Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Prototipo de un Archivo de Documentos Digitales provenientes de la Web. Contenido Web y Almacenes de Datos

Trabajo Especial de Grado presentado ante la ilustre
Universidad Central de Venezuela
Por el Bachiller:
Carlos A. Blanco B. - C.I. 14.519.963
Para optar al título de Licenciado en Computación
Tutor: **Profa. Mercy Ospina**
Caracas, Julio 2012

UNIVERSIDAD CENTRAL DE VENEZUELA

FACULTAD DE CIENCIAS

ESCUELA DE COMPUTACIÓN

ACTA

Quienes suscriben, miembros del jurado designado por el Consejo de la Escuela de Computación, para examinar el Trabajo Especial de Grado titulado "Prototipo de un Archivo de Documentos Digitales provenientes de la Web. Contenido Web y Almacenes de Datos" y presentado por el Br. Carlos A. Blanco B., a los fines de optar al título de Licenciado en Computación, dejamos constancia de lo siguiente:

Leído como fue dicho trabajo, por cada uno de los miembros del jurado, se fijó el día 25 de Julio del 2012, a las 08:00 AM horas, para que los autores lo defendieran en forma pública, lo que estos hicieron en la Sala PBIII de la Escuela de Computación, mediante una presentación oral de su contenido, luego de lo cual respondieron a las preguntas formuladas. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobar con la nota de 20 puntos.

En fe de lo cual se levanta la presente Acta, en Caracas el día 25 de Julio del 2012.

_____

Prof(a). Mercy Ospina

(Tutora)

_____ _____

Prof(a). Tina Di Vasta

(Jurado)

Prof. Carlos Acosta

(Jurado)

RESUMEN

Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Laboratorio de Sistemas Paralelos y Distribuidos

Prototipo para un Archivo de Documentos Digitales provenientes de la Web. Contenido Web y Almacenes de Datos

Autor: Carlos A. Blanco B. – C.I. 14.519.963

Tutora: Profa. Mercy Ospina

Fecha: Junio de 2012

En la actualidad se puede observar que el mayor contenido de información existente en el mundo se encuentra en el internet, esta información que va creciendo de manera dinámica se actualiza en cada momento y puede tener una vida corta. Un gran problema que se puede presentar es que mucha información se puede perder por distintos motivos haciendo que no se pueda acceder a ella.

Teniendo en cuenta este problema, el proyecto en el cual se basa el presente Trabajo Especial de Grado propone salvaguardar esta información para que se pueda acceder a ella desde una base de datos documental (Base de Datos No Relacional Basada en Documentos) llamada MongoDB, y además crear un Almacén de Datos que va a ser alimentado con distinta información obtenida de las páginas web almacenada para poder lograr distintas métricas que se mostraran al usuario.

Este Trabajo Especial de Grado presenta el diseño de dicho Almacén de Datos dividido en dos módulos del proyecto general, estos son el proceso ETL (Extracción, Transformación y Carga) que se encarga de alimentar el Almacén de Datos, y la Interfaz el Usuario, en la que se muestra distintas métricas obtenidas del Almacén de Datos y la página web al usuario.

Palabras Claves: Almacenes de Datos (DW), Ruby on Rails, MongoDB, Metodologia XP

TABLA DE CONTENIDO

ÍNDICE DE CUADROS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Actualmente se observa que una gran parte de la información producida en casi todos los ámbitos de la actividad humana se almacena en medios digitales, usando como soporte los computadores y las redes. Sin embargo, la persistencia de la información no está garantizada de por sí, depende de factores humanos, técnicos y económicos para que sean preservados a lo largo del tiempo.

Imaginemos por un momento que las grandes bibliotecas del mundo permitieran que varias de sus obras fueran destruidas debido a falta de espacio, recorte presupuestario, accidentes o simplemente mala praxis. Muchas de las obras son preservadas para la consulta del público actual y para las generaciones futuras.

Al igual que el contenido en medios impresos, el digital es un resultado de la actividad intelectual humana, entonces ¿por qué no darle el mismo peso y valor a éstas contribuciones? es cierto que el manejo de colecciones de contenido digital se presenta en formatos y maneras diferentes lo que las hace difícilmente medibles, por lo tanto difíciles de gestionar.

Preservar esta información que incluye, por ejemplo, informaciones científicas, datos de investigación, productos de medios de comunicación, obras de artes digitales, entre otros; plantea retos y problemas que deben ser afrontados de manera integral desde un punto de vista humano y tecnológico.

Pensar en realizar un gran archivo general de la web supone la disponibilidad vasta de recursos de almacenamiento y de conectividad. Casos como los del sitio Internet Archive y el su proyecto de ésta (The WayBack Machine) han logrado desde 1996 almacenar gran parte del contenido web en el Internet. Otros trabajos se han enfocado en confeccionar archivos locales, por país, especializados en su cultura y necesidades de

preservación y que puedan interactuar con otros de una manera distribuida (Rauber, 2002).

Venezuela no es ajena a estos problemas, más cuando su desarrollo de contenidos en la web ha ido en aumento en los últimos años, hoy más que nunca es necesario preservar la información digital ya que se observa en ella registrada una buena parte de la historia y acontecimientos recientes del país.

Tomando en cuenta lo antes mencionado los Almacenes de Datos pueden ser utilizados para almacenar información obtenida de los contenidos web para así lograr distintas métricas para un futuro análisis por parte del usuario final.

Con el contenido web almacenado en la base de datos No relacional, basada en documentos (MongoDB) se puede llevar un historial de los cambios realizados sobre el contenido web, con esto el usuario puede observar la evolución del contenido web con respecto a su funcionalidad, diseño, entre otras. También tiene una gran ventaja que si la página deja de existir todavía puede ver las versiones anteriores de la página antes que dejara de existir.

Para facilitar la lectura del documento al lector, a continuación se muestra la estructura de los distintos capítulos en esta Tesis de Grado:

- **CAPITULO 1 - Planteamiento del Problema:** en este capítulo se describe el objetivo general, los objetivos específicos, la solución planteada para los distintos objetivos, y una descripción general de la solución.
- **CAPITULO 2 - Marco Conceptual:** se presenta las distintas definiciones básicas que sirven como bases para entender mucho mejor de lo que se trata esta Tesis de Grado.
- **CAPITULO 3 - Marco Tecnológico:** en este capítulo se explica de toda la tecnología utilizada para el desarrollo del sistema.
- **CAPITULO 4 - Marco Metodológico:** presenta la metodología utilizada para el desarrollo de este proyecto.

- **CAPITULO 5 - Marco aplicativo:** en este capítulo se explica la aplicación de la Metodología XP al caso de estudio.

Finalmente se presentan las Conclusiones, recomendaciones, apéndice y bibliografía consultada a lo largo de Trabajo Especial de Grado.

El presente trabajo se enmarca dentro del proyecto de investigación financiado por el Consejo de Desarrollo Científico y Humanístico (CDCH) de la Universidad Central de Venezuela (UCV). Proyecto No. PG-037353-2008

CAPITULO 1 - PLANTEAMIENTO DEL PROBLEMA

1.1 Titulo

Prototipo para un Archivo de Documentos Digitales provenientes de la Web. Contenido Web y Almacenes de Datos.

1.2 Planteamiento del Problema

En la actualidad, una de las herramientas con mayor auge de utilización a nivel mundial a distancia es la gran red de comunicaciones llamada Internet. Esta se ha convertido en un medio idóneo para mantener un contacto en tiempo real, tanto a nivel empresarial como personal entre sus usuarios, además de un funcionamiento tangible en el ámbito de información, educación, comercialización y entretenimiento.

Esta gran red de información se presenta con una amplia gama de sitios Web donde se puede indagar temas de cualquier índole. Se tiene por ejemplo: sitios de descargas de archivos, blogs, sitios de información de empresas, sitios de comercio electrónico, wiki, sitios de comunidades virtuales, sitios de juegos, entre otros.

Varios son los esfuerzos para preservar documentos, libros, entre otros, para su consulta pública y para las generaciones futuras. Cada día más información se encuentra en formato digital, las noticias se generan y se publican en la Web. Preservar este tipo de información es una necesidad para el mundo. Esta preservación de información digital se denomina Archivo Web, que también permiten el acceso a versiones anteriores de páginas Web que ya no existen.

En el marco de las directrices para la preservación del patrimonio digital de la UNESCO (2003) sobre la digitalización y la accesibilidad en línea del material cultural y la conservación digital, se encuentra el proyecto del Centro de Computación Paralela y Distribuida (CCPD) el cual cubre el desarrollo de un prototipo para la Construcción del Archivo de Documentos Digitales provenientes de la Web, con el fin de conservar los

productos culturales “archivos digitales”, para que puedan convertirse en un patrimonio cultural e intelectual perdurable para las generaciones presentes y venideras de Venezuela.

1.3 Objetivo General y Específico

En la siguiente sección se va a definir los objetivos generales y especifico para el desarrollo de esta Tesis Especial de Grado

1.3.1 Objetivo General

- Diseñar e implementar un Almacén de Datos que facilite el análisis de las distintas métricas obtenidas de la información almacenada de las versiones de documentos provenientes de la web además se implementa una aplicación web para el acceso a estos documentos y sus métricas al usuario final.

1.3.2 Objetivos Específicos

- Diseñar un Almacén de Datos que se adapte mas a la estructura de la información disponible en la web y así poder obtener fácilmente las distintas métricas que se van a mostrar al usuario final

  Actividades:

  - Diseñar un Almacén de Datos en base a las características de los documentos web descritos en el estándar de HTML (Lenguaje de Marcado de Hipertexto).
  - Implementar el Almacén de Datos usando MonetDB.
- Diseñar y construir un proceso ETL (Extraccion, Transformacion y Carga) para introducir la información o datos obtenidos desde la web.

  Actividades:

  - Implementar el proceso ETL (Extraccion, Transformacion y Carga) con Ruby para MonetDB.

  - Extraer la información necesaria para alimentar el Almacén de Datos desde la base de datos iticve_db en MongoDB.

- Ejecutar el proceso ETL con un conjunto de datos previamente definidos.

  Actividades:

  - Seleccionar previamente el conjunto de datos que se va a cargar en el Almacén de Datos.
  - Probar los resultados obtenidos para ver si son los resultados esperados.

- Diseñar y desarrollar una aplicación Web para que el usuario pueda acceder a los documentos web almacenado y a las métricas que son obtenidas del Almacén de Datos.

  Actividades:

  - Implementar la aplicación web usando Ruby on Rails.
  - Extraer información del Almacén de Datos según la URL (Identificador Uniforme de Recurso) de la página web que ingrese el usuario.
  - Mostrar un calendario con las distintas versiones almacenada de la página solicitada por el usuario.
  - Al seleccionar una fecha en calendario se procede a extraer el contenido completo de la página web desde MongoDB y finalmente se le muestra al usuario los datos extraídos.

1.4 Descripción de la solución

Partiendo de los documentos almacenados en MongoDB se procede a extraer los datos con el proceso ETL, el proceso ETL solamente extrae la metadata de los nuevos documentos y los viejos ítems que ya están almacenados en el Almacén de Datos los ignora.

De los documentos web almacenados en MongoDB podemos obtener los siguientes elemetnos: URI (Identificador Uniforme de Recurso), Mime, Información para las Métricas, MongoID. Estos atributos se procesan y se almacenan en el Almacén de Datos (MonetDB).

Finalmente en la interfaz de usuario, el usuario ingresa una URI válida y se procede a buscar las distintas versiones almacenada de la URI ingresada, si se encuentran alguna versión se extrae todos los datos necesario para mostrar esa página web con todo su contenido.

Al tener los datos de la página se le muestra un calendario al usuario donde puede seleccionar las distintas versiones de la página, estas versiones se muestran en el calendario según el día, mes, año, hora, minuto y segundo en la cual fue almacenada la información de la página en el Almacén de Datos. Cada vez que se inserta nuevos datos en el Almacén de Datos se almacena la fecha y hora en que se almaceno. Finalmente cuando el usuario selecciona una versión de la página se extrae los archivos necesarios para poder mostrar la página y se le muestra al usuario.

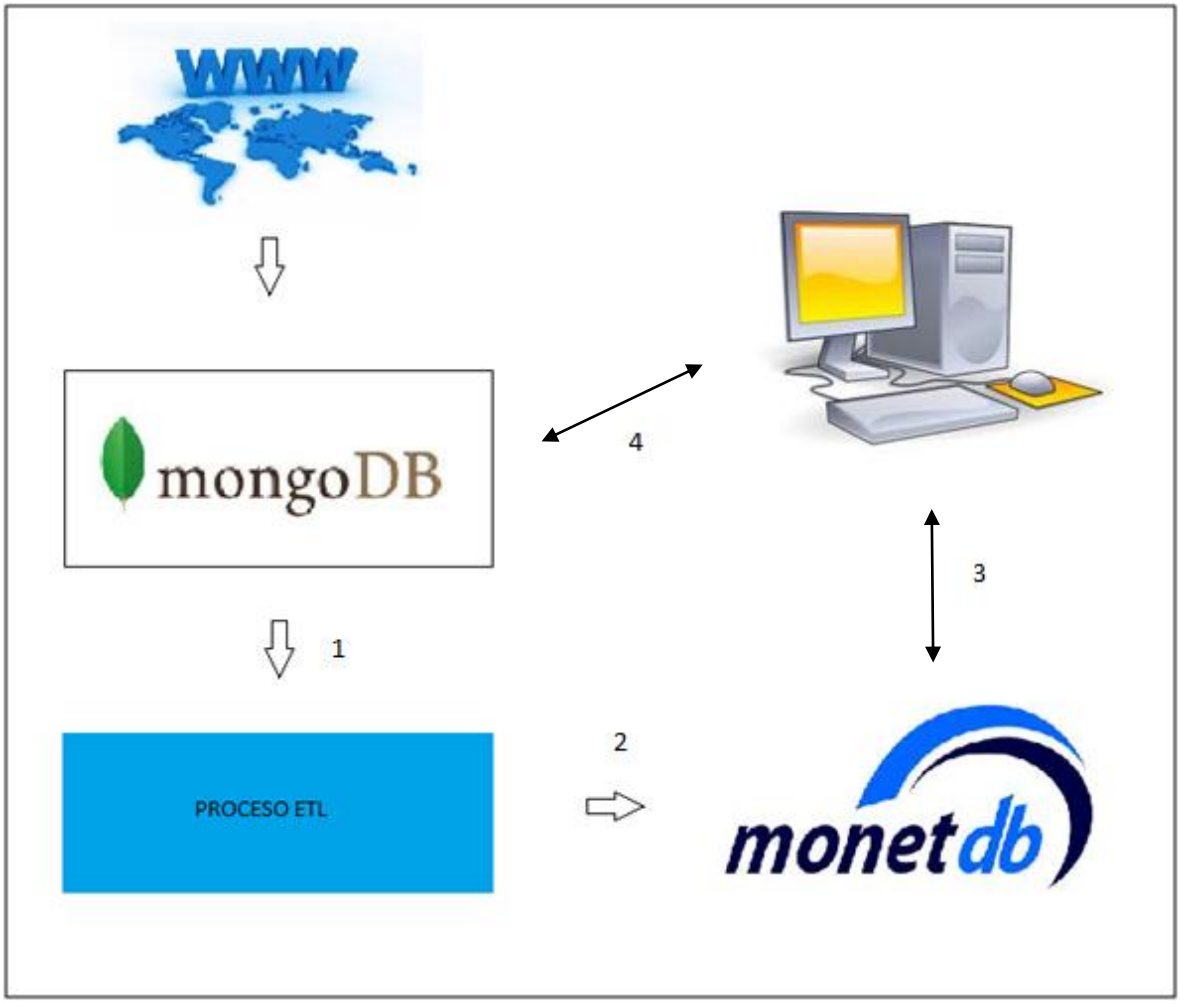

Figura 1 – Proceso de las Funcionalidades del Sistema

A continuación se muestra el proceso (Figura 1):

1. Extraer datos necesario del MongoDB para almacenar en el Almacén de Datos
2. Al Tener los datos procesados se almacenan en el Almacén de Datos.
3. Buscar el URL ingresado por el usuario en el Almacén de Datos para obtener los datos necesarios de la página web.
4. Extraer archivos necesarios para poder mostrar la página según los datos extraídos del Almacén de Datos.

CAPITULO 2 - MARCO CONCEPTUAL

2.1 Preservación Digital

La preservación digital es el conjunto de medidas, que se establecen con el fin de asegurar la perdurabilidad de la capacidad de visualizar, recuperar y utilizar colecciones digitales frente a las infraestructuras, elementos tecnológicos y de organización que cambian con mucha rapidez.

La preservación digital, plantea nuevos desafíos para la disciplina de la conservación. Si tomamos en consideración que la preservación está orientada a largo plazo, nos encontramos con un primer problema:

La información digital, no ha sido pensada para su mantenimiento en el tiempo. A diferencia de materiales tradicionales, que tienden a ir desapareciendo de manera gradual, desvaneciéndose o amarilleándose con el paso del tiempo la información digital no se pierde de manera paulatina, esta existe o simplemente no existe. (Swiatecka, 2008)

2.1.1 Amenazas Técnicas

- La obsolencia del **hardware**
    - Una computadora tiene una estimación de vida de 3 a 5 años. No sólo porque técnicamente puede empezar a fallar, sino sobre todo porque nuevo software exige memorias más grandes, más velocidad, entre otras.
    - Problema general de la obsolescencia planificada de las mercancías.

- La obsolescencia del **software**
    - Un documento digital sólo existe mediante su recuperación: es inseparable del programa necesario para abrirlo.
    - Debido a la evolución de organizaciones, el software deja de satisfacer los requerimientos del usuario, e induce a una de las principales causas del desplazamiento del software.

