10)		Número de la casa donde vive.
emp_horario	INT		Horario asignado al empleado.
emp_pfreserva	CHAR (1)		Indica si se le deposita o no los fondos de reserva en la cuenta.
emp_imp_renta	INT		Indica si paga o no el impuestoa la renta.

Fuente: [Propia]

Tabla 6.24. Estructura de la tabla per_empleado.

6.6.2.2. VISTAS

Vista sarte_recursos

Vista generada a partir de las tablas "*sarte_recurso*", "*sarte_modelo*", "*sarte_strecurso*", "*sarte_trecurso*", "*sarte_grupo*", "*sarte_marca*", "*adm_oficina*", "*sarte_ubicacion*", "*per_empleado*" y "*sarte_estado*" para visualizar en una sola consulta todos los datos correspondientes a los recursos tecnológicos activos del sistema y generar los informes.

CAMPO	TABLA
rec_codigo	sarte_recurso
rec_usuario	sarte_recurso
rec_activo	sarte_recurso
rec_serie	sarte_recurso
rec_fingreso	sarte_recurso
rec_fmodifica	sarte_recurso
rec_observa	sarte_recurso
of_codigo	adm_oficina
of_nombre	adm_oficina
ubi_codigo	sarte_ubicacion
ubi_descrip	sarte_ubicacion
emp_codigo	per_empleado
emp_nombre	per_empleado
grp_codigo	sarte_grupo
grp_descrip	sarte_grupo
trc_codigo	sarte_trecurso
trc_descrip	sarte_trecurso
trc_detalle	sarte_trecurso
str_codigo	sarte_strecurso
str_descrip	sarte_strecurso
mod_codigo	sarte_modelo
mod_descrip	sarte_modelo
mar_codigo	sarte_marca
mar_descrip	sarte_marca
est_codigo	sarte_estado
est_descrip	sarte_estado

Fuente: [Propia]

Tabla 6.25. Estructura lógica de la vista sarte_recursos.

Vista sarte_grupos

Muestra un listado de todos los grupos de recursos activos dentro del sistema, para indicar con que grupos de recursos se debe interactuar.

CAMPO	TABLA
grp_codigo	sarte_grupo
grp_nombre	sarte_grupo
grp_descrip	sarte_grupo
grp_principal	sarte_grupo
grp_detalle	sarte_grupo
grp_estado	sarte_grupo

Fuente: [Propia]

Tabla 6.26. Estructura lógica de la vista sarte_grupos.

Vista sarte_empleados

Vista generada a partir de las tablas "*per_empleado*", "*adm_cargo*", "*per_departamento*", "*adm_oficina*" del sistema de personal; con el fin de visualizar la información más importante de los empleados de la Cooperativa y requerida por el sistema SARTE.

CAMPO	TABLA
emp_codigo	per_empleado
emp_nombre	per_empleado
emp_cargo	adm_cargo
emp_depart	per_departamento
emp_oficina	adm_oficina
emp_fingreso	per_empleado
emp_mail	per_empleado

Fuente: [Propia]

Tabla 6.27. Estructura lógica de la vista sarte_empleados.

Vista sarte_marcas

Vista generada a partir de las tablas "*sarte_marca*" y "*sarte_grupo*" para visualizar todas las marcas disponibles en el sistema y clasificadas por grupo de recursos.

CAMPO	TABLA
mar_codigo	sarte_marca
mar_grupo	sarte_marca
grp_nombre	sarte_grupo
mar_descrip	sarte_marca
mar_estado	sarte_marca

Fuente: [Propia]

Tabla 6.28. Estructura lógica de la vista sarte_marcas.

Vista sarte_detalles

Vista generada a partir de las tablas "*sarte_modelo*", "*sarte_strecurso*", "*sarte_trecurso*", "*sarte_grupo*" y "*sarte_marca*" para mostrar un listado de todas las características de los recursos parametrizadas y que pueden ser asignadas a un recurso.

CAMPO	TABLA
mod_codigo	sarte_modelo
mod_strecurso	sarte_modelo
mod_marca	sarte_modelo
mod_descrip	sarte_modelo
mod_estado	sarte_modelo
str_codigo	sarte_strecurso
str_descrip	sarte_strecurso
trc_codigo	sarte_trecurso
trc_descrip	sarte_trecurso
grp_codigo	sarte_grupo
grp_nombre	sarte_grupo
grp_descrip	sarte_grupo
mar_codigo	sarte_marca
mar_descrip	sarte_marca

Fuente: [Propia]

Tabla 6.29. Estructura lógica de la vista sarte_detalles.