- La obsolescencia de los **formatos** digitales.
    - La evolución de formatos está estrechamente vinculada con la evolución de los programas de software. En la mayoría de los casos no hay **compatibilidad** hacia atrás (o una compatibilidad deficiente) de los nuevos formatos.
    - Es política general de algunos desarrolladores de software propietario hacer que las nuevas versiones de sus programas almacenen los datos en formatos que no pueden ser leídos por versiones anteriores.

- Envejecimiento o deterioro físico de los **soportes** digitales.
    - Puede ser inherente al soporte, por agentes externos como daños físicos al soporte, almacenamiento incorrecto o porque el soporte se queda obsoleto (Swiatecka, 2008).

2.1.2 Amenazas dentro de la Organización

- La **seguridad** de la colección (por ejemplo: alteraciones no autorizada de la colección)
- Brechas en la **memoria** institucional debido a la rotación de personal
- Mantenimiento de registro y metadatos administrativos **inadecuados**

2.1.3 Plan de Preservación

- Las estrategias tienen que ser bien definidas.
- Con reglas y procedimientos a seguir.
- Requieren de un esfuerzo periódico y planificado.
- Los asuntos asociados con la larga duración necesitan ser discutidos desde el comienzo de cualquier iniciativa de digitalización de objetos.
- Las soluciones exitosas requieren la integración de consideraciones administrativas y técnicas.
- La responsabilidad debe exponerse explícita y seriamente.

- Esta política debe ser revisada periódicamente (Swiatecka, 2008).

2.1.4 Dificultades

- Compromiso institucional de preservación a largo plazo insuficiente y falta de políticas y procedimientos de preservación.
- Escasez de recursos humanos y financieros.
- Intereses variables (y asincrónicos) de quienes participan en la creación, mantenimiento y distribución de colecciones de documentos digitales.
- No existe un Plan de Conservación del Patrimonio Digital nacional, como en otros países (Pandora, DNEP, KulturaW3, entre otras) (Swiatecka, 2008).

Después de que se hablo explica la Preservación Digital, se explicara de los que es el Internet Archive que trata de la Preservación Web y se dará unos ejemplos de páginas que se dedican a la Preservación Web

2.2 Internet Archive

El Internet Archive Wayback Machine es un servicio que permite a la gente a visitar las versiones archivadas de sitios Web. Por dar un ejemplo los visitantes de la Wayback Machine pueden escribir una dirección URL, seleccione un intervalo de fechas, y luego comenzar a navegar en una versión archivada de la Web (Archive, 2001).

2.2.1 Origen

Creada en 1996, se encuentra en el Presidio de San Francisco (California). La colaboración que llevó a cabo la amplitud de esa página fue gracias a Alexa Internet y de otros colaboradores aparte de la colaboración de la Biblioteca del Congreso. Tiene una gran cantidad de archivos misceláneos como audio, video y texto, algunos de ellos en dominio público, o de licencias a base del Creative Commons o cualquier otra licencia que permita la distribución (Archive, 2001).

Ah continuación se va hablar de 3 distintas Archive Web conocidas en el mundo, primero comenzamos con las que se uso como base para este proyecto:

2.2.2 Ejemplos de Archivos Web

2.2.2.1 The Wayback Machine

Wayback Machine es un sitio que se dedica a archivar Internet. Es decir, rastrea Internet cada cierto tiempo, como lo hacen los buscadores tipo Google, y copia todas las webs que encuentra y que son públicamente accesibles, formando una gigantesca biblioteca de webs, permitiendo su acceso de manera gratuita.

Sólo archiva los sitios que Alexa conoce. No se pueden buscar palabras clave dentro del archivo, sino solamente URL de direcciones, y seleccionar la fecha. Las webs que copian tardan un tiempo en aparecer en el archivo. No es instantáneo.

El rastreo de Internet respeta las directivas de exclusión de robots, por lo que quien no quiere aparecer en el archivo le basta con configurar de manera adecuada el fichero robots.txt de su web. O solicitar después que le borren del archivo.

Wayback Machine es parte de Internet Archive, una biblioteca digital sin ánimo de lucro de San Francisco (EE.UU.), dedicada a ofrecer acceso público general gratis de páginas webs, música, películas y libros digitales. Trabaja en colaboración con la Universidad de Toronto (Canadá) y más de 150 bibliotecas y universidades de todo el mundo (qbitácora, 2010).

Utilidad:

- Acceder a webs que ya no existen, como por ejemplo, las alojadas en el hospedaje Gocetes, que cerró el 26-10-2009.
- Examinar cómo era una web en el pasado, cómo ha cambiado.
- Investigar sobre algo del pasado.

A veces no se puede o casi no se puede acceder a la dirección raíz de una web, pero sí a alguna dirección interior, y navegar a partir de esa dirección por el sitio tal y como era hace años (qbitácora, 2010).

Defectos:

- A veces falla, y el sitio buscado no fue archivado.
- A veces funciona muy lento. Normal, porque es gigantesco.
- Tampoco lo almacena todo: La funcionalidad dinámica hecha con Java Script, formularios, o bases de datos (la web profunda) no se guardan, por lo que puede que se vea distinto de cómo era la web original.

Pero el texto está ahí, guardado, y también fotos. Incluso a veces se ah llegado a descargar algún vídeo. Siempre que una web o una dirección ya no existan, se prueba a ver si se encuentra en the Wayback Machine, y esto resulta muy útil. Sin embargo, mucha gente todavía no la conoce (qbitácora, 2010).

2.2.2.2 Archivo Web de Cataluña

PADICAT es el archivo web de Cataluña, creado por la institución pública responsable de recoger, conservar y difundir el patrimonio bibliográfico de Cataluña, la Biblioteca de Catalunya.

La información sobre PADICAT está disponible:

- En la información institucional del repositorio, que contiene explicaciones sobre los objetivos, la historia y el equipo de PADICAT.
- En la radiografía de las páginas web integradas a PADICAT.
- En los datos de hardware y software del repositorio.
- En las últimas novedades y las apariciones de PADICAT en los medios de comunicación.
- En la información para periodistas en la sala de prensa.

- En los contenidos para profesionales de bibliotecas, archivos y museos (Padicat, 2011).

¿Qué es padicat?

PADICAT (Patrimoni Digital de Catalunya) es una iniciativa de la Biblioteca de Catalunya para capturar, preservar y difundir el archivo web de Cataluña.

La Biblioteca de Catalunya, responsable de capturar, conservar y difundir el patrimonio bibliográfico de Cataluña, y por extensión el patrimonio digital, cuenta con la colaboración tecnológica del Centre de Serveis Científics i Acadèmics de Catalunya (CESCA) para preservar y dar acceso a versiones antiguas de páginas web publicadas en Internet, des de 2005 (Padicat, 2011).

Misión y Objetivos

La misión del PADICAT es capturar, conservar y difundir el patrimonio digital de Cataluña nacido en Internet.

El sistema se basa en la aplicación de una serie de programas informáticos que permiten la captura, el almacenamiento, la organización y el acceso permanente a las páginas web publicadas en Internet.

Sus objetivos son:

- Compilar masivamente el dominio .cat.
- Impulsar el depósito sistemático de la producción web de las entidades y las empresas de Cataluña.
- Promover líneas de investigación procesando de manera monográfica los recursos de eventos de la vida pública catalana, como campañas electorales en Internet, el fenómeno de la música en línea, o los museos en Internet.

Después de unas etapas de nacimiento (2005-2006), crecimiento (2007-2008) y consolidación (2009-2011), a partir del 2012 se persigue sistematizar la capacidad de crecimiento con la meta de incorporar anualmente unas 75.700 versiones de aproximadamente 32.000 páginas web, procedentes de:

- Compilación semestral de 30.000 recursos del dominio .cat.
- Compilación semestral de 550 recursos de las 450 entidades con las que se ha llegado a un convenio de cooperación.
- Compilación semestral de los 800 recursos procedentes de recomendaciones de los usuarios.
- Compilación única de 1.000 recursos de colecciones monográficas.
- Compilación diaria de una parte sustancial de 30 publicaciones seriadas en línea.

A estas metas concretas se añaden cuatro ejes permanentes de trabajo:

- Definición de las estrategias de preservación digital para el patrimonio nacido en Internet. PADICAT proporciona radiografías periódicas de la web catalana; detecta los formatos que experimentan a corto plazo problemas de ilegibilidad; identifica los lenguajes más usados, etc.
- Impulso a líneas de investigación a partir de la creación de colecciones monográficas que cuentan con la implicación de expertos de cada materia.
- Creación y mantenimiento de la hemeroteca digital en Internet, con la captura sistematizada de publicaciones digitales en serie. Actualmente, con una muestra representativa en cuanto a tipos y contenidos, seleccionando las nacidas digitales, sin equivalente analógico.
- Cooperación con otros archivos web y depósitos de preservación digital, de bibliotecas, archivos y museos, para dar una respuesta eficiente a los retos de preservación digital y acceso a los recursos depositados (Padicat, 2011).

2.2.2.3 Pandora Archivo Web de Australia

El proyecto está liderado por la National Library of Australia (Australia). Se inicia en el año 1996 y sigue el modelo selectivo. Su alcance se centra en la selección de publicaciones en línea y webs sobre Australia, de autor australiano o sobre tema australiano. La catalogación es exhaustiva y les posibilidades de búsqueda, muy avanzadas. Dispone de un software propio, *Pandas*, que se ha implementado en otros proyectos.

El archivo web de Australia, Pandora, fue creado en 1996 por la National Library of Australia para garantizar el acceso permanente a una selección de publicaciones en línea y sedes web de y sobre Australia.

A falta de una ley que regule el depósito legal digital (la vigente es de 1968), la política de la biblioteca y los socios de proyecto, que forman el comité científico de la política selectiva, es llegar a acuerdos con las entidades editoras de los documentos susceptibles de ser capturados. Existe una guía publicada con los criterios de selección de las sedes capturadas. Los datos estadísticos de septiembre de 2005 muestran que el archivo contiene 27 millones de ficheros y tiene un crecimiento mensual de 30 GB. Es consultable en línea.

Los inconvenientes del sistema australiano están relacionados con su propia naturaleza: el criterio de la selección es forzosamente subjetivo, pese a la transparencia de la política de selección. El contexto (los enlaces a los cuales apunta el recurso), quedan desligados del documento, porque pueden no estar incluidos en la selección. Finalmente, el coste de tratamiento (selección, captura periódica, catalogación, etc.) de cada ítem es muy elevado.

Por contra, los beneficios se concentran en la calidad del tratamiento y la presentación del patrimonio. La accesibilidad en línea, en abierto, es posible por los acuerdos subscritos con los productores (que comporta el acceso a los recursos de la infranet). Los datos de la catalogación son compartibles con el resto de equipamientos

australianos (o internacionales). Se procura un crecimiento temático equilibrado de la colección.

Vistos los modelos integral y selectivo, la tercera vía a considerar es la mixta. Como se ha mencionado, buena parte de los depósitos digitales nacionales planteados inicialmente como integrales han ido adoptando medidas para incluir recursos muy significativos, como publicaciones periódicas, en sus fondos (Fonollosa, 2005).

2.3 Almacenes de Datos

El término **Almacenes de Datos** fue introducido por *Bill Inmon* a principios de la década de los ´90, quien lo definió como:

"Es una colección de datos orientado a temas, integrado, variable en el tiempo y no volátil para ayudar al proceso de toma de decisiones gerenciales".

También *Ralph Kimball* define **Almacenes de Datos** de una forma más sencilla y práctica pero igual de importante:

"Es una copia de los datos transaccionales específicamente estructurados para consultas y análisis".

Un **Almacén de Datos** es una **base de datos orientada al análisis** de la información histórica contenida en ella. Dependiendo las necesidades de análisis de la organización puede almacenarse desde unos meses hasta varios años de información. El modelo que soporta la información que contiene se encuentra diseñado, estructurado e implementado con la finalidad y propósito del análisis y navegación de los datos. Se entiende por navegación de los datos, la posibilidad de ver información correspondiente a diferentes contextos o entornos, por ejemplo, analizar las ventas anuales, trimestrales, mensuales y poder "abrirlas" según zonas, vendedor o clientes (Bi-Argentina, 2008).

2.3.1 Objetivos de los Almacenes de Datos

1. **Hace que la información de la organización sea accesible:** los contenidos de los Almacenes de Datos son entendibles y navegables, y el acceso a ellos son caracterizado por el rápido desempeño. Estos requerimientos no tienen fronteras y tampoco limites fijos. Cuando hablamos de entendible significa, que los niveles de la información sean correctos y obvios. Y Navegables significa el reconocer el destino en la pantalla y llegar a donde queramos con solo un clic. Rápido desempeño significa, cero tiempos de espera. Todo lo demás es un compromiso y por consiguiente algo que queremos mejorar.
2. **Hacer que la información de la organización sea consistente:** la información de una parte de la organización puede hacerse coincidir con la información de la otra parte de la organización. Si dos medidas de la organización tienen el mismo nombre, entonces deben significar la misma cosa. Y a la inversa, si dos medidas no significan la misma cosa, entonces son etiquetados diferentes. Información consistente significa, información de alta calidad. Significa que toda la información es contabilizada y completada. Todo lo demás es un compromiso y por consiguiente algo que queremos mejorar.
3. **Es información adaptable y elástica:** el Almacén de Datos está diseñado para cambios continuos. Cuando se le hacen nuevas preguntas al Almacén de Datos, los datos existentes y las tecnologías no cambian ni se corrompen. Cuando se agregan datos nuevos al Almacén de Datos, los datos existentes y las tecnologías tampoco cambian ni se corrompen. El diseño de Data Marts separados que hacen al Almacén de Datos, deben ser distribuidos e incrementados. Todo lo demás es un compromiso y por consiguiente algo que queremos mejorar.
4. **Es un seguro baluarte que protege los valores de la información:** el Almacén de Datos no solamente controla el acceso efectivo a los datos, si no que da a los dueños de la información gran visibilidad en el uso y abusos de los datos, aún después de haber dejado el Almacén de Datos. Todo lo demás es un compromiso y por consiguiente algo que queremos mejorar.

5. **Es la fundación de la toma de decisiones:** el Almacén de Datos tiene los datos correctos para soportar la toma de decisiones. Solo hay una salida verdadera del Almacén de Datos: las decisiones que son hechas después de que el Almacén de Datos haya presentado las evidencias. La original etiqueta que preside el Almacén de Datos sigue siendo la mejor descripción de lo que queremos construir: un sistema de soporte a las decisiones. (Herrera, 2007).

2.3.2 Características de los Almacenes de Datos

Orientado al tema: una primera característica de los Almacenes de Datos es que la información se clasifica en base a los aspectos que son de interés para la empresa. Siendo así, los datos tomados están en contraste con los clásicos procesos orientados a las aplicaciones.

El ambiente operacional se diseña alrededor de las aplicaciones y funciones tales como préstamos, ahorros, tarjeta bancaria y depósitos para una institución financiera. Por ejemplo, una aplicación de ingreso de órdenes puede acceder a los datos sobre clientes, productos y cuentas. La base de datos combina estos elementos en una estructura que acomoda las necesidades de la aplicación.

En el ambiente de Almacenes de Datos se organiza alrededor de sujetos tales como cliente, vendedor, producto y actividad. Por ejemplo, para un fabricante, éstos pueden ser clientes, productos, proveedores y vendedores. Para una universidad pueden ser estudiantes, clases y profesores. Para un hospital pueden ser pacientes, personal médico, medicamentos, entre otros.

La alineación alrededor de las áreas de los temas afecta el diseño y la implementación de los datos encontrados en los Almacenes de Datos. Las principales áreas de los temas influyen en la parte más importante de la estructura clave.

Las aplicaciones están relacionadas con el diseño de la base de datos y del proceso. En el Almacén de Datos se enfoca el modelamiento de datos y el diseño de la base de datos. El diseño del proceso (en su forma clásica) no es separado de este ambiente.

Las diferencias entre la orientación de procesos y funciones de las aplicaciones y la orientación a temas (ver figura 2), radican en el contenido de la data a escala detallada. En los Almacenes de Datos se excluye la información que no será usada por el proceso de sistemas de soporte de decisiones, mientras que la información de las orientadas a las aplicaciones, contiene datos para satisfacer de inmediato los requerimientos funcionales y de proceso, que pueden ser usados o no por el analista de soporte de decisiones.

Otra diferencia importante está en la interrelación de la información. Los datos operacionales mantienen una relación continua entre dos o más tablas basadas en una regla comercial que está vigente. Las de los Almacenes de Datos miden un espectro de tiempo y las relaciones encontradas en el Almacén de Datos son muchas. Muchas de las reglas comerciales (y sus correspondientes relaciones de datos) se representan en el Almacén de Datos, entre dos o más tablas (Herrera, 2007).

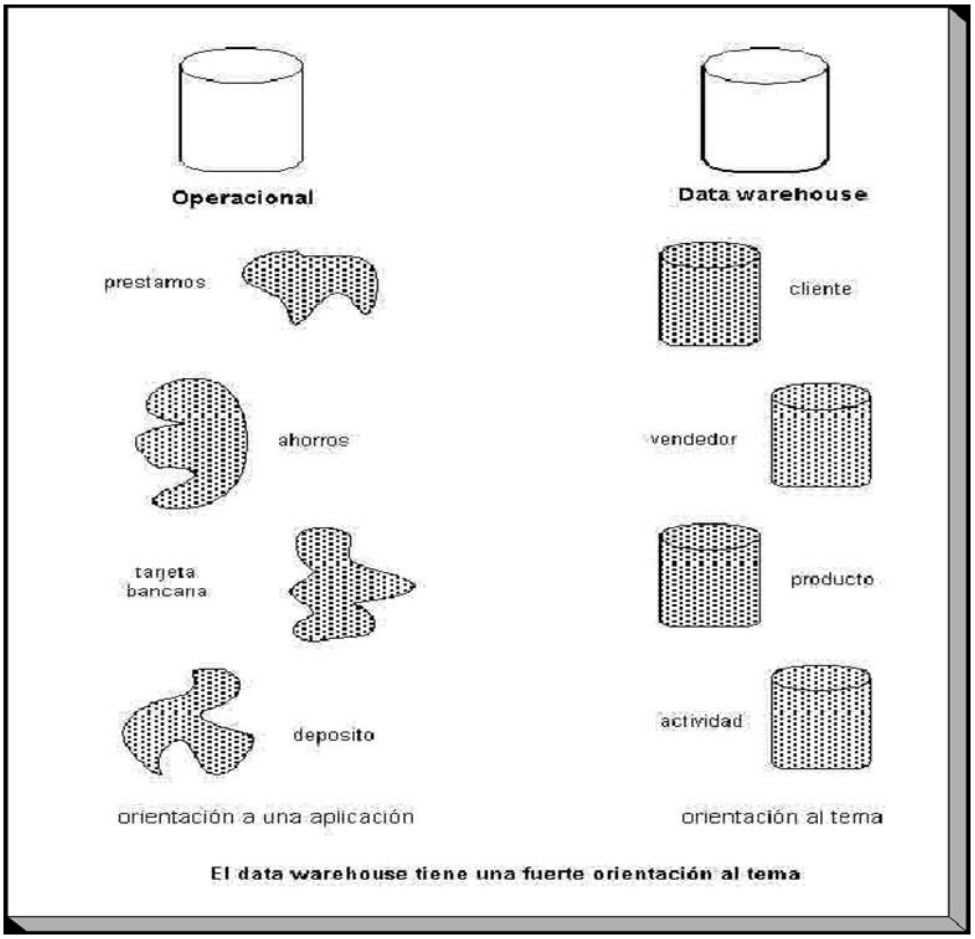

Figura 2 – Almacenes de Datos Orientados a Temas

Integrado: el aspecto más importante del ambiente de Almacenes de Datos es que la información encontrada al interior está siempre integrada.

La integración de datos se muestra de muchas maneras: en convenciones de nombres consistentes, en la medida uniforme de variables, en la codificación de estructuras consistentes, en atributos físicos de los datos consistentes, fuentes múltiples y otros.

El contraste de la integración encontrada en los Almacén de Datos con la carencia de integración del ambiente de aplicaciones, se muestran en la figura 3, con diferencias bien marcadas.

A través de los años, los diseñadores de las diferentes aplicaciones han tomado sus propias decisiones sobre cómo se debería construir una aplicación. Los estilos y diseños personalizados se muestran de muchas maneras.

Se diferencian en la codificación, en las estructuras claves, en sus características físicas, en las convenciones de nombramiento y otros. La capacidad colectiva de muchos de los diseñadores de aplicaciones, para crear aplicaciones inconsistentes, es asombrosa.

- **Codificación**. Los diseñadores de aplicaciones codifican el campo GÉNERO en varias formas. Un diseñador representa GÉNERO como una "M" y una "F", otros como un "1" y un "0", otros como una "X" y una "Y" e inclusive, como "masculino" y "femenino". No importa mucho cómo el GÉNERO llega al Almacén de Datos. Probablemente "M" y "F" sean tan buenas como cualquier otra representación. Lo importante es que sea de cualquier fuente de donde venga, el GÉNERO debe llegar al Almacén de Datos en un estado integrado uniforme.

  Por lo tanto, cuando el GÉNERO se carga en el Almacén de Datos desde una aplicación, donde ha sido representado en formato "M" y "F", los datos deben convertirse al formato del Almacén de Datos.
- **Medida de atributos**. Los diseñadores de aplicaciones miden las unidades de medida de las tuberías en una variedad de formas. Un diseñador almacena los datos de tuberías en centímetros, otros en pulgadas, otros en millones de pies cúbicos por segundo y otros en yardas. Al dar medidas a los atributos, la transformación traduce las diversas unidades de medida usadas en las diferentes bases de datos para transformarlas en una medida estándar común. Cualquiera que sea la fuente, cuando la información de la tubería llegue al Almacén de Datos necesitará ser medida de la misma manera.
- **Convenciones de Nombramiento**. El mismo elemento es frecuentemente referido por nombres diferentes en las diversas aplicaciones. El proceso de transformación asegura que se use preferentemente el nombre de usuario.

- **Fuentes Múltiples**. El mismo elemento puede derivarse desde fuentes múltiples. En este caso, el proceso de transformación debe asegurar que la fuente apropiada sea usada, documentada y movida al depósito. Los puntos de integración afectan casi todos los aspectos de diseño - las características físicas de los datos, la disyuntiva de tener más de una de fuente de datos, el problema de estándares de denominación inconsistentes, formatos de fecha inconsistentes y otros. Cualquiera que sea la forma del diseño, el resultado es el mismo, la información necesita ser almacenada en el Almacén de Datos en un modelo globalmente aceptable y singular, aun cuando los sistemas operacionales subyacentes almacenen los datos de manera diferente. Cuando el analista de sistema de soporte de decisiones observe el Almacén de Datos, su enfoque deberá estar en el uso de los datos que se encuentre en el depósito, antes que preguntarse sobre la confiabilidad o consistencia de los datos (Herrera, 2007).

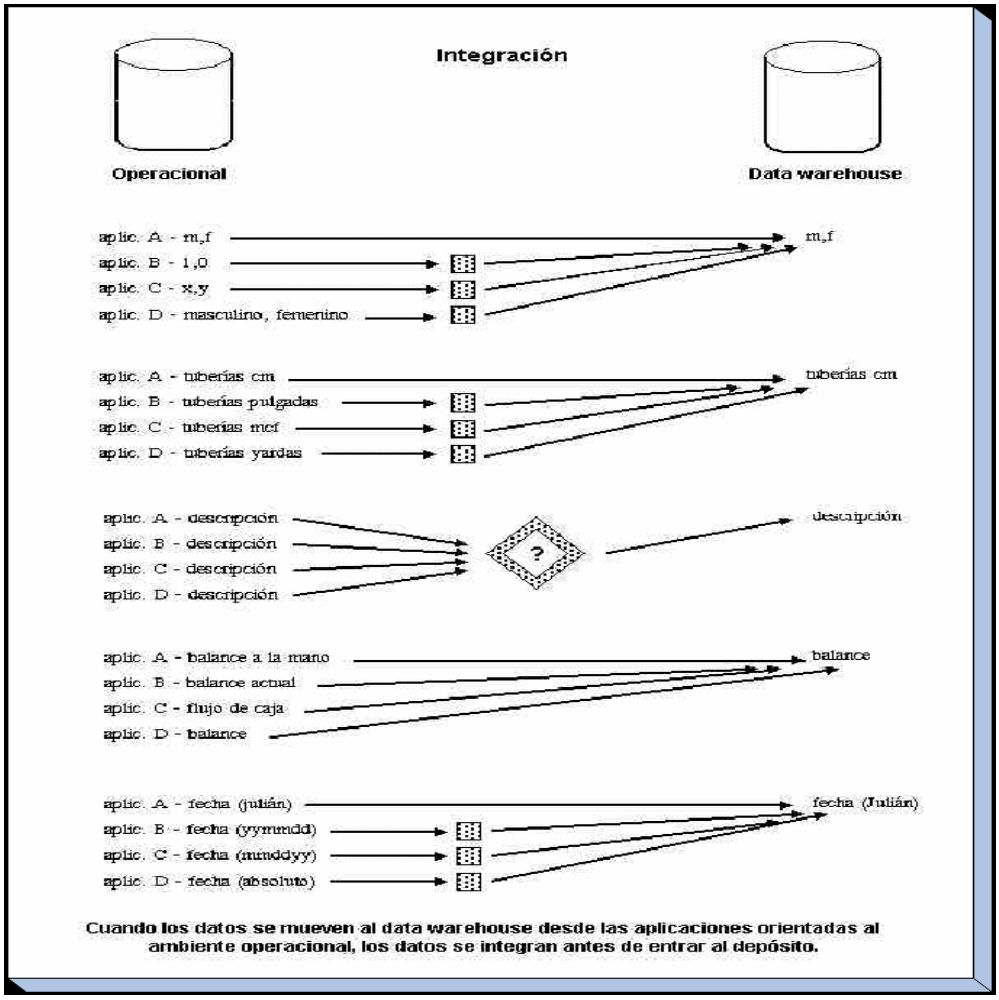

Figura 3 - Almacenes de Datos Integración

Variable en el tiempo: los datos son relativos a un periodo de tiempo y estos deben ser integrados periódicamente, los mismos son almacenados como fotos que se corresponden a un periodo de tiempo.

Toda la información del Almacén de Datos es requerida en algún momento. Esta característica básica de los datos en un depósito, es muy diferente de la información encontrada en el ambiente operacional. En éstos, la información se requiere al momento de acceder. En otras palabras, en el ambiente operacional, cuando se accede a una unidad de información, se espera que los valores requeridos se obtengan a partir del momento de acceso.

Como la información en el Almacén de Datos es solicitada en cualquier momento (es decir, no "ahora mismo"), los datos encontrados en el depósito se llaman de "tiempo variante".

Los datos históricos son de poco uso en el procesamiento operacional. La información del depósito por el contraste, debe incluir los datos históricos para usarse en la identificación y evaluación de tendencias.

El tiempo variante se muestra de varias maneras (ver Figura 4):

- La más simple es que la información representa los datos sobre un horizonte largo de tiempo, desde cinco a diez años. El horizonte de tiempo representado para el ambiente operacional es mucho más corto, desde valores actuales hasta sesenta a noventa días. Las aplicaciones que tienen un buen rendimiento y están disponibles para el procesamiento de transacciones, deben llevar una cantidad mínima de datos si tienen cualquier grado de flexibilidad. Por ello, las aplicaciones operacionales tienen un corto horizonte de tiempo, debido al diseño de aplicaciones rígidas.
- La segunda manera en la que se muestra el tiempo variante en el Almacén de Datos está en la estructura clave. Cada estructura clave en el Almacén de Datos contiene, implícita o explícitamente, un elemento de tiempo como día, semana, mes, etc. El elemento de tiempo está casi siempre al pie de la clave concatenada, encontrada en el Almacén de Datos. En ocasiones, el elemento de tiempo existirá implícitamente, como el caso en que un archivo completo se duplica al final del mes, o al cuarto.
- La tercera manera en que aparece el tiempo variante es cuando la información del Almacén de Datos, una vez registrada correctamente, no puede ser actualizada. La información del Almacén de Datos es, para todos los propósitos prácticos, una serie larga de "snapshots" (vistas instantáneas). Por supuesto, si los snapshots de los datos se han tomado incorrectamente, entonces pueden ser cambiados. Asumiendo que los snapshots se han tomado adecuadamente, ellos no son

alterados una vez hechos. En algunos casos puede ser no ético, e incluso ilegal, alterar los snapshots en el Almacén de Datos. Los datos operacionales, siendo requeridos a partir del momento de acceso, pueden actualizarse de acuerdo a la necesidad (Herrera, 2007).

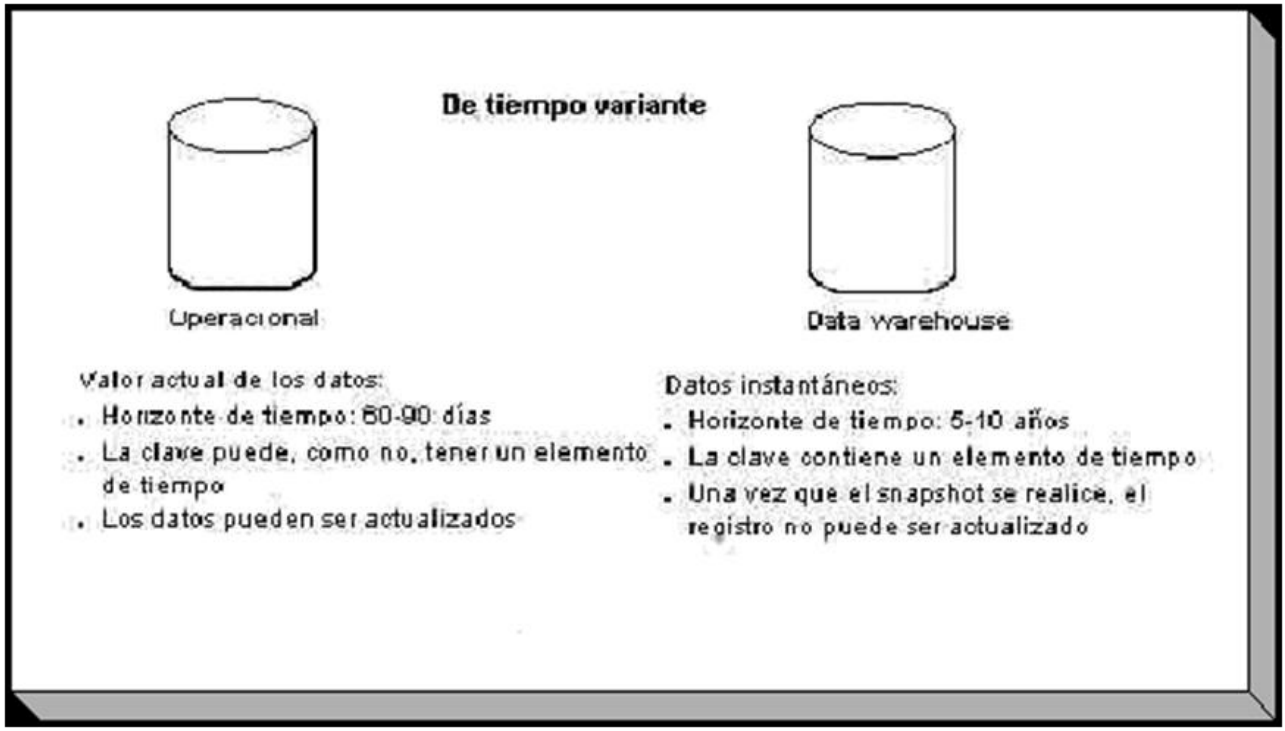

Figura 4 - Almacenes de Datos Variable en el Tiempo

No volátil: los datos que son almacenados no sufren ninguna actualización solo son incrementados. El período cubierto para un Almacén de Datos va de 2 a 10 años.

La información es útil sólo cuando es estable. Los datos operacionales cambian sobre una base momento a momento. La perspectiva más grande, esencial para el análisis y la toma de decisiones, requiere una base de datos estable.

Hay algunas consecuencias muy importantes de esta diferencia básica, entre el procesamiento operacional y del Almacén de Datos. En el nivel de diseño, la necesidad de

ser precavido para actualizar las anomalías no es un factor en el Almacén de Datos, ya que no se hace la actualización de datos (ver figura 5). Esto significa que en el nivel físico de diseño, se pueden tomar libertades para optimizar el acceso a los datos, particularmente al usar la normalización y des normalización física.

Otra consecuencia de la simplicidad de la operación del Almacén de Datos está en la tecnología subyacente, utilizada para correr los datos en el depósito. Teniendo que soportar la actualización de registro por registro en modo on-line (como es frecuente en el caso del procesamiento operacional) requiere que la tecnología tenga un fundamento muy complejo debajo de una fachada de simplicidad (Herrera, 2007).

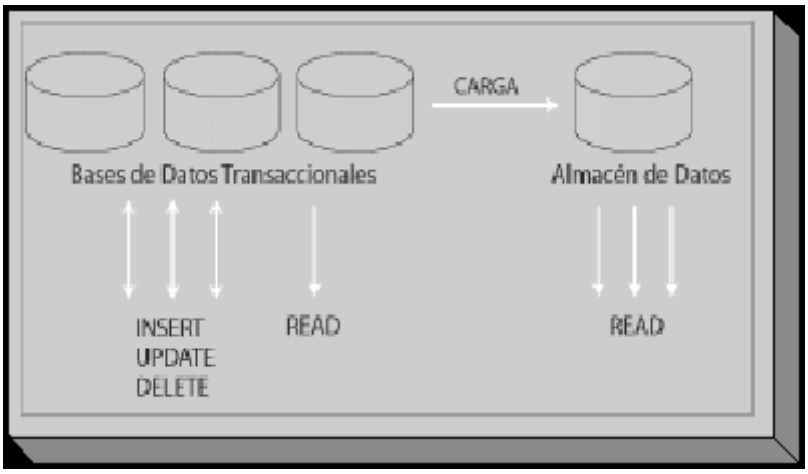

Figura 5 - Almacenes de Datos No Volátil

2.3.3 Proceso ETL (Extracción, Transformación y Carga)

Se define como el proceso a través del cual se gestionan datos obtenidos de múltiples fuentes, con el fin de recolectarlos, limpiarlos y cargarlos en **Bases de Datos Especializadas**, denominadas **Almacenes de Datos**, para analizar y apoyar una determinada **Línea de Producto** o **Unidad de Negocios** (Inteligencias de Negocios, 2009).

Extracción: el inicio de una Estrategia ETL consiste en adquirir o sustraer los datos *brutos* desde los Sistemas de Origen; es, en esencia, un proceso de almacenamiento masivo de datos, capaz de integrar, en una misma Metodología de Negocios, toda la información empresarial proveniente de diferentes fuentes; de hecho, cada sistema por separado organiza sus datos de diversas maneras.

Los formatos de las fuentes de datos *brutos*, por lo general, se ubican en Bases de Datos Relacionales o lo que se denomina como Ficheros Planos; no obstante, pueden incluir Bases de Datos No Relacionales u otras Estructuras de Datos diferentes.

La Extracción convierte los datos a un Formato Estándar, con el cual se inicia el Proceso de Transformación. Una parte intrínseca del proceso de extracción es la de analizar los datos extraídos, de lo cual resulta un chequeo que verifica si los datos cumplen con las pautas estipuladas y se adaptan al formato estándar diseñado. De no ser así, los datos son rechazados. La Tarea de Extracción debe causar un impacto mínimo en el sistema origen. Si los datos a extraer son muchos, el sistema de origen se podría ralentizar e incluso colapsar; provocando que éste no pueda implementarse con normalidad para su uso cotidiano. Por esta razón, en sistemas grandes las operaciones de extracción suelen programarse en horarios o días donde este impacto sea nulo o mínimo (Inteligencias de Negocios, 2009).