Vista sarte_drecursos

Vista generada a partir de las tablas "*sarte_drecurso*", "*sarte_modelo*", "*sarte_strecurso*", "*sarte_trecurso*" y "*sarte_marca*" que nos permite visualizar todos los recursos ingresados en el sistema con sus respectivas características.

CAMPO	TABLA
drc_codigo	sarte_drecurso
drc_recurso	sarte_drecurso
drc_modelo	sarte_drecurso
drc_estado	sarte_drecurso
mod_codigo	sarte_modelo
mod_descrip	sarte_modelo
str_codigo	sarte_strecurso
str_trecurso	sarte_strecurso
str_descrip	sarte_strecurso
trc_codigo	sarte_trecurso
trc_descrip	sarte_trecurso
trc_grupo	sarte_trecurso
mar_codigo	sarte_marca
mar_descrip	sarte_marca

Fuente: [Propia]

Tabla 6.30. Estructura lógica de la vista sarte_drecursos.

Vista sarte_soportes

Vista generada a partir de las tablas "*sarte_soporte*", "*sarte_tsoporte*", "*sarte_recurso*" y "*sarte_modelo*" para mostrar un listado de todos los mantenimientos realizados a los recursos.

CAMPO	TABLA
sop_codigo	sarte_soporte
sop_tsoporte	sarte_soporte
sop_recurso	sarte_soporte
sop_fentrega	sarte_soporte
sop_frecibe	sarte_soporte
sop_descrip	sarte_soporte
sop_observa	sarte_soporte
sop_estado	sarte_soporte
tsp_codigo	sarte_tsoporte
tsp_descrip	sarte_tsoporte
rec_codigo	sarte_recurso
rec_oficina	sarte_recurso
rec_ubicacion	sarte_recurso
rec_custodio	sarte_recurso
rec_modelo	sarte_recurso
rec_usuario	sarte_recurso
rec_estrec	sarte_recurso
rec_activo	sarte_recurso
rec_fingreso	sarte_recurso
rec_fmodifica	sarte_recurso
rec_serie	sarte_recurso
rec_observa	sarte_recurso
rec_estado	sarte_recurso
mod_codigo	mod_codigo
mod_descrip	mod_codigo

Fuente: [Propia]

Tabla 6.31. Estructura lógica de la vista sarte_soportes.

Vista sarte_trecursos

Vista generada a partir de las tablas "*sarte_grupo*" y "*sarte_trecurso*" que nos permite visualizar un listado de todas las categorías parametrizadas en el sistema.

CAMPO	TABLA
grp_codigo	sarte_grupo
grp_nombre	sarte_grupo
trc_codigo	sarte_trecurso
trc_descrip	sarte_trecurso
trc_estado	sarte_trecurso

Fuente: [Propia]

Tabla 6.32. Estructura lógica de la vista sarte_trecursos.

Vista sarte_modelos

Vista generada a partir de las tablas "*sarte_modelo*", "*sarte_strecurso*", "*sarte_trecurso*", "*sarte_grupo*" y "*sarte_marca*" para mostrar un listado de todos los modelos de recursos disponibles para el ingreso de nuevos recursos al sistema.

CAMPO	TABLA
mod_codigo	sarte_modelo
mod_strecurso	sarte_modelo
mod_marca	sarte_modelo
mod_descrip	sarte_modelo
mod_estado	sarte_modelo
str_codigo	sarte_strecurso
str_descrip	sarte_strecurso
trc_codigo	sarte_trecurso
trc_descrip	sarte_trecurso
grp_codigo	sarte_grupo
grp_nombre	sarte_grupo
grp_descrip	sarte_grupo
mar_codigo	sarte_marca
mar_descrip	sarte_marca

Fuente: [Propia]

Tabla 6.33. Estructura lógica de la vista sarte_modelos.

Vista sarte_strecursos

Vista generada a partir de las tablas "*sarte_grupo*", "*sarte_trecurso*" y "*sarte_strecurso*" que muestra el detalle de todas las subcategorías de recursos parametrizadas en el sistema.

CAMPO	TABLA
grp_codigo	sarte_grupo
grp_nombre	sarte_grupo
trc_codigo	sarte_trecurso
trc_descrip	sarte_trecurso
str_trecurso	sarte_strecurso
str_descrip	sarte_strecurso
str_estado	sarte_strecurso

Fuente: [Propia]

Tabla 6.34. Estructura lógica de la vista sarte_strecursos.