Transformación: en esta fase se aplican una serie de **Procedimientos de Negocios** sobre los datos extraídos, con el objeto de convertirlos en datos aptos para ser cargados. Algunas fuentes de datos, requerirán una pequeña manipulación de los mismos, no obstante, en otros casos, puede ser necesario aplicar algunas transformaciones específicas mínimo (Inteligencias de Negocios, 2009).

Carga: la fase de carga es el momento cuando los datos, provenientes de la fase anterior, son incluidos en el sistema de destino. Dependiendo de los requerimientos de la organización, este proceso puede abarcar una amplia variedad de acciones diferentes; en algunas bases de datos, se sobrescribe la información antigua con nuevos datos; el Almacén de Datos, por ejemplo, mantienen un historial de los registros de manera que se

pueda hacer una auditoría de los mismos y disponer de un rastro del comportamiento de un determinado valor a lo largo del tiempo.

Existen dos formas básicas de desarrollar el **Proceso de Carga**: la primera es la denominada **Acumulación simple**, la cual consiste en realizar un resumen de todas las transacciones comprendidas en el período de tiempo seleccionado y transportar el resultado como una única transacción hacia el Almacén de Datos; almacenando siempre un **Valor Calculado** que expresa el promedio de la magnitud considerada. La segunda forma de Proceso de Carga es el **Rolling** que se aplica en los casos en los cuales se almacena información resumida a distintos niveles, correspondientes a distintas agrupaciones de la unidad de tiempo o diferentes niveles jerárquicos, en alguna o varias de las dimensiones de la magnitud almacenada, por ejemplo, totales diarios, totales semanales, etc.

La fase de carga interactúa directamente con la base de datos de destino. Al realizar esta operación se aplicarán todas las restricciones que se hayan definido en ésta, por ejemplo, valores únicos, integridad referencial, campos obligatorios, rangos de valores. Estas restricciones están bien definidas, en tanto garantizan la calidad de los datos en el Proceso ETL (Inteligencias de Negocios, 2009).

El Proceso ETL se puede observar en la figura 6.

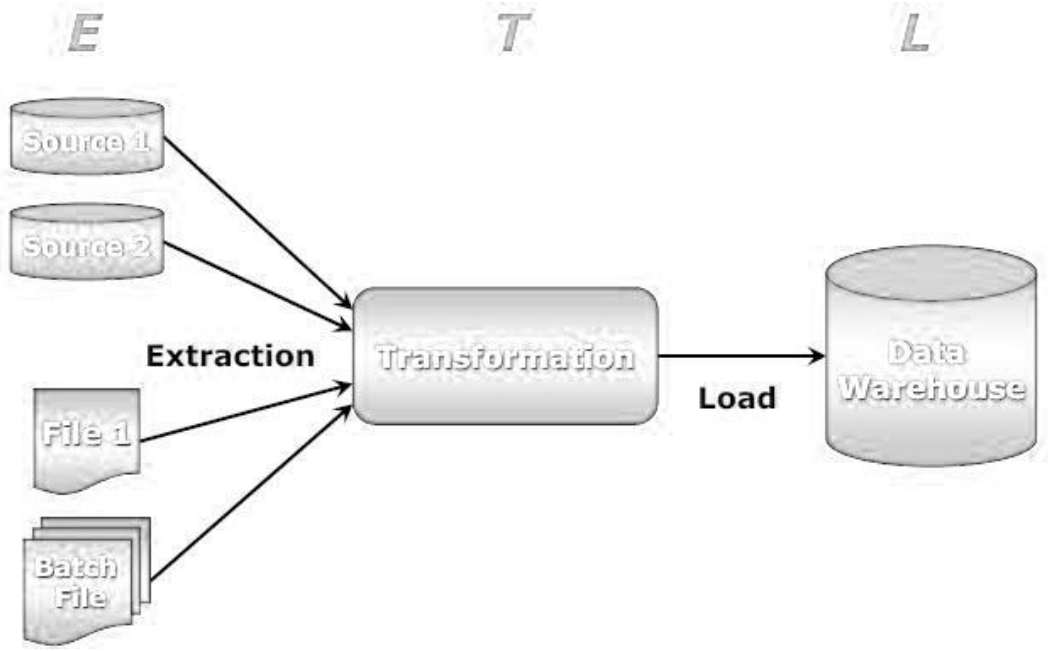

Figura 6 - Proceso ETL

2.3.4 Diseño de un Almacén de Datos

Con la vista puesta de entrar en detalle a hablar sobre el diseño de un Almacén de Datos, vamos a definir conceptos básicos: tabla de hecho, dimensión y métrica.

Si bien en las estructuras relacionales existentes en nuestras bases de datos transaccionales tenemos que crear un diseño lógico siguiendo las formas normales, en el Almacén de Datos no debe seguirse ese patrón de diseño. La idea principal es que la información sea presentada des normalizada para optimizar la consultas. Para ello debemos identificar en el seno de la organización, los procesos de negocio, las vistas para el proceso de negocio y medidas cuantificables asociadas a los mismos. De esta manera hablaremos de:

- **Tabla de hecho**: es la representación en el Almacén de Datos de los procesos de negocio de la organización. Por ejemplo, una venta puede identificarse como un proceso de negocio de manera que es factible, si corresponde en nuestra organización, considerar la tabla de hecho ventas.

- **Dimensión**: es la representación en el Almacén de Datos de una vista para un cierto proceso de negocio. Si regresamos al ejemplo de una venta, para la misma tenemos el cliente que ha comprado, la fecha en la que se ha realizado, etc. Estos conceptos pueden ser considerados como vistas para este proceso de negocio. Puede ser interesante recuperar todas las compras realizadas por un cliente. Ello nos hace entender por qué la identificamos como una dimensión.
- **Métrica**: son los indicadores de negocio de un proceso de negocio. Aquellos conceptos cuantificables que permiten medir nuestro proceso de negocio. Por ejemplo, en una venta tenemos el importe de la misma (Curto, 2007).

2.3.5 Tipos de Almacenes de Datos

En base a lo presentado en (Curto, 2007) se tiene que los tipos de Almacenes de Datos son:

Esquema en estrella: consiste en estructurar la información en procesos, vistas y métricas recordando a una estrella (ver figura 7). Es decir, tendremos una visión multidimensional de un proceso que medimos a través de unas métricas. A nivel de diseño, consiste en una tabla de hechos (lo que en los algunos libros se llamara como *fact table*) en el centro para el hecho objeto de análisis y una o varias tablas de dimensión (*dimension table*) por cada dimensión de análisis que participa de la descripción de ese hecho. En la tabla de hecho encontramos los atributos destinados a medir (cuantificar) el hecho: sus métricas. Mientras, en las tablas de dimensión, los atributos se destinan a elementos de nivel (que representan los distintos niveles de las jerarquías de dimensión) y a atributos de dimensión (encargados de la descripción de estos elementos de nivel). En el esquema en estrella la tabla de hechos es la única tabla del esquema que tiene múltiples joins que la conectan con otras tablas (*foreign keys* hacia otras tablas). El resto de tablas del esquema (tablas de dimensión) únicamente hacen join con esta tabla de hechos. Las tablas de dimensión se encuentran además totalmente de normalizadas, es decir, toda la información referente a una dimensión se almacena en la misma tabla.

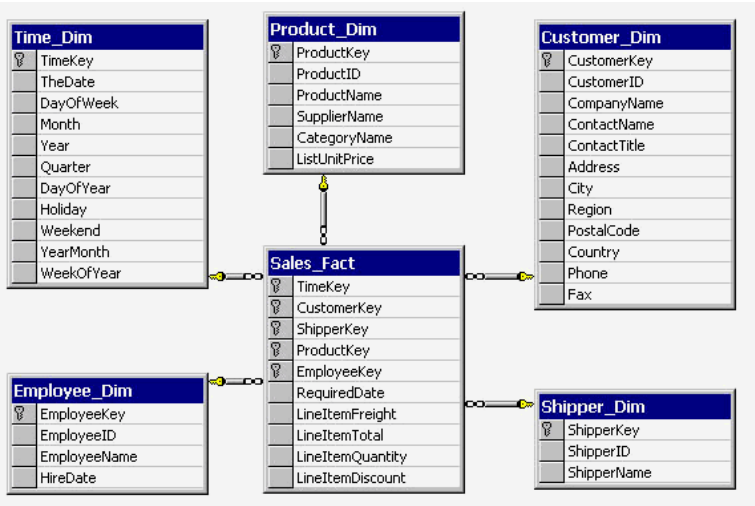

Figura 7 - Modelo Estrella (Almacenes de Datos)

Esquema en copo de nieve: El esquema en copo de nieve es un esquema de representación derivado del esquema en estrella, en el que las tablas de dimensión se normalizan en múltiples tablas (ver figura 8). Por esta razón, la tabla de hechos deja de ser la única tabla del esquema que se relaciona con otras tablas, y aparecen nuevas joins gracias a que las dimensiones de análisis se representan ahora en tablas de dimensión normalizadas. En la estructura dimensional normalizada, la tabla que representa el nivel base de la dimensión es la que hace join directamente con la tabla de hechos. La diferencia entre ambos esquemas (estrella y copo de nieve) reside entonces en la estructura de las tablas de dimensión. Para conseguir un esquema en copo de nieve se ha de tomar un esquema en estrella y conservar la tabla de hechos, centrándose únicamente en el modelado de las tablas de dimensión, que si bien en el esquema en estrella se encontraban totalmente de normalizadas, ahora se dividen en sub-tablas tras un proceso de normalización. Es posible distinguir dos tipos de esquemas en copo de nieve, un esquema copo de nieve completo (en el que todas las tablas de dimensión en el esquema en estrella aparecen ahora normalizadas en el esquema copo de nieve) o un esquema copo de nieve parcial (sólo se lleva a cabo la normalización de algunas de ellas). (Curto, 2007)

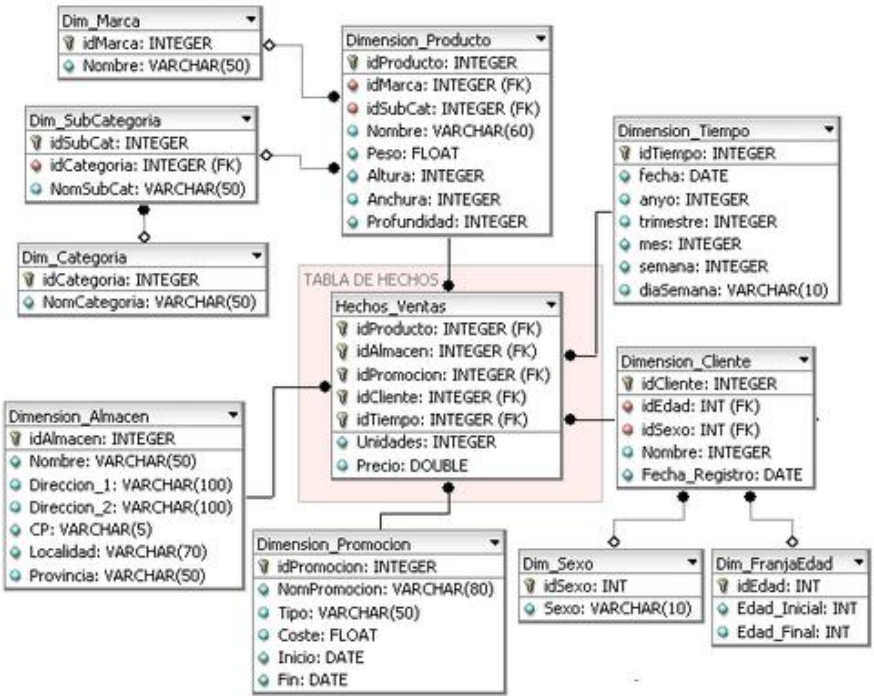

Figura 8 - Modelo Copo de Nieve (Almacenes de Datos)

2.3.6 Diferencia entre una Base de Datos Operacional y un Almacén de Datos

En un **Almacén de Datos** se almacena toda la información de interés para una organización que luego queramos analizar, mientras que, en una base de datos operacional se almacenan todas las transacciones de la organización, tanto datos útiles como no útiles. (Buigues, 2012)

En la figura 9 podemos ver qué se diferencian una base de datos operacional de un **Almacén de Datos**

Aspectos	BD operacional	Data Warehouse
Objetivo	De tipo operativo (operaciones del día a día)	Análisis y toma de decisiones
Proceso	De transacciones. Repetitivo y conocido.	De consultas masivas. Puntual y no conocido.
Actividad	Predomina la actualización	Predomina la consulta
Rendimiento	Importancia del tiempo de respuesta de la transacción instantánea	Importancia de la respuesta masiva.
Explotación	Explotación de la información relacionada con la operativa de cada aplicación	Explotación de toda la información interna y externa relacionada con el negocio
Volatilidad	Actualizable	Carga, pero no actualización
Usuarios	Usuarios de perfiles medios o bajos	Usuarios de perfiles altos
Organización	Estructura normalmente relacional	Visión multidimensional
Granularidad	Datos generales desagregados, al detalle	Datos en distintos niveles de detalle y agregación
Horizonte histórico	30 a 90 días	5 a 10 años
Perspectiva	Importancia del dato actual	Importancia del dato histórico
Volumen de datos	Pequeño/medio. Del orden del Mb a Gb.	Medio/grande. Del orden del Gb a Tb.

Figura 9 - Diferencias entre una Base de Datos Operacional y un Almacén de Datos

CAPITULO 3 - MARCO TECNOLÓGICO

3.1 MonetDB

Es un manejador de Base de Datos Orientados a Columnas que mejora la velocidad de las bases de datos XQuery y SQL bajo las reglas del Sistema Administrador de Base de Datos

Su objetivo es introducir innovaciones en todas las capas de un DBMS, por ejemplo, un modelo del almacenamiento basado en la fragmentación vertical, una arquitectura moderna de la ejecución de las consultas, unos índices automáticos y configurables, una optimización del tiempo de ejecución de las consultas y una arquitectura modular del programa (Softpedia, 2007).

3.1.1 Características

SQL:

- Es basado en el core estándar de SQL'99 incluyendo soporte para claves foráneas, consultas anidadas y vistas.
- Soporta Módulos de almacenamiento persistente, triggers y funciones definidas por el usuario escrito en el lenguaje C.
- Soporta tipos de secuencias del estándar SQL'03.
- Soporta aplicaciones geográficas con el estándar OPEN-SIG.
- Soporta la mayoría del estándar del SQL/XML.

Paquetes del software MONETDB:

- El código fuente del Kernel está escrito en ANSI-C y compatible con POSIX.
- La librería de códigos fuentes cumple con las últimas versiones del lenguaje.
- El código fuente está escrito en un estilo de programación literal, para estimular la proximidad del código y su documentación.

- El código fuente fue compilado y testeado con muchas opciones de compilación en distintas plataforma para asegurar la portabilidad.

Servidor MonetDB:

- Un total esquema de almacenamiento descompuesto usando mapeo de archivos en memoria.
- Soporta base de datos escalables en plataforma de 32bits y 64bits.
- La conectividad se puede realizar a través de TCP/IP, SSH y otras conocidas.
- Selección, creación y mantenimiento de los índices es de manera automática.
- Los operadores relacionales materializan sus resultados y se auto optimizan.
- El back-end de la base de datos es multi-hilos y guarda una instancia física de la base de datos.

Otras Características importante:

- **Es un sistema manejador de base de datos de alto rendimiento:** MonetDB es un código abierto DBMS de fácil acceso para aplicaciones en base a SQL y base de datos para proyectos de investigación.
- **Es un sistema de multi-modelos:** MonetDB soporta múltiples lenguajes de consultas a parte de su lenguaje propio, llamado MonetDB Assembly Language (MAL) que apunta a ANSI SQL-2003 y W3C Xquery con facilidades de actualización. El sistema está diseñado para proveer una base común entre los dos lenguajes y prepara un soporte para lenguajes en base a otro modelo de datos.
- **Es un Kernel de base de datos almacenados por columnas:** MonetDB y hecho en una representación canoníca de los elementos de la base de datos, llamado Binary Assciation Tables (BATs), MonetDB es una de las principales manejadores de base de datos que usan almacenamiento por columnas, un BAT es usado por cada columna de una tabla. La páginación en la memoria es sustituida por una que utiliza el sistema BAT que puede intercambiar cientos de megas en la memoria cuando es necesario.

- **Un sistema de Base de Datos:** MonetDB está en continuo desarrollo para soportar más aplicaciones de distintos campos, se está proyectando para el uso de grandes Base de Datos cuando antes se usaba para puros productos CRM.
- Es un software de código abierto (MonetDB).

3.2 MongoDB

MongoDB es una Base de Datos No Relacional Basada en Documentos, así no se usa las tablas y relaciones entre tablas, en MongoDB lo más parecido que encontraremos a una tabla son las colecciones, que vienen a ser una especie de listas donde vamos almacenando los diferentes objetos y sus atributos. Se abandona el enfoque relacional por bases de datos más orientadas a objetos y de esta manera es como se procesa la información (Ubuntu Life, 2010).

¿Base de Datos No Relacional Basada en Documentos?

A estas alturas es extraño toparse con alguien que no haya oído hablar de las **Base de Datos No Relacional Basada en Documentos**. No obstante, no todos los desarrolladores habitualmente tienen claro el por qué de su popularidad, posiblemente debido a que no han tenido aún la oportunidad de trabajar con alguno de los sistemas que se basan en este concepto.

En la carrera de la informática, muchos han aprendido que los **sistemas de bases de datos** se clasifican mayormente en tres tipos: Las bases de datos **relacionales**, las **orientadas a objetos**, y las **relacionales orientadas a objetos**. Sin embargo, pronto uno se da cuenta que, en la práctica, la mayoría de los motores de bases de datos más populares se basan en la arquitectura relacional, y todos ellos utilizan el lenguaje de consultas SQL (con variaciones) para operar con los datos. Tanto es así, que SQL se convirtió con el paso de los años en un estándar “de facto”, debido a su uso. (Paramio, 2011)

Su mayor ventaja es que están preparados para ser muy rápidos.

Según su tipo, cada una sigue una estrategia completamente diferente para persistir la información.

Cabe destacar que normalmente **no sustituyen a la base de datos clásica SQL**, sino que surgen por otra necesidad. Una necesidad de rendimiento extremo. Si se utilizan de una manera única, o se combinan con una base de datos SQL es una decisión de arquitectura del sistema (Pérez, 2011).

¿Por qué aparecen los sistemas de Base de Datos No Relacional Basada en Documentos?

Las bases de datos relacionales no tienen nada de malo: Precisamente gracias al transcurso de los años, se ha logrado aprender técnicas bastante comunes para **normalizarlas** en la medida de lo posible, **escalarlas** según crece la demanda, y utilizarlas como **sistema de persistencia** para almacenar información desde nuestro lenguaje procedural u orientado a objetos favorito (entre otros). La cuota de uso de software como SQLite, MySQL, PostgreSQL u Oracle, por poner cuatro ejemplos conocidos, es muy alta, encontrándose en la mayor parte de los desarrollos modernos.

Pero llegó la web, el software como servicio, los servicios en la nube y las startups de éxito con millones de usuarios. Y con todo ello llegaron los problemas de **alta escalabilidad**. Si bien los modelos relacionales se pueden adaptar para hacerlos escalar incluso en los entornos más difíciles, sí que es cierto que, a menudo, se hacen cada vez menos intuitivos a medida que aumenta la complejidad. Triples y cuádruples joins en consultas SQL que asustan al más pintado nada más verlas, a veces poco eficientes, y sistemas de almacenamiento de resultados en cachés para acelerar la resolución de las peticiones y evitar ejecutar cada vez estas pesadas operaciones, son el pan de cada día en muchos de estos proyectos de software.

Los sistemas de Base de Datos No Relacional Basada en Documentos intentan atacar este problema proponiendo una estructura de almacenamiento **más versátil**, aunque sea a costa de perder ciertas funcionalidades como las transacciones que engloban operaciones en más de una colección de datos, o la incapacidad de ejecutar el producto cartesiano de dos tablas (también llamado JOIN) teniendo que recurrir a la des normalización de datos.

¿En qué se diferencian exactamente?

Si se tuviera que resumir las características comunes en estos sistemas, se diría que son principalmente tres: **Ausencia de esquema** en los registros de datos, **escalabilidad** horizontal sencilla, y **velocidad** (aunque esto último no siempre es cierto, pues muchos de estos sistemas aún no están suficientemente maduros).

La primera característica significa que los datos no tienen una definición de atributos fija, es decir: Cada registro (o documento, como se les suele llamar en estos casos) puede contener una información con diferente forma cada vez, pudiendo así almacenar sólo los atributos que interesen en cada uno de ellos, facilitando el polimorfismo de datos bajo una misma colección de información. También se pueden almacenar estructuras de datos complejas en un sólo documento, como por ejemplo almacenar la información sobre una publicación de un blog (título, cuerpo de texto, autor, etc.) junto a los comentarios y etiquetas vertidos sobre el mismo, todo en un único registro. Hacerlo así aumenta la **claridad** (al tener todos los datos relacionados en un mismo bloque de información) y el **rendimiento** (no hay que hacer un JOIN para obtener los datos relacionados, pues éstos se encuentran directamente en el mismo documento).

Con escalabilidad horizontal se refiere a la posibilidad de aumentar el rendimiento del sistema simplemente añadiendo más nodos, sin necesidad en muchos casos de realizar ninguna otra operación más que indicar al sistema cuáles son los nodos disponibles. Muchos sistemas De Base de Datos No Relacional Basada en Documentos permiten utilizar consultas del tipo **Map-Reduce**, las cuales pueden ejecutarse en todos los nodos a la vez (cada uno operando sobre una porción de los datos) y reunir luego los resultados antes de devolverlos al cliente. La gran mayoría permiten también indicar otras cosas como el número de réplicas en que se hará una operación de escritura, para garantizar la disponibilidad. Y gracias al **sharding** y a no tener que replicar todos los datos en cada uno de los nodos, la información que se mueve entre las distintas instancias del motor de base de datos no tiene por qué ser demasiado intensiva. Por supuesto, se seguirá encontrándose con problemas de escalabilidad inherentes al tipo de software que

estemos construyendo, pero seguramente se podrá resolver más fácilmente con la ayuda de estas características.

Por último, muchos de estos sistemas realizan operaciones directamente en memoria, y sólo vuelcan los datos a disco cada cierto tiempo. Esto permite que las operaciones de escritura sean realmente rápidas. Por supuesto, trabajar de este modo puede sacrificar fácilmente la durabilidad de los datos, y en caso de cuelgue o apagón se podrían perder operaciones de escritura o perder la consistencia. Normalmente, esto lo resuelven permitiendo que una operación de escritura haya de realizarse en más de un nodo antes de darla por válida, o disminuyendo el tiempo entre volcado y volcado de datos a disco. Pero claro, aún así, existe ese riesgo. (Paramio, 2011)

3.2.1 MongoDB: Descripción y licencia

MongoDB es un sistema de base de datos **multiplataforma** orientado a **documentos**, de esquema libre. Como ya se explico, esto significa que cada entrada o registro puede tener un esquema de datos diferentes, con atributos o "columnas" que no tienen por qué repetirse de un registro a otro. Está escrito en C++, lo que le confiere cierta cercanía al *bare metal*, o recursos de hardware de la máquina, de modo que es bastante rápido a la hora de ejecutar sus tareas. Además, está licenciado como GNU AGPL 3.0, de modo que se trata de un software de licencia libre. Funciona en sistemas operativos Windows, Linux, OS X y Solaris.

Las características que más destacaría de MongoDB son su **velocidad** y su completo pero sencillo **sistema de consulta** de los contenidos de la base de datos. Se podría decir que alcanza un balance perfecto entre rendimiento y funcionalidad, incorporando muchos de los tipos de consulta que utilizaríamos en nuestro sistema relacional preferido, pero sin sacrificar en rendimiento (Paramio, 2011).

3.2.2 Terminología básica en MongoDB

En MongoDB, cada registro o conjunto de datos se denomina **documento**. Los documentos se pueden agrupar en **colecciones**, las cuales se podría decir que son el equivalente a las tablas en una base de datos relacional (sólo que las colecciones pueden

almacenar documentos con muy diferentes formatos, en lugar de estar sometidos a un esquema fijo). Se pueden crear **índices** para algunos atributos de los documentos, de modo que MongoDB mantendrá una estructura interna eficiente para el acceso a la información por los contenidos de estos atributos (Paramio, 2011).

3.2.3 Formato de los documentos en MongoDB

Los distintos documentos se almacenan en formato **BSON**, o Binary JSON, que es una versión modificada de JSON que permite búsquedas rápidas de datos. Para hacer una idea, BSON guarda de forma explícita las longitudes de los campos, los índices de los arrays, y demás información útil para el escaneo de datos. Es por esto que, en algunos casos, el mismo documento en BSON ocupa un poco más de espacio de lo que ocuparía de estar almacenado directamente en formato JSON. Pero una de las ideas claves en los sistemas De Base de Datos No Relacional Basada en Documentos es que el almacenamiento es barato, y es mejor aprovecharlo si así se introduce un considerable incremento en la velocidad de localización de información dentro de un documento.

Sin embargo, en la práctica, nunca se verá el formato en que verdaderamente se almacenan los datos, y trabajaremos siempre sobre un documento en **JSON** tanto al almacenar como al consultar información. Un ejemplo de un documento en MongoDB podría ser cómo se Muestra en la Figura 11:

Figura 10 - Ejemplo de Formato de un Documento (MongoDB)

Este documento pretende representar la manera en que podrían almacenarse los datos correspondientes a un post de un blog. Los atributos "_id" (o clave principal) pueden tener el formato que se desee, aunque MongoDB utiliza un valor parecido a un **UUID** en hexadecimal por defecto si no se ha especificado ninguno. A pesar de parecer un valor completamente aleatorio, utilizan como base una **semilla basada en la MAC de la interfaz de red** de la máquina (y otros detalles de la misma) para evitar que dos máquinas diferentes puedan generar el mismo valor para la clave de un documento. Y los primeros bytes corresponden a una marca de tiempo, de modo que las claves **se ordenan de forma natural por orden de creación** (o casi, pues está claro que las distintas máquinas corriendo MongoDB deben tener la fecha y hora sincronizadas) sin tener que mirar cuál fue el último valor usado. Una solución inteligente, es mucho más eficiente que un campo auto numérico, en especial para evitar que una máquina bloquee la inserción de registros en una colección sólo para asegurarse que no se dan condiciones de carrera al intentar

dos máquinas diferentes escribir un documento con el mismo valor para "_id". Por cierto, este atributo "_id" es el único obligatorio para un documento.

Las etiquetas y los comentarios están en el propio documento que representa al post, en lugar de guardarlos en colecciones separadas y utilizar claves foráneas para referenciar a los mismos. Sin embargo, en el atributo "liked_by" sí que guardamos una relación de claves, que corresponden a los usuarios que han marcado el post como que les ha gustado. Utilizar una forma u otra dependerá de las necesidades de acceso a estos datos En este caso, por ejemplo, se sabe que no se va a mostrar información sobre los usuarios que han marcado un post con un "me gusta", pero sí queremos ver cuántos lo han marcado así, o si el usuario actual ya lo ha marcado o no, con lo que almacenar únicamente las claves de esos usuarios y guardar su información personal detallada en otra colección es lo más conveniente.

Por supuesto, no es necesario pedir a MongoDB que nos devuelva todo el documento cada vez que lo consultamos. Si por ejemplo se va a mostrar únicamente un listado de posts recientes, seguramente sea suficiente obtener el atributo "title", con los documentos ordenados por "published_at". Así, **ahorramos ancho de banda** entre el motor de base de datos y la aplicación, al mismo tiempo que **memoria** dado que no hay que instanciar todo el documento. Además, si tenemos muchos miles de visitantes, el atributo "liked_by" podría llegar a crecer bastante.

A veces, toca **des normalizar** para poder tener a mano la información necesaria a la hora de mostrar un post. Es por eso que en el atributo "author_info" utiliza una versión intermedia: Si bien tenemos la clave principal del usuario que ha escrito este post, como es habitual que mostremos el nombre del autor, se ha almacenado también dicho nombre en el documento que representa al post, para que no sea necesario realizar una segunda consulta a la colección "usuarios". Estas des normalizaciones dependen nuevamente del uso que se den a los datos. En este caso, tengo claro que el nombre de un usuario **no va a cambiar demasiado**, así que recorrer todos los posts para cambiar este valor en caso de que el usuario realice esta operación, si bien es una modificación que podría llevar un

tiempo considerable para ejecutarse, no es una operación habitual frente a la consulta del nombre del autor, y por tanto compensa. Incluso se podría llegar a tratar el post como algo más permanente, de modo que aunque un usuario cambiara su nombre a posteriori, **el nombre utilizado para firmar los posts anteriores no varíe**, o sencillamente los posts puedan firmarse con diferentes pseudónimos.

El modelado del esquema de datos con MongoDB depende más de la forma en que se consultara o se actualizaran los datos que de las limitaciones del propio sistema (Paramio, 2011).

3.3 Ruby on Rails

Ruby on Rails (RoR o Rails) es un entorno de desarrollo web de código abierto escrito en Ruby y optimizado para satisfacción de los programadores y de la productividad. Te permite escribir aplicaciones web siguiendo el paradigma de la arquitectura Modelo Vista Controlador (MVC) (Alvarez, 2008).

3.3.1 Lenguaje de Programación Ruby

Es un lenguaje de scripts, multiplataforma, netamente orientado a objetos es software libre, fue creado por Yukihiro Matsumoto conocido como Matz. La primera versión fue liberada en 1995, hereda varias características de lenguajes como: Perl, Smalltalk, Eiffel, Ada y Lisp. Como lo indica su propio autor, es un lenguaje "aparentemente sencillo pero internamente complejo".

Esto quiere decir que mientras más nos abstraemos en el paradigma orientado a objetos notaremos realmente la complejidad del lenguaje, se considera un lenguaje muy intuitivo casi a un nivel de lenguaje humano.

Ruby fue diseñado para un desarrollo rápido y sencillo. Cada día este lenguaje va ganando más adeptos, tanto así que la empresa Sun Microsystems, está apoyando un proyecto llamado JRuby que es un intérprete de Ruby escrito 100% en Java.

Entre las características del lenguaje se encuentran:

- Posibilidad de hacer llamadas directamente al sistema operativo.
- Muy potente para el manejo de cadenas y expresiones regulares.
- No se necesita declarar las variables.
- La sintaxis es simple y consistente.
- Gestión de memoria automática.
- Todo es un objeto.
- Métodos Singleton (Santos, 2006).

3.3.2 FrameWorks Rails

Ruby on Rails es una plataforma de trabajo para realizar desarrollos web. Se puede definir como un framework de software libre. Ruby hace referencia al lenguaje de programación que se utiliza: todo se escribe en Ruby. Por otro lado, on Rails (sobre raíles) indica que el framework "nos va guiando" para hacer fáciles nuestros desarrollos.

Por framework entendemos un entorno o conjunto de programas o herramientas agrupadas. Por software libre entendemos software que cualquier usuario puede ejecutar, copiar, modificar, distribuir, cambiar o mejorar. Ruby es un lenguaje de alto nivel, es decir, de sintaxis más aproximada a la forma de expresarnos de los humanos que otros lenguajes. Por ejemplo el método "pluralize" nos permite obtener el plural de una palabra y el método "singularize" obtener el singular. Lo gracioso es que el lenguaje es casi humano: pluralize "cat" nos devuelve "cats" pero pluralize "person" nos devuelve "people", que es el plural en inglés de person. El número de usuarios de Ruby está yendo en aumento en los últimos años; sus usuarios lo defienden dando varios argumentos entre los que podríamos destacar que "funciona de forma eficiente" y por otro lado valoran mucho la sencillez de su sintaxis, que se aproxima a lo que podría ser el inglés escrito, un lenguaje "casi natural". Todo lo que se escribe en Ruby on Rails se escribe en Ruby, luego el framework se encarga de transformarlo en otros lenguajes en función de las necesidades: por ejemplo, en Java Script para mostrar un formulario en una página web o en SQL para realizar comunicaciones con una base de datos. El hecho de manejar un solo

lenguaje hace que Ruby on Rails resulte sencillo de utilizar. Como contrapartida, la transformación desde Ruby on Rails a otros lenguajes da lugar a que el código que subyace (el código que existe detrás de nuestro desarrollo web) no va a ser tan limpio y depurado como si hubiéramos utilizado directamente los lenguajes correspondientes.

Aunque Ruby es multiplataforma, es decir, se puede ejecutar bajo distintos sistemas operativos, es bajo Linux como se consigue un mejor rendimiento. Ruby on Rails debemos correrlo en Linux, en otras plataformas, por ejemplo Windows, su uso resulta problemático.

Como es propio en el software libre, hay una comunidad de programadores trabajando en Ruby on Rails. En España existe una comunidad bastante activa.

Ruby on Rails evoluciona al ir apareciendo sucesivamente nuevas versiones. Las migraciones (cambio de versión) con las últimas versiones pueden calificarse de sencillas y también reversibles, es decir, si después de hacer una migración se comprueba que existen problemas, se puede revertir el proceso con pocas complicaciones. Esto no significa que no puedan aparecer problemas, sino que con versiones antiguas estos procesos eran mucho más complicados (Rancel, 2011).

Rails está basado en dos principios de desarrollo:

- **No lo vuelvas a repetir:** El primer principio avisa de que con esta forma de trabajo podemos crear aplicaciones sin necesidad de duplicar código, por lo que la programación se hace mucho más rápida que con otros lenguajes. Además, al tener menos código, se hace más sencillo el localizar un error y es más claro e intuitivo para el programador. Esta primera idea significa poder tener, por ejemplo, un formulario definido que pueda ser llamado en cualquier parte del código y las veces que se desee mediante una sola línea.
- **Convención antes que Configuración:** El segundo principio señala que no existen archivos de configuración. En lugar de eso, podemos utilizar una serie de convenciones simples ya existentes. Por ejemplo, el tener una clase ‘Cliente’ en el

modelo que herede de la clase 'ActiveRecord::Base' significaría para Rails que existe una tabla en la base de datos llamada 'clientes', y que debe ser mapeada con esa clase 'Cliente'. Esto hace muy sencillo crear una aplicación desde cero siguiendo esta serie de convenciones, pero también puede ser adaptado a nuestros gustos si nos debemos apoyar en una base de datos ya existente, por ejemplo. Si en nuestra base de datos nuestra tabla 'clientes' tiene otro nombre, mediante una simple línea de código podremos indicárselo a Rails (Martinez, 2010).

Aspectos Técnicos:

Ruby on Rails se basa en el MVC (Modelo – Vista – Controlador) (ver figura 11). De forma simplificada, diremos que existe:

- Un elemento Action View (Modelo), encargado de controlar las vistas: aquello que se muestra al usuario o se envía. Action View se encarga de "mostrar" los datos adecuadamente: HTML para navegadores, XML para servicios web o aplicaciones o WML para teléfonos móviles.
- Otro elemento, Active Record (Vista), encargado de gestionar el modelo de datos.
- Otro elemento, Action Controller (Controlador), que se encarga de "dirigir" o comunicar modelo y vista.