6.7. IMPLEMENTACION DEL PROYECTO

6.7.1. Instalación del sistema

Para realizar la instalación del Sistema SARTE (Sistema de Administración y Control de Recurso TEcnológicos), se debe seguir los siguientes pasos:

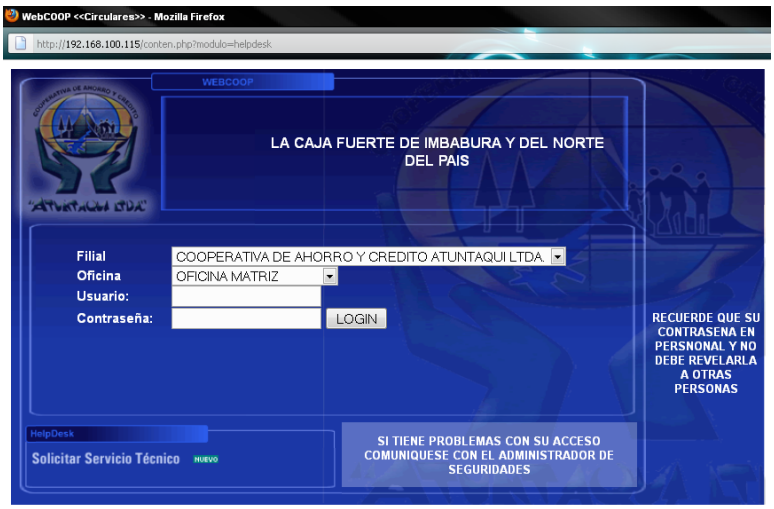
1. Compilar los script adjuntos en el presente proyecto para la creación de los nuevos objetos del sistema en la base de datos.
2. Instalar un servidor de aplicaciones que soporte la tecnología Java como Tomcat o GlassFish.
3. Copiar el archivo comprimido final del sistema "sarte.war", en el directorio principal del servidor de aplicaciones.
4. Por último, reinicar el servidor de aplicaciones

Para comprobar que la instalación se realizo sin ningún inconveniente se deberá dirigirse al navegador e ingresar a la siguiente dirección: "*http://servidor:puerto/sarte*" siendo "*servidor*" el nombre o dirección ip del equipo donde se encuentra nuestro servidor de aplicaciones y "*puerto*" el número del puerto configurado con al servidor de aplicaciones.

6.7.2. Funcionalidad del sistema

Una vez iniciado el sistema desde un navegador web, la primera pantalla es la de autentificación, en donde el usuario deberá ingresar su login y contraseña para ingresar al sistema.

Si el ingreso es satisfactorio se desplegará la pantalla inicial del sistema, caso contrario se pedirá nuevamente el ingreso de la contraseña hasta un máximo de tres veces, si la tercera vez el ingreso del login y la contraseña resulta erróneo el usuario al cual se hace referencia se bloquea por seguridad.



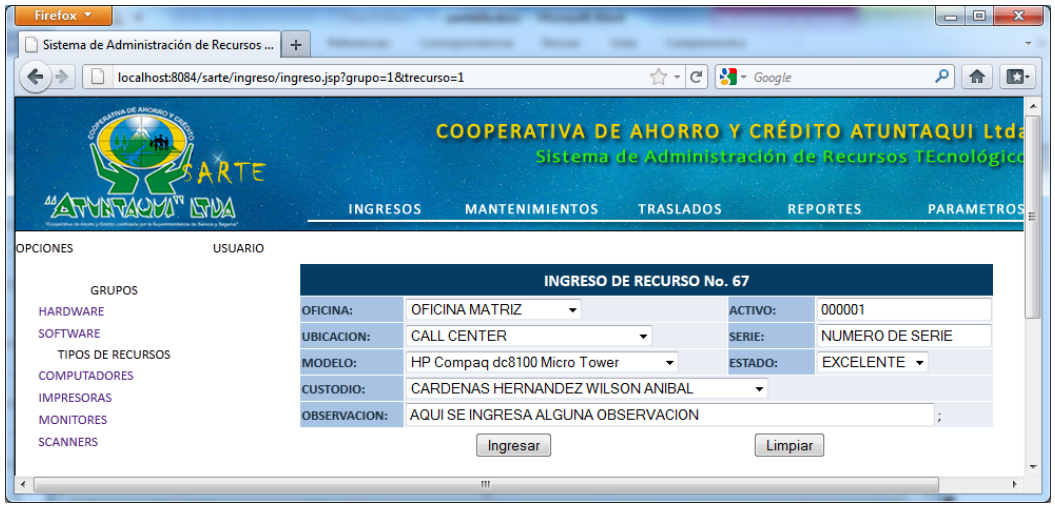
Fuente: [Propia]

Figura 6.13. Pantalla de autentificación.