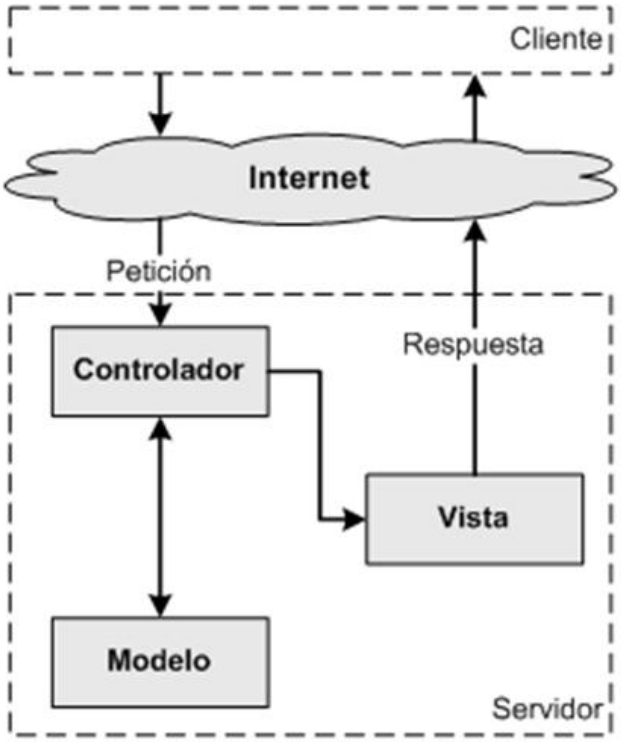

Figura 11 - Modelo Vista Controlador (Ruby On Rails)

Un proyecto de Ruby on Rails se organiza en carpetas. Orientativamente se va a tener al menos las siguientes carpetas (ver figura 12):

- **app:** carpeta donde va a ir prácticamente todo el código del proyecto.
- **config:** carpeta donde especificamos aspectos como la configuración de la base de datos entre otros aspectos de configuración.
- **public:** carpeta destinada a contener ficheros Java Script y hojas de estilo (CSS) entre otros.
- **scripts:** carpeta con varias herramientas para creación de contenidos.

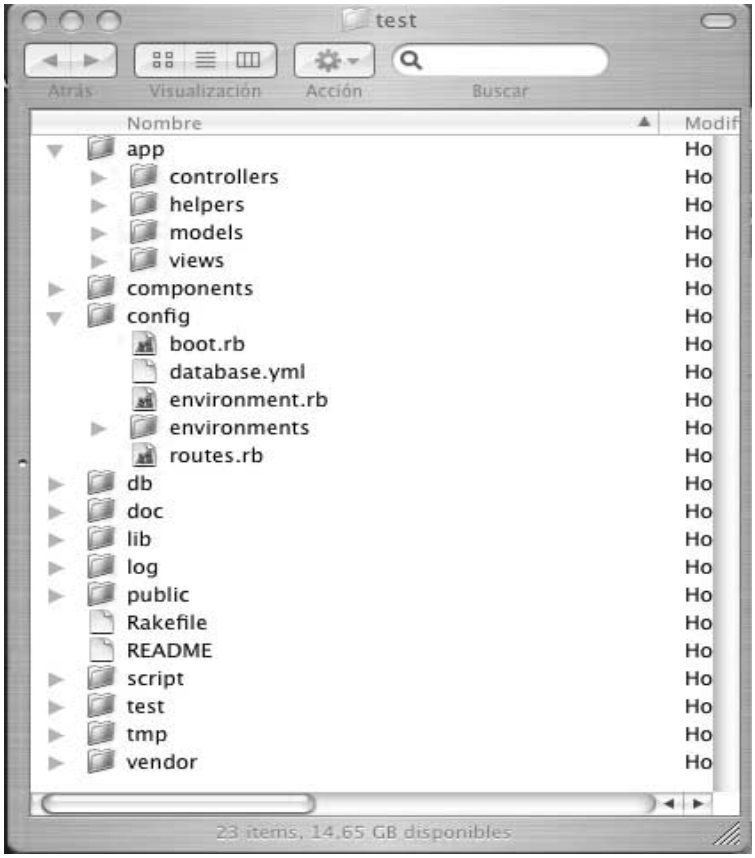

Figura 12 - Estructura de Carpetas (Ruby on Rails)

Ruby on Rails utiliza como elementos fundamentales plugins: pequeños fragmentos de código que permiten realizar tareas muy concretas. Por ejemplo hay un plugin que permite que un usuario se registre y que cuando se registre se le envíe un e-mail. Los desarrolladores de Ruby on Rails tienen su propia colección de plugins y sus desarrollos se basan principalmente en ellos. La comunidad de programadores publica los plugins en el sitio web "GitHub Social Coding" (github.com) donde quedan a disposición del resto de usuarios. ¿Para qué estar generando código que ya está generado? Por tanto, para aquellas personas que empiecen con Ruby on Rails, es muy recomendable que dediquen sus primeros días a explorar algunos de los plugins disponibles, su implementación y funcionamiento: encontrarán más plugins de los que van a tener tiempo a revisar o probar. Hay que tener en cuenta que los desarrolladores profesionales

basan sus proyectos en aproximadamente un 90 % de código en plugins y un 10 % de código propio. Por algo será.

Los plugins están desarrollados para versiones concretas de Ruby on Rails, por tanto no tenemos garantías de que vayan a funcionar cuando realizamos una migración. Esto es uno de los aspectos "problemáticos" de Ruby on Rails. Como siempre, procede ser cautelosos, y mantener copias de seguridad que eviten pérdidas de información si aparecen problemas.

Ruby on Rails puede generar aplicaciones que usen servidores Apache. Se suelen usar bases de datos MySql, pero también se pueden usar otras como Oracle. Podemos reseñar que trabajando bajo este framework podemos, en principio, cambiar de base de datos sin tener que hacer nada: por ejemplo, pasarnos de MySql a Oracle sin mayores problemas ya que el framework se encarga automáticamente de las adaptaciones necesarias.

Cuando creamos una aplicación con Ruby on Rails es muy sencillo hacer tests para comprobar que la aplicación responde tal y como esperamos antes de poner la aplicación en uso. El framework ejecuta un test y nos devuelve un informe. Es muy recomendable ir haciendo pruebas conforme se va realizando el desarrollo con Ruby on Rails, ya que no existe compilación como en otros lenguajes. Por tanto, testear aunque parezca una pérdida de tiempo en realidad va a evitarnos tener que hacer tediosas correcciones por fallos en el comienzo del desarrollo de un proyecto (Rancel, 2011).

3.3.3 RubyGems

RubyGems es un gestor de paquetes para el lenguaje de programación Ruby que proporciona un formato estándar y auto contenido (llamado **gemas**) para poder distribuir programas o librerías en Ruby, una herramienta destinada a gestionar la instalación de éstos, y un servidor para su distribución (Xqbot, 2009).

Gemas: Las Gemas son plugins y/o códigos añadidos a nuestros Proyectos de Ruby on Rails, que nos permiten nuevas funcionalidades como nuevos create, nuevas funciones pre-escritas (Como login de Usuarios) o nuevas herramientas para el desarrollo.

El gestor de gemas es un comando que nos permite listar las gemas instaladas, buscar gemas remotas o localmente (de las instaladas), instalar, actualizar y eliminar gemas. Se puede saber si está instalado simplemente ejecutando:

```
gem –version
```

Instalar una gema: Por ejemplo, si queremos usar el código siguiente:

```
require "Rubygems";
require "json";
puts [ "hola a todos", 12, "C/Sin nombre, 25" ].to_json
```

Tendremos que instalar previamente la gema *json*. Para instalarla, solo tenemos que ejecutar el siguiente comando:

```
$ sudo gem install json
[sudo] password for marubio:
Building native extensions.  This could take a while...
Successfully installed json-1.6.4
1 gem installed
Installing ri documentation for json-1.6.4...
Installing RDoc documentation for json-1.6.4...
```

Como se puede apreciar, se instala la gema y se genera la documentación para *ri* y *rdoc*. La generación de la documentación es bastante lenta y si tenemos que instalar muchas gemas, puede ser algo tedioso. Se puede eliminar esta tarea agregando al comando de instalación: `--no-ri --no-rdoc`

Ahora, cuando se listen los paquetes instalados podremos ver:

```
$ sudo gem list

*** LOCAL GEMS ***

json (1.6.4)
```

Nos indica la versión, ya que podríamos tener varias versiones instaladas y conviviendo, y desde el código emplear la que requiramos específicamente. Si se quiere instalar una versión específica de un paquete, podemos hacerlo agregando la opción, por ejemplo: `--v=1.6.1 (Bombadil, 2012)`

CAPITULO 4 - MARCO METODOLÓGICO

En este capítulo se describe el método que se utiliza para el desarrollo del sistema propuesto. El método a utilizar es la programación extrema es perfecta para proyectos pequeños y medianos, pero es aún mejor para grandes proyectos o de alto riesgo.

En la programación extrema se da por supuesto que es imposible prever todo antes de empezar a codificar. Es imposible capturar todos los requisitos del sistema, saber qué es todo lo que tiene que hacer ni hacer un diseño correcto al principio. Es bastante normal hacer un diseño, ponerse a codificar, ver que hay faltantes o errores en el diseño, empezar a codificar fuera del diseño y al final el código y el diseño, o no se parecen, o hemos echado un montón de tiempo en cambiar la documentación de diseño para que se parezca al código.

En vez de tratar de luchar contra todo eso, lo asume y busca una forma de trabajar que se adapte fácilmente a esas circunstancias. Básicamente la idea de la programación extrema consiste en trabajar estrechamente con el cliente, haciéndole mini-versiones con mucha frecuencia (cada dos semanas). En cada mini-versión se debe hacer el mínimo de código y lo más simple posible para que funcione correctamente.

Ah continuación se explicara un poco sobre el Método XP.

4.1 Metodología XP (Programación Extrema)

A principios de 1990, un hombre llamado Kent Beck estaba pensando en diferentes maneras para desarrollar software. Hasta que en 1996, puso en práctica Kent su nueva metodología en el proyecto de DaimChrysler utilizó distintas técnicas de desarrollo de software, esta nueva metodología para desarrollar software, fue nombrada Extreme programming.

Extreme programming es una nueva disciplina del desarrollo de software que se basa en la **comunicación, simplicidad, retroalimentación** y **valor**.

La **simplicidad** ayuda a que los desarrolladores de software encuentren soluciones más simples a problemas, según el cliente lo estipula. Los desarrolladores también crean características en el diseño que pudieran ayudar a resolver problemas en un futuro.

La **comunicación** prevalece en todas las prácticas de extreme programming. Comunicación cara a cara es la mejor forma de comunicación, entre los desarrolladores y el cliente. Método muy ágil. Gracias a esto el equipo esta pude realizar cambios que al cliente no le gustaron. También apoya agilidad con la extensión del conocimiento tácito dentro del equipo del desarrollo, evitando la necesidad de mantener la documentación escrita.

La **Retroalimentación** continua del cliente permite a los desarrolladores llevar y dirigir el proyecto en una dirección correcta hacia donde el cliente quiera.

El **valor** requiere que los desarrolladores vayan a la par con el cambio, porque sabemos que este cambio es inevitable, pero el estar preparado con una metodología ayuda a ese cambio.

La metodología se diseña para entregar el software según las necesidades de cliente y cuando sea necesaria.

Extreme programming establece cuatro variables para cualquier proyecto de software: **costo, tiempo, calidad** y **alcance**.

Un problema de esto es la calidad, porque muchas veces se ignora, porque nadie es capaz de trabajar bien cuando está sometido a mucha presión.

El **costo** del proyecto se incrementa cuando se necesita máquinas más rápidas, mas especialistas técnicos en determinadas áreas o mejores oficinas para el equipo de desarrollo.

La **calidad** puede representar un cambio extraño; debido a que a mayor calidad menor tiempo de realización del proyecto. Por lo tanto el equipo de desarrolladores está encargado de la tarea de hacer las pruebas con los mejores resultados posibles para así tener una idea de cuál es el problema y como lo van a resolver de una manera simple y eficiente, para que la calidad del proyecto se mantenga al 100% y tener una facilidad de adaptarse a los cambios del código lo que hace este proceso más rápido.

El alcance del proyecto, en la cual el equipo determina: la estimación de las tareas a realizar, que es lo que el cliente quiere, la implementación de los requisitos más importantes de manera que este siempre sea funcional.

En extreme programming el costo del cambio maneja un papel muy importante, porque comparado con otras metodologías para implementar software, es mucho más barato, debido a que las pruebas se van haciendo según las versiones liberadas. (Paredes Villarreal, 2007)

Diagrama de los ciclos de desarrollo de software se muestra en la figura 13.

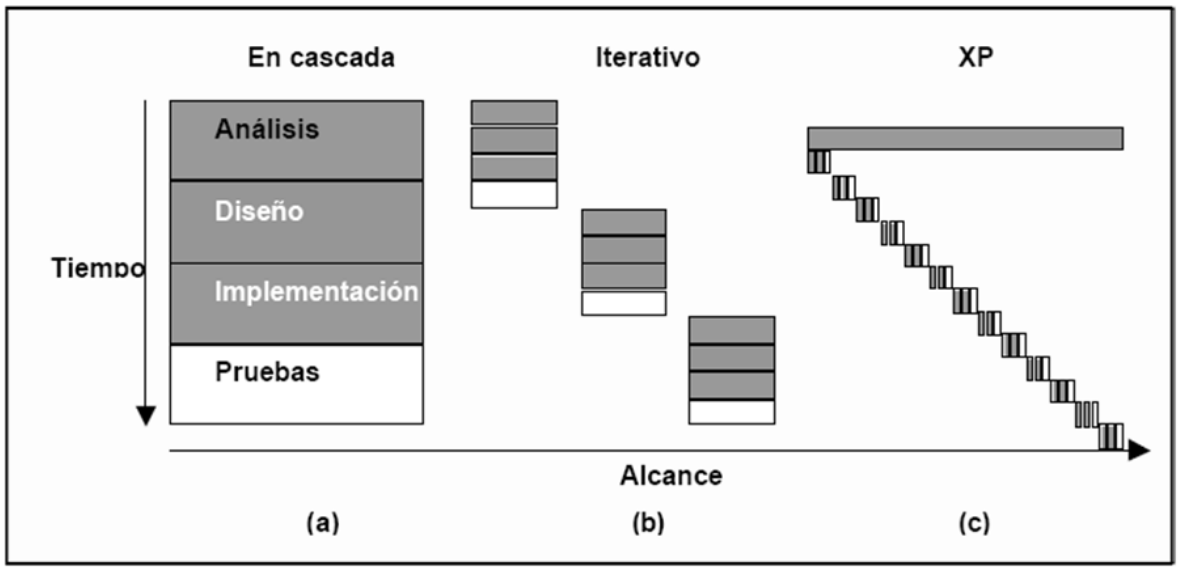

Figura 13 - Diagrama de los ciclos de desarrollo de software

4.1.1 Valores del Extreme Programing

- **El Juego de Planificación:** Son una serie de actividades planificadas que son llamados juegos planificados por extreme programming, son asumidos durante el transcurso del proyecto. Un juego inicial planeado se emprende durante el inicio del proyecto e implica una sesión con el cliente de reflexión con algunos personas claves como desarrolladores para determinar los requisitos iniciales y el contorno, conocidos en la metodología como historias. Si el proyecto continúa, se sigue con un emprende "Planning game de lanzamiento" donde el cliente escribe otras historias en tarjetas y los desarrolladores buscan términos de Unidades Ideales de Ingeniería.
- **Desarrollo de Probar-Conducir:** Esta es la forma de prueba de unidad y de prueba de aceptación, donde las pruebas se escriben antes de la implementación del código.
- **El reorganizado sin piedad:** Los desarrolladores requieren mejorar el código sin que este cambie su función para realizar un código simple. Esto dará oportunidad a modificarlo cuando haya necesidad de cambiar una característica.
- **Integración continua:** Al cabo de un día el sistema deberá de ser integrado por una máquina, cada vez que los programadores tengan una funcionalidad ya probada unitariamente. Si al añadir la funcionalidad al sistema este sigue funcionando correctamente, la tarea fue realizada con éxito. De lo contrario, esto permite que los desarrolladores detecten errores en la integración del sistema en una etapa antes ejecutando las pruebas de unidad del proyecto.
- **Cliente en el sitio:** Los desarrolladores tienen acceso continuo con el cliente para poner en claro las historias y discutir el desarrollo de ediciones y proporcionar retroalimentación.
- **Lanzamientos pequeños**: Al final de cada iteración el sistema es ejecutado y el cliente lo observa. Se ejecuta primero unos meses antes de estar completamente

terminado, las otras versiones serán más frecuentes entre un día y un mes. La mayoría de estas ejecuciones es para conseguir retroalimentación del cliente.

- **Aplique los estándares de la codificación:** El código es la forma principal de documentación y por lo tanto debe de estar escrito de una manera clara y constante, que pueda ser identificado fácilmente por cualquier programador de otro equipo.
- **Metáfora del sistema:** Un lenguaje de metáforas se utiliza para describir la arquitectura del sistema. Esto ayuda a la comunicación entre los mismos desarrolladores y entre los desarrolladores y el cliente.
- **Reunión:** Una reunión de 15 minutos en el comienzo de cada día para discutir problemas encontrados el día anterior y los problemas que se resolverán durante el día. (Paredes Villarreal, 2007)

4.1.2 Ventajas del Método XP

Evidentemente, para que algo esté siendo tomado tan en cuenta como la Extreme Programming, debe ofrecer una serie de ventajas a la hora de ponerlo en práctica que haga que el esfuerzo de entender y aplicar sus prácticas, sea insignificante con respecto a los beneficios obtenidos.

- Se consiguen productos usables con mayor rapidez.
- El proceso de integración es continuo, por lo que el esfuerzo final para la integración es nulo.
- Se consigue integrar todo el trabajo con mucha mayor facilidad.
- Se atienden las necesidades del cliente con mayor exactitud. Esto se consigue gracias a las continuas versiones que se ofrecen al usuario.
- Se consiguen productos más fiables y robustos contra los fallos gracias al diseño de los test de forma previa a la codificación.
- Obtenemos código más simple y más fácil de entender, reduciendo el número de errores. (Paredes Villarreal, 2007)

4.1.3 Adaptación del Proceso XP

Aquí se explicara algunos elementos de la metodología XP que se van a utilizar en el desarrollo del Sistema.

- **Iteraciones**

Todo proyecto que siga la metodología XP. Se ha de dividir en iteraciones para lograr la evolución del proyecto de manera escalable, estas iteraciones normalmente son propuestas por tiempo de duración, para el desarrollo de esta Tesis de Grado se decidió que cada iteración represente un objetivo.

- **Historias de usuario**

El primer paso a seguir con la metodología XP es elaborar Historias de Usuario con el cliente, son formatos muy son diligenciadas por el cliente, con un lenguaje no técnico. Son usadas para estimar tiempos de desarrollo de la aplicación que describen, también se verifica en la fase de Pruebas para afirmar que el programa cumple con la Historia de Usuario específica.

Cuando se implementa una historia de Usuario, el cliente se reúne con los desarrolladores para concretar y detallar lo que contiene la Historia a realizar. El formato que se va a utilizar para una Historia de Usuario es el siguiente:

Numero:	**Nombre:**
Usuario:	**Iteración Asignada:**
Tipo:	**Tiempo Estimado:**
Descripción:	

Cuadro 1 - Formato de Historia de Usuarios

- **Roles de la Programación Extrema**

Los actores son todas las personas involucradas en el desarrollo del proyecto, los cuales a su vez cumplen distintos roles o responsabilidades según su importancia y nivel de participación. A continuación se destacan los roles existentes en el presente proceso de desarrollo:

- **Programador:** El programador escribe las pruebas unitarias y produce el código del sistema.
- **Cliente:** Escribe las historias de los usuarios y las pruebas funcionales para validar su implementación. El cliente da una gran prioridad a las historias de usuarios y decide cual implementar en cada iteración centrándose en aportar mayor valor al negocio.
- **Encargado de Pruebas:** Ayuda al cliente a escribir las pruebas funcionales. Se encarga de ejecutar las pruebas con regularidad, difunde los resultados obtenidos al equipo y es el responsable de las herramientas que dan soporte a las pruebas.
- **Encargado de Seguimiento:** Es el que proporciona la realimentación al equipo. Realiza el seguimiento del proceso de cada iteración y verifica el grado de acierto entre las estimaciones realizadas y el tiempo real dedicado en ello para la mejora de futuras estimaciones.

Nombre del Actor / Roles	Programador	Cliente	Encargado de Pruebas	Encargado de Seguimiento
Carlos Blanco	X		X	
Mercy Ospina		X		X

Cuadro 2 - Nombre de Actores y sus Roles

- **Actividades XP**

La Metodología XP está compuesta principalmente de cuatros actividades, estas actividades van a estar incluidas en cada una de las iteraciones necesarias para el desarrollo de la Tesis de Grado. A continuación se dará una explicación de cada una de ellas.

- **Planificación:** En cualquier proyecto que usa la metodología XP se parte de las historia de usuarios para conocer detalles como funcionalidades y actividades, se utilizan también para estimar el tiempo de desarrollo requerido para las distintas iteraciones (Jeffries, 2001).

  También se utilizan en la fase de pruebas, para verificar si el programa cumple con lo que especifica la historia de usuario. Cuando llega la hora de implementar una historia de usuario, el cliente y los desarrolladores se reúnen para concretar y detallar lo que tiene que hacer dicha historia.

  Para las iteraciones se usara el siguiente formato:

Numero de Iteración:	**Numero de Historia:**
Fecha de Inicio:	**Fecha de Fin:**
Descripción:	**Tipo:**

Cuadro 3 - Formato del Cuadro de Interacciones

- **Diseño:** La metodología XP sugiere que hay que conseguir diseños simples y sencillos. Hay que procurar hacerlo todo lo menos complicado posible para conseguir un diseño fácilmente entendible e impleméntable que a la larga costará menos tiempo y esfuerzo desarrollar.

  También en la fase de diseño se puede Refactorizar que es mejorar y modificar la estructura y codificación de códigos ya creados sin alterar su funcionalidad. Refactorizar supone revisar de nuevo estos códigos para

procurar optimizar su funcionamiento. Es muy común rehusar códigos ya creados que contienen funcionalidades que no serán usadas y diseños obsoletos. Esto es un error porque puede generar código completamente inestable y muy mal diseñado; por este motivo, es necesario refactorizar cuando se va a utilizar código ya creado (Jeffries, 2001).

- **Codificación:** en la codificación el cliente es una parte más del equipo de desarrollo; su presencia es indispensable en las distintas fases de XP. A la hora de codificar una historia de usuario su presencia es aún más necesaria. No olvidemos que los clientes son los que crean las historias de usuario y negocian los tiempos en los que serán implementadas. Antes del desarrollo de cada historia de usuario el cliente debe especificar detalladamente lo que ésta hará y también tendrá que estar presente cuando se realicen los test que verifiquen que la historia implementada cumple la funcionalidad especificada.

  La codificación debe hacerse ateniendo a estándares de codificación ya creados. Programar bajo estándares mantiene el código consistente y facilita su comprensión y escalabilidad.

  XP sugiere un modelo de trabajo usando repositorios de código dónde las parejas de programadores publican cada pocas horas sus códigos implementados y corregidos junto a los test que deben pasar. De esta forma el resto de programadores que necesiten códigos ajenos trabajarán siempre con las últimas versiones. Para mantener un código consistente, publicar un código en un repositorio es una acción exclusiva para cada pareja de programadores (Jeffries, 2001).

- **Pruebas:** Uno de los pilares de la metodología XP es el uso de test para comprobar el funcionamiento de los códigos que vayamos implementando. Un punto importante es crear test que no tengan ninguna dependencia del código que en un futuro evaluará. Hay que crear los test abstrayéndose del futuro

código, de esta forma aseguraremos la independencia del test respecto al código que evalúa.

El uso de los test es adecuado para observar la refactorización. Los test permiten verificar que un cambio en la estructura de un código no tiene porqué cambiar su funcionamiento.

Al ser las distintas funcionalidades de la aplicación no demasiado extensas, no se harán test que analicen partes de las mismas, sino que las pruebas se realizarán para las funcionalidades generales que debe cumplir el programa especificado en la descripción de requisitos (Jeffries, 2001).

Para esta Tesis de grado se usara pruebas del programador que consiste en una técnica simple que consistirá en evaluar parámetros de entrada seleccionado por los programadores y observar las salidas constatando que cumplan con lo esperado. Para estas pruebas se usara el siguiente formato:

Numero de Caso de Prueba:	**Numero de Historia de Usuario:**
Descripción:	
Resultados Esperados:	**Resultados Obtenidos:**

Cuadro 4 - Formato de Casos de Pruebas

4.2 Metodología de Kimball para el Diseño de un Almacen de Datos

Ah continuación se explicara los 4 pasos para el diseño de un Almacén de Datos según Kimball (Rivadera, 2010)

- **Elegir el proceso de negocio:** el primer paso es elegir el área a modelizar. Esta es una decisión de la dirección, y depende fundamentalmente del análisis de requerimientos y de los temas analíticos anotados en la etapa anterior.

- **Establecer el nivel de granularidad:** la granularidad significa especificar el nivel de detalle. La elección de la granularidad depende de los requerimientos del negocio y lo que es posible a partir de los datos actuales. La sugerencia general es comenzar a diseñar el Almacen de Datos al mayor nivel de detalle posible, ya que se podría luego realizar agrupamientos al nivel deseado. En caso contrario no sería posible abrir (drill-down) las sumarizaciones en caso de que el nivel de detalle no lo permita.
- **Elegir las dimensiones:** las dimensiones surgen naturalmente de las discusiones al momento de diseñar, y facilitadas por la elección del nivel de granularidad y de la matriz de procesos/dimensiones. Las tablas de dimensiones tienen un conjunto de atributos (generalmente textuales) que brindan una perspectiva o forma de análisis sobre una medida en una tabla hechos.
- **Identificar las tablas de hechos y medidas:** el último paso consiste en identificar las medidas que surgen de los procesos de negocios. Una medida es un atributo (campo) de una tabla que se desea analizar, sumarizando o agrupando sus datos, usando los criterios de corte conocidos como dimensiones. Las medidas habitualmente se vinculan con el nivel de granularidad y se encuentran en tablas que denominamos tablas de hechos. Cada tabla de hechos tiene como atributos una o más medidas de un proceso organizacional, de acuerdo a los requerimientos. Un registro contiene una medida expresada en números, como ser cantidad, tiempo, dinero, etc., sobre la cual se desea realizar una operación de agregación (promedio, conteo, suma, etc.) en función de una o más dimensiones. La granularidad es el nivel de detalle que posee cada registro de una tabla de hechos.
- **Identificación de atributos de dimensiones y tablas de hechos:** la segunda parte de la sesión inicial de diseño consiste en completar cada tabla con una lista de atributos bien formada. Esta lista se forma colocando en las filas los atributos de la tabla, y en las columnas la siguiente información:

- Características relacionadas con la futura tabla dimensional del almacén de datos, por ejemplo tipo de datos, si es clave primaria, valores de ejemplo, etc.
- El origen de los datos (por lo general atributos de las tablas transaccionales).
- Reglas de conversión, transformación y carga, que nos dicen como transformar los datos de las tablas de origen a las del almacén de datos.

CAPITULO 5 - MARCO APLICATIVO

5.1 Plan de Iteraciones

El ciclo de vida de XP se enfatiza en el carácter iterativo e incremental del desarrollo, una iteración de desarrollo es un período de tiempo en el que se realiza un conjunto de funcionalidades determinadas que en el caso de XP corresponden a un conjunto de historias de usuarios.

Las iteraciones son relativamente cortas ya que se piensa que entre más rápido se le entreguen desarrollos al cliente, más retroalimentación se va a obtener y esto va a representar una mejor calidad del producto a largo plazo. Existe una fase de análisis inicial orientada a programar las iteraciones de desarrollo y cada iteración incluye planificación, diseño, codificación y pruebas, fases superpuestas de tal manera que no se separen en el tiempo (Patricio, 2006).

A continuación se presenta las diferentes iteraciones relacionada con el proyecto:

5.1.1 ITERACIÓN 0

- **Planificación:** Como primera iteración se va a instalar el ambiente de trabajo eso incluye los lenguajes necesarios y todas sus dependencias necesarias para la ejecución correcta del proyecto. También en esta iteración se realiza el diseño del Almacén de Datos en la cual se va a implementar en la siguiente iteración.

Numero de Iteración: 0	**Numero de Historia:** 1 y 2
Fecha de Inicio: 01/01/2011	**Fecha de Fin:** 20/01/2011
Descripción: Instalación del ambiente de trabajo y diseño del Almacén de Datos	**Tipo:** Desarrollo

- **Historias de Usuario:**

Numero: 1	**Nombre:** Diseño del Almacén de Datos
Usuario: Carlos Blanco	**Iteración Asignada:** 0
Tipo: Nueva	**Tiempo Estimado:** 9 días
Descripción: Se realiza el diseño del Almacén de Datos, estableciendo la tabla de hechos, las distintas dimensiones y las variables contenida en cada tabla.	

Numero: 2	**Nombre:** Instalación del ambiente de trabajo
Usuario: Carlos Blanco	**Iteración Asignada:** 0
Tipo: Nueva	**Tiempo Estimado:** 10 días
Descripción: Se procede a instalar todas las herramientas necesaria para el correcto funcionamiento de todo lo necesario para el desarrollo. Estas herramientas son: MonetDB, MongoDB, Ruby on Rails, RubyGems y Java	

- **Descripción:** En esta primera iteración se va a instalar primero los lenguajes necesarios para el desarrollo del proyecto estos lenguajes son los siguientes:
  - Ruby 1.8.7 (2011-06-30 patchlevel 352) [i686-linux]
  - Java(TM) SE Runtime Environment (build 1.6.0_26-b03)

  Al tener instalados estos lenguajes se instala el manejador de Almacenes de Datos MonetDB en su versión (11.7.9-20120312) y el manejador de base de datos no relacional basada en documentos MongoDB en su versión (2.0.1), posteriormente se instala la versión de Rubygems (1.5.3).

Finalmente se instala las distintas gemas que no están incluidas por defecto en Rubygems pero que son necesarias para el desarrollo del proyecto, las gemas instaladas se mostraran en el cuadro 5:

actionmailer (2.3.5)	nokogiri (1.5.0)
actionpack (2.3.5)	open4 (1.2.0)
activerecord (2.3.5)	popen4 (0.1.2)
activeresource (2.3.5)	rails (2.3.5)
activesupport (2.3.5)	rake (0.9.2.2)
bson (1.4.0)	sqlite3 (1.3.4)
bson_ext (1.4.0)	sqlite3-Ruby (1.3.3)
mongo (1.4.0)	

Cuadro 5 - Gemas Instaladas

Se puede observar que la versión de Rails utilizada es la 2.3.5, todas las versiones seleccionadas se usan porque representa su versión más estable.

- **Pruebas:** En la fase de prueba de la iteración 0 se realizo las siguientes pruebas:

Numero de Caso de Prueba: 1	**Numero de Historia de Usuario:** 2
Descripción: Imprimir por pantalla cualquier dato para probar el funcionamiento de los lenguajes Ruby y Java.	
Resultados Esperados: Mostrar algún dato en pantalla usando los imprimir en pantalla de cada lenguaje.	**Resultados Obtenidos:** Se imprimió en pantalla el clásico Hola Mundo en ambos lenguajes

Numero de Caso de Prueba: 2	**Numero de Historia de Usuario:** 2
Descripción: Usar la base de datos de prueba de MonetDB y realizar cualquier consulta.	
Resultados Esperados: Mostrar algún dato en pantalla ejecutando un query	**Resultados Obtenidos:** Se imprimió en pantalla todos los datos de una tabla usando el query de Select * from

Numero de Caso de Prueba: 4	**Numero de Historia de Usuario:** 2
Descripción: Hacer consultas de la base de datos iticve_db desde Ruby usando la librería mongo (1.4.0).	
Resultados Esperados: Mostrar algún dato en pantalla ejecutando un query en de MongoDB usando Ruby	**Resultados Obtenidos:** Se imprimió en pantalla los datos de la base de datos iticve_db usando el comando ítems.find(), ítems es la colección donde están los datos.

Numero de Caso de Prueba: 5	**Numero de Historia de Usuario:** 2
Descripción: Crear cualquier ambiente de trabajo web para verificar el correcto funcionamiento de Rails.	
Resultados Esperados: Mostrar una página usando el Frameworks Ruby on Rails.	**Resultados Obtenidos:** Se mostro la pantalla de un proyecto creado en Ruby on Rails usando la dirección web “locahost”
Numero de Caso de Prueba: 6	**Numero de Historia de Usuario:** 2
Descripción: Crear un query sencillo en java para realizar una consulta a la base de datos de ejemplo en MonetDB.	
Resultados Esperados: Mostrar los resultado de un query en MonetDB usando JAVA	**Resultados Obtenidos:** Se creó un archivo con query sencillo en java y al ejecutarlo se realizo con éxito el query mostrando los resultados por pantalla.

Numero de Caso de Prueba: 7	**Numero de Historia de Usuario:** 2
Descripción: Usar la librería popen4 en Ruby para ejecutar el query escrito en java desde Ruby y los resultados imprimirlos por pantalla.	
Resultados Esperados: Mostrar los resultado de un query en MonetDB usando la librería Popen4 a través de Ruby.	**Resultados Obtenidos:** Se creó un archivo con query sencillo en java, ese archivo se ejecuto en Ruby usando Popen4 y el resultado del query se mostraron en Ruby.

5.1.2 ITERACIÓN 1

- **Planificación:** En esta iteración se crea el Almacén de Datos en el manejador de Almacenes de Datos MonetDB, partiendo del diseño creado en la iteración anterior tenemos los datos necesarios para crear las distintas tablas de dimensiones y la tabla de hechos, todas las tablas con sus distintas variables respectivas.

Numero de Iteración: 1	**Numero de Historia:** 3
Fecha de Inicio: 21/01/2011	**Fecha de Fin:** 28/01/2011
Descripción: Implementación del Almacén de Datos	**Tipo:** Desarrollo

- **Historia de Usuario:**

Numero: 3	**Nombre:** Implementación del Almacén de Datos
Usuario: Carlos Blanco	**Iteración Asignada:** 1
Tipo: Nueva	**Tiempo Estimado:** 7 días
Descripción: Partiendo del Diseño del Almacén de Datos se procede a implementar el diseño en MonetDB	

- **Diseño:** Utilizando el lenguaje SQL utilizado por el MonetDB, se crear los distintos querys para crear las diferentes tablas y variables del Almacén de Datos. Al tener estos querys se puede salvar en un archivo para poder crear las tablas y variables de nuevo en cualquier otra máquina.