Una vez ingresado al sistema se muestra la pantalla incial donde podemos acceder a los diferentes módulos del sistema, como son: Ingreso de datos, Mantenimiento de recursos, traslado de recursos, emisión de reportes y la configuración de parámetros.

Ingreso de datos

En el módulo de ingreso de datos tenemos toda la información necesaria para ingresar los nuevos recursos tecnológicos.



Fuente: [Propia]

Figura 6.14. Módulo de Ingreso de datos del sistema.

Mantenimiento de recursos

En el módulo de mantenimiento registramos todo las modificaciones y alteraciones que los recursos sufren durante el tiempo de vida útil.

Fuente: [Propia]

Figura 6.15. Módulo de Mantenimientos del sistema.

Traslado de recursos

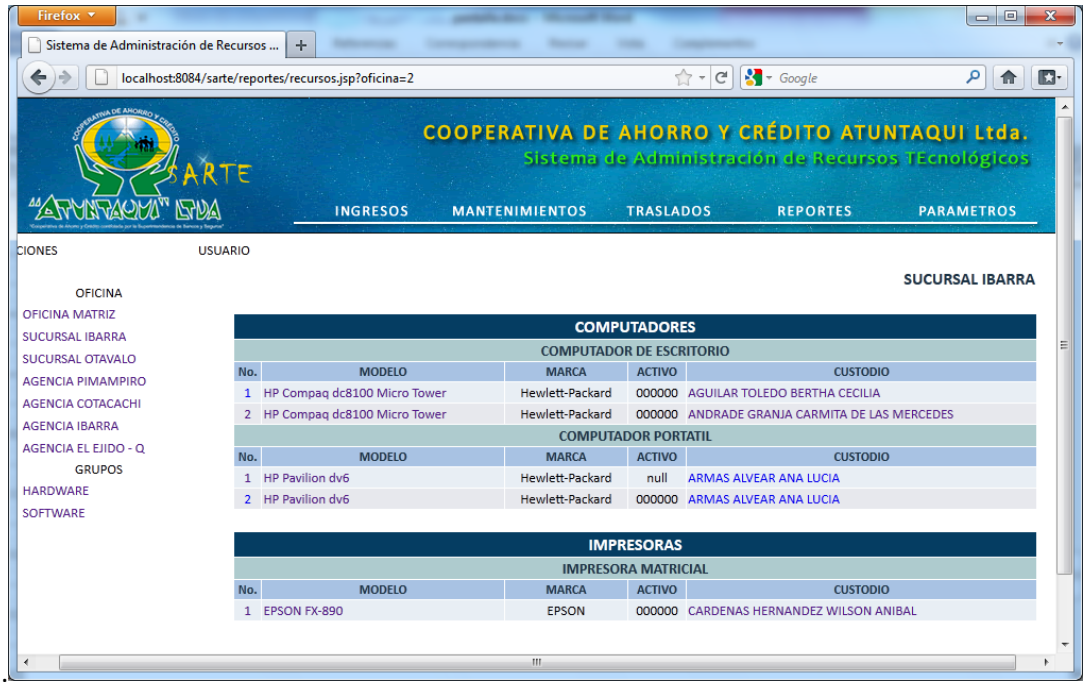
En el módulo de traslado se registra todos los cambios de ubicación y de custodios que se realiza con los registros, para de esta manera poder llevar un control eficiente de el lugar y responsable del recurso tecnológico.

Fuente: [Propia]

Figura 6.16. Módulo de Traslado de equipos del sistema.

Generación de reportes

En el módulo de reportes se encuentran una variedad de reportes que el usuario puede generar dependiendo de la necesidad.



Fuente: [Propia]

Figura 6.17. Módulo de Reportes del sistema.

Administración del sistema

En el módulo de administración se encuentra la configuración de todos los parámetros del sistema A continuación una pantalla capturada del sistema:.

Fuente: [Propia]

Figura 6.18. Módulo de Administración del sistema.

6.7.3. Pruebas

Una vez finalizado el proceso de diseño, desarrollo e instalación del sistema se realizaron las pruebas correspondientes del mismo; pudiendo palpar de manera real la funcionalidad del sistemas y verificando que la operatividad del sistema esta acorde a las necesidades y requerimientos planteados por el usuario.

6.7.4. Capacitación

Al culminar la implementación del sistema se realizó una capacitación al personal del departamento informático de la Cooperativa de Ahorro y Crédito Atuntaqui Ltda., para de est manera aprovechar las bondades del sistema; misma que concluyó de manera satisfactoria.

6.7.5. Puesta en Marcha

Por último, al culminar la implementación del sistema, la dirección del Departamento de Sitemas de la Cooperativa, dicidió que durante el año en curso se procederá con el ingreso de datos al sistema, para de esta manera poder contar con información exacta de los recursos tecnológicos existentes en la Cooperativa y desde el próximo año continuar con las demás opciones del sistema como son: el registro de mantenimientos y traslado de equipo.

CAPÍTULO VII

CONCLUSIONES Y RECOMENDACIONES

VERIFICACIÓN DE LA HIPÓTESIS

CONCLUSIONES

RECOMENDACIONES

TEMAS DE TESIS

7.1. VERIFICACIÓN DE LA HIPÓTESIS

La hipótesis planteada es:

"*El proceso de administración y control de los recursos tecnológicos de la Cooperativa de Ahorro y Crédito Atuntaqui Ltda., mejorará sustancialmente con la implementación de una aplicación web que optimice el proceso y proporcione información eficiente, precisa, clara y oportuna.*"

Al término de este trabajo de investigación, y después de haber diseñado e implementado un sistema de información web en la Cooperativa de Ahorro y Crédito Atuntaqui Ltda., para la administración y control de los recursos tecnológicos, se concluye que la **"hipótesis es verdadera"**, pues se verifico que:

La implementación de un sistema informático en la Cooperativa de Ahorro y Crédito "Atuntaqui" Ltda., simplicó notablemente el proceso de administración y control de los recursos tecnológicos con la automatización de tareas repetitivas y registro de manera organizada de todas las actividades. Además, con la puesta en marcha del sistema en la Cooperativa de Ahorro y Crédito Atuntaqui Ltda., se dispone de información eficiente, precisa, clara y oportuna de los recursos tecnológicos existentes y de sus respectivas modificaciones y adaptaciones.

7.2. Conclusiones

- El uso de una herramienta informática para el diseño y generación de reportes en los sistemas de información web, como JasperReports; disminuye el costo del proyecto ya que permite un ahorro de grandes cantidades de dinero destinado a la adquisición de licencias, a la vez que facilita el diseño de los reportes que visualizan la información de los sistemas disminuyendo el tiempo de desarrollo del proyecto.
- El estudio minucioso y detallado de las herramientas de desarrollo utilizadas, en el diseño e implementación de nuevos sistemas de información nos permite aprovechar las ventajas y beneficios que estas nos brindan.
- JasperReports es una herramienta de código abierto muy poderosa para producir documentos listos para imprimir, visualizar o exportar a otros formatos como PDF, XLS, HTML, entre otros, que puede ser incorporada a cualquier sistema de información desarrollado en la plataforma Java.

- El diseño de una interfaz de usuario fácil, amigable y eficiente, facilita la integración y adaptación del usuario con el nuevo sistema de información, permitiéndonos aprovechar las ventajas que este nos brinda.
- La Cooperativa de Ahorro y Crédito Atuntaqui Ltda cuenta con una herramienta de software especializada y sofisticada para administrar los recursos tecnológicos, logrando que el proceso de control de la tecnología sea eficiente, con la automatización de tareas que antes eran manuales, registrando de manera organizada los datos generador por el proceso y disponiendo de información clara, precisa y oportuna.
- El desarrollo de aplicaciones utilizando herramientas Open Source, permite que otros programadores puedan modificar el código de los sistemas, permitiendo adaptarlo a sus necesidades.

7.3. Recomendaciones

- Utilizar herramientas informáticas Open Source pues estas nos brindan un sin número de ventajas técnicas y permitiéndonos disminuir los costos por licenciamiento de herramientas, logrando disminuir notablemente el costo final de nuestro proyecto de software.
- El personal técnico de la Cooperativa de Ahorro Crédito "Atuntaqui" Ltda.use el sistema de información implementado para alimentar de datos al sistema y en un futuro poder acceder a esta información y obtener datos estadísticos indispensables para el proceso de administración de tecnología.
- Brindar el mantenimiento necesario al sistema de información, para conservar la integridad de la información y que el sistema siempre este disponible para la administración de la tecnología.
- Realizar actualizaciones periódicas con los nuevos requerimientos.de los usuarios de manera que la herramienta de software se encuentre acorde a las necesidades de los usuarios y esta se convierte en una herramienta eficiente de soporte.
- Utilizar la herramienta de desarrollo JasperReports en nuestros proyectos de software con tecnología Java, para la generación de reportes; debido a que esta a parte de ser una herramienta de código abierto es un de las herramientas muy robustas para la creación de documentos.

7.4. Posibles temas de tesis

- Estudio de la plataforma Open Source Pentaho Business Intelligence, en el diseño de aplicaciones Java.
- Estudio de la herramienta de desarrollo JavaFX, en el diseño de Aplicaciones Ricas de Internet (RIA)
- Estudio del framework Java Server Faces en el diseño de aplicaciones Java JEE.
- Estudio de la herramienta Oracle Application Express (APEX).

GLOSARIO

A

ASE (Adaptive Server Enterprise): es el motor de bases de datos (RDBMS) insignia de la compañía Sybase. ASE es un sistema de gestión de datos, altamente escalable, de alto rendimiento, con soporte a grandes volúmenes de datos, transacciones y usuarios, y de bajo costo.

API (Application Programming Interface): Una interfaz de programación de aplicaciones es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

B

Bytecode: es un código intermedio más abstracto que el código máquina. Habitualmente es tratado como un fichero binario que contiene un programa ejecutable similar a un módulo objeto, que es un fichero binario producido por el compilador cuyo contenido es el código objeto o código máquina.

C

Compilación: Un compilador es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar. Usualmente el segundo lenguaje es lenguaje de máquina, pero también puede ser simplemente texto. Este proceso de traducción se conoce como compilación.

CSV (Comma-Separated Values): son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas (o punto y coma en donde la coma es el separador decimal: España, Francia, Italia...) y las filas por saltos de línea.

Clase: en la programación orientada a objetos es una construcción que se utiliza como un modelo (o plantilla) para crear objetos de ese tipo. El modelo describe el estado y el comportamiento que todos los objetos de la clase comparten.

D

DTD (Document Type Definition): es una descripción de estructura y sintaxis de un documento XML o SGML. Su función básica es la descripción del formato de datos, para usar un formato común y mantener la consistencia entre todos los documentos que utilicen la misma DTD.

E

Encapsulamiento: En programación modular, y más específicamente en programación orientada a objetos, se denomina encapsulamiento al ocultamiento del estado, es decir, de los datos miembro, de un objeto de manera que sólo se puede cambiar mediante las operaciones definidas para ese objeto.

F

File Transfer Protocol (FTP): Protocolo de transferencia de archivos. Protocolo TCP/IP que le permite copiar archivos de una computadora a otra en Internet. Con FTP, también se pueden renombrar o borrar archivos.

G

GNU (General Public License): es una licencia creada por la Free Software Foundation en 1989 orientada a proteger la libre distribución, modificación y uso de software.

H

Herencia: En orientación a objetos la herencia es el mecanismo fundamental para implementar la reutilización y extensibilidad del software. A través de ella los diseñadores pueden construir nuevas clases partiendo de una jerarquía de clases ya existente (comprobadas y verificadas) evitando con ello el rediseño, la modificación y verificación de la parte ya implementada. La herencia facilita la creación de objetos a partir de otros ya existentes, obteniendo características (métodos y atributos) similares a los ya existentes.

HTML (HyperText Markup Language): es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

I

Interfaz de usuario: es el medio con que el usuario puede comunicarse con una máquina, un equipo o una computadora, y comprende todos los puntos de contacto entre el usuario y el equipo. Normalmente suelen ser fáciles de entender y fáciles de accionar.

Iteración: es la repetición de una serie de instrucciones en un programa de computadora. Puede usarse tanto como un término genérico (como sinónimo de repetición) así como para describir una forma específica de repetición con un estado mutable.

J

JDBC (Java Database Connectivity): más conocida por sus siglas, es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo 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.

JasperReports: es una herramienta de creación de informes que tiene la habilidad de entregar contenido enriquecido al monitor, a la impresora o a ficheros PDF, HTML, XLS, CSV y XML.

Java: lenguaje de programación originalmente desarrollado por Sun Microsystems, adquirida por Oracle, para aplicaciones software independiente de la plataforma.

JVM (Java Virtual Machine): es un máquina virtual de proceso, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en bytecode, el cual es generado por el compilador del lenguaje Java

L

LDAP (Lightweight Directory Access Protocol): hace referencia a un protocolo a nivel de aplicación el cual permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red. LDAP también es considerado una base de datos (aunque su sistema de almacenamiento puede ser diferente) a la que pueden realizarse consultas.

Lenguajes de cuarta generación (4GL): son ciertas herramientas prefabricadas, que aparentemente dan lugar a un lenguaje de programación de alto nivel que se parece más al idioma inglés que a un lenguaje de tercera generación, porque se aleja más del concepto de "procedimiento". Pueden acceder a bases de datos.

N

Netscape Navigator: fue un navegador web y el primer producto comercial de la compañía Netscape Communications, creada por Marc Andreessen, uno de los autores de Mosaic, cuando se encontraba en el NCSA (Centro Nacional de Aplicaciones para Supercomputadores) de la Universidad de Illinois en Urbana-Champaign. Netscape fue el primer navegador comercial.

O

Open Source: es el término con el que se conoce al software distribuido y desarrollado libremente. El código abierto tiene un punto de vista más orientado a los beneficios prácticos de compartir el código que a las cuestiones éticas y morales las cuales destacan en el llamado software libre.

P

Parámetro: un argumento o parámetro es una variable que puede ser recibida por una rutina o subrutina. Una subrutina usa los valores asignados a sus argumentos para alterar su comportamiento en tiempo de ejecución. La mayor parte de los lenguajes de programación pueden definir subrutinas que aceptan cero o más argumentos.

PDF (Portable Document Format): formato de documento portátil) es un formato de almacenamiento de documentos, desarrollado por la empresa Adobe Systems. Este formato es de tipo compuesto (imagen vectorial, mapa de bits y texto).

Plataforma: es un sistema que sirve como base para hacer funcionar determinados módulos de hardware o de software con los que es compatible. Dicho sistema está definido por un estándar alrededor del cual se determina una arquitectura de hardware y una plataforma de software (incluyendo entornos de aplicaciones). Al definir plataformas se establecen los tipos de arquitectura, sistema operativo, lenguaje de programación o interfaz de usuario compatibles.

R

Runtime (Tiempo de Ejecución): Se denomina tiempo de ejecución al intervalo de tiempo en el que un programa de computadora se ejecuta en un sistema operativo. Este tiempo se inicia con la puesta en memoria principal del programa, por lo que el sistema operativo comienza a ejecutar sus instrucciones. El intervalo finaliza en el momento en que éste envía al sistema operativo la señal de terminación, sea ésta una terminación normal, en que el programa tuvo la posibilidad de concluir sus instrucciones satisfactoriamente, o una terminación anormal, en el que el programa produjo algún error y el sistema debió forzar su finalización.

S

Solaris: es un sistema operativo de tipo Unix desarrollado desde 1992 inicialmente por Sun Microsystems y actualmente por Oracle Corporation como sucesor de SunOS. Es un sistema certificado oficialmente como versión de Unix. Funciona en arquitecturas SPARC y x86 para servidores y estaciones de trabajo.

Servidor: es una computadora que, formando parte de una red, provee servicios a otras computadoras denominadas clientes.

Servlets: son objetos que corren dentro del contexto de un contenedor de servlets (ej: Tomcat) y extienden su funcionalidad.

SGML (Standard Generalized Markup Language): consiste en un sistema para la organización y etiquetado de documentos. El lenguaje SGML sirve para especificar las reglas de etiquetado de documentos y no impone en sí ningún conjunto de etiquetas en especial.

T

TCP/IP: Es un conjunto de protocolos de red en los que se basa Internet y que permiten la transmisión de datos entre computadoras.

Thread: En sistemas operativos, un hilo de ejecución, hebra o subproceso es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo.

U

Unicode: es un estándar de codificación de caracteres diseñado para facilitar el tratamiento informático, transmisión y visualización de textos de múltiples lenguajes y disciplinas técnicas además de textos clásicos de lenguas muertas. El término Unicode proviene de los tres objetivos perseguidos: universalidad, uniformidad y unicidad.

URL (Uniform Resource Locator): es una secuencia de caracteres, de acuerdo a un formato modélico y estándar, que se usa para nombrar recursos en Internet para su localización o identificación.

V

Variables: son espacios reservados en la memoria que, como su nombre indica, pueden cambiar de contenido a lo largo de la ejecución de un programa. Una variable corresponde a un área reservada en la memoria principal del ordenador pudiendo ser de longitud.

W

W3C: El World Wide Web Consortium, es un consorcio internacional que produce recomendaciones para la World Wide Web. Está dirigida por Tim Berners-Lee.

X

XLS (Microsoft Excel): es una aplicación para manejar hojas de cálculo. Este programa es desarrollado y distribuido por Microsoft, y es utilizado normalmente en tareas financieras y contables.

XML (eXtensible Markup Language): es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML).

BIBLIOGRAFÍA

LIBROS

- **[LIBRO 01]**
  BRUCE, Eckel. *Piensa en Java, 4ta Edición.*
  Madrid: Prentice Hall, 2007

- **[LIBRO 02]**
  HEFFELFINGER, David. *JasperReports 3.5 for Java Developers.*
  Birmingham: Packt Publishing, 2009

- **[LIBRO 03]**
  AHAMMAD, Shamsuddin. *iReport 3.7.*
  Birmingham: Packt Publishing, 2010

- **[LIBRO 04]**
  DANCIU, Teodor y CHIRITA, Lucian. *The Definitive Guide to JasperReports.*
  New York: Apress, 2007

- **[LIBRO 05]**
  TOFFOLI, Giulio. *The Definitive Guide to iReport.*
  New York: Apress, 2007

- **[LIBRO 06]**
  DANCIU, Teodor. *The JasperReports Ultimate Guide.*
  New York: Apress, 2007

- **[LIBRO 07]**
  Sybase Company. *Adaptive Server Enterprise 12.5, Manual de Instalación.*
  New York: Sybase Inc., 2001

INTERNET

- **[WWW01]**
  http://www.java.com/es
  SITIO OFICIAL DE JAVA

- **[WWW02]**
  http://jasperforge.org
  PROYECTO JASPERREPORTS E iREPORT

- **[WWW03]**
  http://www.netbeans.org
  SITIO OFICIAL IDE NETBEANS

- **[WWW04]**
  http://tomcat.apache.org
  CONTENEDOR DE SERVLETS TOMCAT

- **[WWW05]**
  http://www.sybase.com
  SISTEMA DE GESTION DE BASES DE DATOS SYBASE

- **[WWW06]**
  http://www.mtbase.com
  SISTEMA DE GESTION DE BASES DE DATOS SYBASE (ESPAÑOL)

- **[WWW07]**
  http://es.wikipedia.org
  LA ENCICLOPEDIA LIBRE

- **[WWW08]**
  http://www.programacion.net/java/tutorial/j2ee
  PROGRAMACION EN JAVA

- **[WWW09]**
  http://www.w3c.es/divulgacion/guiasbreves/tecnologiasxml
  PAGINA OFICAIAL DEL CONSORCIO W3C (ESPAÑA)

- **[WWW10]**
  http://www.cepeu.edu.py/LIBROS_ELECTRONICOS_3
  CENTRO DE ESPECIALIZACIÓN PROFESIONAL Y EXTENSIÓN UNIVERSITARIA

- **[WWW11]**
  http://download.oracle.com/javase/tutorial
  SITIO OFICIAL DE ORACLE CORPORATION

ANEXOS

SOFTWARE

- **CÓDIGO FUENTE SISTEMA SARTE**
  UBICACIÓN: \SARTE\CODIGO

- **NETBEANS**
  UBICACIÓN: \HERRAMIENTAS\NETBEANS

- **iREPORT**
  UBICACIÓN: \HERRAMIENTAS\iREPORT

- **JASPERREPORTS**
  UBICACIÓN: \HERRAMIENTAS\JASPERREPORTS

- **TOMCAT**
  UBICACIÓN: \HERRAMIENTAS\TOMCAT

MANUALES

- **MANUAL DE INSTALACIÓN**
  UBICACIÓN: \MANUALES

- **MANUAL DE USUARIO**
  UBICACIÓN: \MANUALES

- **MANUAL DE TÉCNICO**
  UBICACIÓN: \MANUALES