- **Codificación:** A continuación mostramos los querys para crear distintas tablas y variables en el idioma SQL:

```
CREATE TABLE "sys"."dim_tiempo" (
        "id_time"  INTEGER      NOT NULL      DEFAULT next value for
"sys"."seq_5515",
        "dia"     VARCHAR(2),
        "mes"     VARCHAR(2),
        "ano"     VARCHAR(4),
        "hora"    VARCHAR(2),
        "minutos"  VARCHAR(2),
        "segundos" VARCHAR(2),
        CONSTRAINT "dim_tiempo_id_time_pkey" PRIMARY KEY ("id_time")
                                        );
```

Cuadro 6 - Query para crear la Tabla Dimension Tiempo

```
CREATE TABLE "sys"."dim_sitios" (
        "id_sitio" INTEGER      NOT NULL      DEFAULT next value for "sys"."seq_5501",
        "lang"     VARCHAR(30),
        "host"     VARCHAR(30),
        "server"   VARCHAR(30),
        "protocol" VARCHAR(30),
        "domain"   VARCHAR(30),
        CONSTRAINT "dim_sitios_id_sitio_pkey" PRIMARY KEY ("id_sitio")
);
```

Cuadro 7 - Query para crear la Tabla Dimension Sitios

```
CREATE TABLE "sys"."dim_metadata" (
        "id_metadata"  INTEGER      NOT NULL      DEFAULT next value for "sys"."seq_5530",
        "name"         VARCHAR(50),
        "os"           VARCHAR(20),
        "html_version" VARCHAR(50),
        "description"  VARCHAR(50),
        "domain"       VARCHAR(50),
        CONSTRAINT "dim_metadata_id_metadata_pkey" PRIMARY KEY ("id_metadata")
);
```

Cuadro 8 - Query para crear la Tabla Dimension Metadata

```
CREATE TABLE "sys"."fact_página" (
        "id_metadata"     INTEGER      NOT NULL,
        "id_time"         INTEGER      NOT NULL,
        "id_sitio"        INTEGER      NOT NULL,
        "id_mongo"        VARCHAR(30)  NOT NULL,
        "uri"             VARCHAR(100),
        "w3c_validation"  VARCHAR(10),
        "http_status"     VARCHAR(5),
        "ip"              VARCHAR(20),
        "cantidad_flash"  VARCHAR(4),
        "cantidad_videos" VARCHAR(4),
        "cantidad_div"    VARCHAR(4),
        "cantidad_sonidos" VARCHAR(4),
        "cantidad_css"    VARCHAR(4),
        "cantidad_tables" VARCHAR(4),
        "cantidad_jpg"    VARCHAR(4),
        "cantidad_bmp"    VARCHAR(4),
        "cantidad_gif"    VARCHAR(4),
        "cantidad_png"    VARCHAR(4),
        "cantidad_scripts" VARCHAR(4),
        CONSTRAINT "fact_página_id_metadata_fkey" FOREIGN KEY ("id_metadata")
REFERENCES "sys"."dim_metadata" ("id_metadata"),
        CONSTRAINT "fact_página_id_sitio_fkey" FOREIGN KEY ("id_sitio") REFERENCES
"sys"."dim_sitios" ("id_sitio"),
        CONSTRAINT "fact_página_id_time_fkey" FOREIGN KEY ("id_time") REFERENCES
"sys"."dim_tiempo" ("id_time")
);
```

Cuadro 9 - Query para crear la tabla de Hechos Página

- **Pruebas:** Las pruebas en esta iteración se realizaron de la siguiente manera:

Numero de Caso de Prueba: 8	**Numero de Historia de Usuario:** 3
Descripción: Insertar valores de pruebas validos en las distintas tablas.	
Resultados Esperados: Al insertar nuevos valores en alguna tabla en MonetDB no se produzca ningún error.	**Resultados Obtenidos:** se insertaron datos en las distintas tablas de dimensiones y de hechos sin producir ningún error

Numero de Caso de Prueba: 9	**Numero de Historia de Usuario:** 3
Descripción: Realizar distintos querys para observar sus resultados.	
Resultados Esperados: Con los datos de prueba que están en las distintas tablas, se realiza un query avanzado y se muestra el resultado	**Resultados Obtenidos:** se ejecuto un query avanzado usando distintos Join y se mostraron los datos que tienen las distintas tablas. El query se muestra en el cuadro 10.

```
Select T4.lang,T4.host,T4.server,T4.protocol,T4.domain,T3.name,T3.os,T3.html_version,
T3.description, T1.w3c_validation, T1.http_status, T1."IP", T1.cantidad_flash, T1.cantidad_videos,
T2."day", T2."month", T2."year", T2."hour", T2."minutes" from "DIM_Sitio" as T4 inner
join("DIM_Metadata" as T3 inner join("FACT_Página" as T1 inner Join "DIM_Tiempo" as T2
on(T1.id_time = T2.id_time)) on (T3.id_metadata = T1.id_metadata)) on (T4.id_sitio = T1.id_sitio);
```

Cuadro 10 - Ejemplo de un Query en MonetDB usando el JOIN

5.1.3 ITERACIÓN 2

- **Planificación:** En esta iteración se va a desarrollar la funcionalidad más importante del todo el proyecto, que es el proceso ETL (Extracción, Transformación y Carga). Este proceso es importante en el mundo de los Almacenes de Datos, ya que se extrae la información necesaria desde cualquier fuente, luego transforma estos datos según las reglas del negocio y luego carga estos datos en Almacén de Datos.

Numero de Iteración: 2	**Numero de Historia:** 4, 5, 6 y 7
Fecha de Inicio: 01/10/2011	**Fecha de Fin:** 10/12/2011
Descripción: Proceso ETL	**Tipo:** Desarrollo

- **Historias de Usuario:**

Numero: 4	**Nombre:** Adaptación del Analizador de Documentos
Usuario: Carlos Blanco	**Iteración Asignada:** 2
Tipo: Mejora	**Tiempo Estimado:** 15 días
Descripción: Al analizador de Documentos proporcionado por María Gabriela Gaetano se le realizara unas modificaciones para que se adapte a las necesidades del Proceso ETL.	

Numero: 5	**Nombre:** Implementación en Java de los Querys SQL
Usuario: Carlos Blanco	**Iteración Asignada:** 5
Tipo: Nueva	**Tiempo Estimado:** 5 Días
Descripción: Se crea un archivo java llamando querys.java, donde va a tener todos los querys necesarios para la corrida de las distintas funciones. Este archivo es compilado y se llamara en Ruby a través de la función Popen4 para luego recibir los resultados de los querys y trabajar con los resultados en Ruby.	

Numero: 6	**Nombre:** Propuesta del Proceso ETL
Usuario: Carlos Blanco	**Iteración Asignada:** 2
Tipo: Nueva	**Tiempo Estimado:** 5 días
Descripción: Se desarrolla la propuesta del proceso ETL, en la cual se decide de cómo obtener los datos necesarios para alimentar las distintas variables del Almacén de Datos.	

Numero: 7	**Nombre:** Implementar proceso ETL
Usuario: Carlos Blanco	**Iteración Asignada:** 2
Tipo: Nueva	**Tiempo Estimado:** 40 días
Descripción: Se procede a desarrollar todo el proceso ETL, eso incluye extraer los datos necesario del MongoDB procesarlos para luego transformarlos en información que finalmente será cargada en el Almacén de Datos (MonetDB)	

- **Diseño:** En el proceso ETL, se sabe de dónde se extrae la información que se va a cargar en el Almacén de Datos, esta información proviene de la información almacenada en la base de datos MongoDB que se llama iticve_db, la estructura de datos de esta información se muestra en el (Cuadro 11). De cada elemento que se encuentra en la estructura mencionada podemos obtener distintos datos necesarios, como son: MongoID (Clave Primaria en MongoDB), URL, tipo de contenido (mime) y contenido del archivo ya que la estructura representa distintos tipos de archivos de una página web como pueden ser: los archivos HTML, archivos de imágenes, archivos de hojas de estilos (CSS), etc. Finalmente al procesar estos distintos datos según la regla de negocio se procede a cargar estos datos en el Almacén de Datos.


```
{
        "_id" : ObjectId("4f82268bb2db8520bc000002"),
        "uri" : "http://www.blogprueba.com/style.CSS",
        "contenido" :
"BAgiAgoRLyoNCglTaW1wbGUgQml6IGJ5IEFkb25pcyBSb25xdWlsbG8gZm9y\nIEZyZWUgV2Vic2l0Z
SBUZW1wbGF0ZXMNCgl3d3cuZnJlZXdlYnNpdGV0ZW1w\nbGF0LmVzIC8gd3d3LmRvbmkudXMNCg
lJbWFnZXMgYnkgSW1hZ2UgQmFzZSBo\ ",  (Se ve de esta manera el contenido porque esta
codificado)
        "document" : "true",
        "fetch_date" : "ISODate(2012-04-08T19:30:11-04:30)",
        "mime" : "text/CSS"
}
```


Cuadro 11 - Estructura de Datos en MongoDB

- **Codificación:** Primero se realiza unas modificaciones en la librería Scraper que es requerida por el analizador de documentos, estos cambios es incluir unas

expresiones regulares que detecte las líneas que contengan: imágenes, scripts, flash, etc. en el contenido HTML de las páginas, estos cambios se observan en la figura 14.

```
46  def initialize_list(doc)
47
48    # => Hash con el patron de busqueda de los diferentes elementos en un documento web
49    #    lista de los nodos que contiene el documento actual
50    @list_node_set = {
51      :table => {:name => "tablas", :pattern => ["//*/table"], :nodes => []},
52      :div => {:name => "div", :pattern => ["//*/div"], :nodes => []},
53      :jpg => {:name => "jpg", :pattern => ['//*[regex(.,".*.jpg","src")]'], :nodes => []},
54      :bmp => {:name => "bmp", :pattern => ['//*[regex(.,".*.bmp","src")]'], :nodes => []},
55      :gif => {:name => "gif", :pattern => ['//*[regex(.,".*.gif","src")]'], :nodes => []},
56      :png => {:name => "png", :pattern => ['//*[regex(.,".*.png","src")]'], :nodes => []},
57      :link => {:name => "enlaces", :pattern => ["//*/a"], :nodes => []},
58      :css => {:name => "css", :pattern => ["//*/style", "//*/link[@rel = 'stylesheet']", "//*[@style]"], :nodes => []},
59      :applet => {:name => "applets", :pattern => ["//*/applet"], :nodes => []},
60      :object => {:name => "objetos", :pattern => ["//*/object"], :nodes => []},
61      :script => {:name => "script", :pattern => ["//*/script"], :nodes => []},
62      :flash => {:name => "flash", :pattern => ['//*[regex(.,"application\/x-shockwave-flash","type")]'], :nodes => []},
63      :video => {:name=> "videos", :pattern => ['//*[regex(.,"video\/.*","type")]'], :nodes => []},
64      :audio => {:name=> "audios", :pattern => ['//*[regex(.,"audio\/.*","type")]'], :nodes => []}
65    }
66    # => Bandera que indica que se ha generado la lista de elementos del documento actual
67    @set_flag = false
68  end
```

Figura 14- Cambios realizados en la Librería Scraper

Después de realizar estos cambios creamos nuestra función analizador (ver figura 15) que lo que hace es contar las cantidades de elementos señalados en la figura anterior, también obtenemos otros datos como el titulo, metadata y tipo de codificación de la página web. Todos estos datos son almacenados en distintas variables para posteriormente se cargan en el Almacén de Datos.

```
if !html.nil?
  counter = analyzer.count_elements(html)
  counter.each { |c|
    if c[:element] == "videos"
      @videos = c[:count]
    end
    if c[:element] == "css"
      @css = c[:count]
    end
    if c[:element] == "enlaces"
      @enlaces = c[:count]
    end
    if c[:element] == "jpg"
      @jpg = c[:count]
    end
    if c[:element] == "audios"
      @audios = c[:count]
    end
    if c[:element] == "bmp"
      @bmp = c[:count]
    end
    if c[:element] == "div"
      @div = c[:count]
    end

    if c[:element] == "gif"
      @gif = c[:count]
    end
    if c[:element] == "png"
      @png = c[:count]
    end
    if c[:element] == "flash"
      @flash = c[:count]
    end
    if c[:element] == "tablas"
      @tablas = c[:count]
    end
    if c[:element] == "objetos"
      @objetos = c[:count]
    end
    if c[:element] == "applets"
      @applets = c[:count]
    end
    if c[:element] == "script"
      @scripts = c[:count]
    end
  }
```

Figura 15 - Función Analizador de Documentos

Al comenzar el proceso ETL lo primero que se hace es obtener todos los datos desde MongoDB, luego se obtiene el máximo de las claves primarias (que son numéricas) de las dimensiones Tiempo, Sitios y Metadata. Esto se hace a la hora de insertar un nuevo elemento en estas tablas para saber el nuevo valor que debe tener la siguiente clave primaria, también se hace un query que se llama

sel_factpágina para obtener todos los MongoID almacenados en la tabla de hecho páginas. Esto se puede observar en la figura 16:

```
conection = Mongo::Connection.new
db = conection.db('iticve_db')
coll = db.collection('items')

query = coll.find().to_a #Obtiene todos los elementos almacenada en la MongoDB y la pasa a un arreglo
##################################
#Obtener el maximo valor del id de cada pagina para calcular el siguiente numero para el id
POpen4::popen4("java -cp ./monetdb-1.22-jdbc.jar:./ querys sel_dimsitios") do |stdout, stderr, stdin, pid|
  @query = stdout.read.strip.split
end
POpen4::popen4("java -cp ./monetdb-1.22-jdbc.jar:./ querys sel_factpagina") do |stdout, stderr, stdin, pid|
  @query2 = stdout.read.strip.split
end
@id_time = '0'
POpen4::popen4("java -cp ./monetdb-1.22-jdbc.jar:./ querys sel_time") do |stdout, stderr, stdin, pid|
  @id_time = stdout.read.strip
end
@id_metadata = '0'
POpen4::popen4("java -cp ./monetdb-1.22-jdbc.jar:./ querys sel_metadata") do |stdout, stderr, stdin, pid|
  @id_metadata = stdout.read.strip
end
```

Figura 16 - Código para obtener valores máximo de las Dimensiones

Como se había mencionado los querys se realizan a través de java, estos querys son ejecutado desde Ruby gracia a la librería Popen4, cuando ejecutamos el query nos devuelve un string que parseamos para obtener los resultados que deseamos. Por ejemplo: Se retorna los datos 0 1 2 3 4 5 y estos datos se almacenan en un string en Ruby, que luego transformamos en un arreglo [0, 1, 2, 3, 4, 5], transformado en un arreglo ya podemos acceder más fácil a estos valores. Un ejemplo de un query utilizado para calcular el máximo número de las claves primarias en alguna de las tablas se puede observar en la figura 17:

```
} else if (args[0].equals("sel_metadata")) {

    rs = st.executeQuery("SELECT MAX(id_metadata) FROM dim_metadata");
    rs.next();
    System.out.print(rs.getString(1));
```

Figura 17 - Ejemplo de Query para calcular el Máximo de una Clave Primaria

Cuando usamos la librería Popen4 es como que ejecutaremos el comando en consola. En nuestro proyecto usamos es Java entonces la ejecución seria en consola así: java querys args(0) args(1) args(2) …, el primer argumento es para saber que query vamos a ejecutar en el caso de la Figura 21 es "sel_metadatas". En otros casos los demás argumentos es para otros valores que se van a usar en el query, como en el caso de las inserciones se pasan a través de los otro argumentos los valores que se van a insertar en la tabla.

En el proceso ETL se toma una sola vez los valores del tiempo como: dia, mes, año, hora y minutos (ver figura 18) se insertan en la tabla de dimensión de tiempo, así todos los elementos que se inserte en la tabla de hechos páginas tendrán el mismo identificador de tiempo.

```
##################################
#obtiene el tiempo actual y lo inserta en la tabla dim_tiempo
time = Time.new
@year = time.year
@month = time.month
@day = time.day
@hour = time.hour
@minutes = time.min
@seconds = time.sec
POpen4::popen4("java -cp ./monetdb-1.22-jdbc.jar:./ querys ins_time #{@day} #{@month} #{@year} #{@hour} #{@minutes} #{@seconds}")
end
```

Figura 18 - Código para Insertar en la Tabla Dimension Tiempo

Se puede observar en la figura 18 que en el comando Popen4, los destinos argumentos pasados al comando Java, recordemos que el primer argumento es para indicar que query (Ejemplo de un query ver figura 19) vamos a ejecutar y los demás argumentos son valores necesarios para el funcionamiento del query, en este ejemplo los otros argumentos son para insertarlos en la tabla dimensión tiempo.

```
} else if (args[0].equals("ins_time")) {
    st.executeUpdate("INSERT INTO dim_tiempo (dia, mes, ano, hora, minutos, segundos) VALUES ("
            + "'"
            + args[1]
            + "','"
            + args[2]
            + "','"
            + args[3]
            + "','"
            + args[4] + "','" + args[5] + "','" + args[6] + "')");
```

Figura 19 - Ejemplo de un Query para Insertar elementos

Después de insertar nuestros valores de tiempo en la tabla de dimensión tiempo se continúa con el proceso ETL, podemos observar el código donde partimos de un gran ciclo usando la variable query, tenemos que recordar que esta variable contiene un arreglo con todos los elementos que se encuentra en la base de datos iticve_db de MongoDB (Figura 24). Para cada elemento del arreglo query obtenemos el mime, uri (URL) y el contenido. Con el mime podemos saber qué tipo de contenido es el elemento que se está procesando en ese momento. En el caso que sea HTML se procede a decodificar el contenido para luego pasarlo por el analizador de documentos para obtener distintas métricas, también obtenemos el titulo, la codificación del documento y la versión de HTML, antes de insertar el nuevo contenido en las distintas tablas como son la de metadata, sitios y páginas se verifica primero el arreglo query2 que contiene todos los MongoID que se encuentra almacenado actualmente en el Almacén de Datos , esto se hace para evitar datos duplicados. Después de esta verificación se comprueba si en la tabla de dimensiones sitios ya se encuentra el host de la URL de la página que estamos procesando, si no se encuentra se inserta los elementos en la tabla de dimensión sitios y metadata, como también se insertar los valores en la tabla de hechos páginas, si el host ya se encuentra en la tabla de sitios no se inserta ningún nuevo valor en la tabla sitios, lo que se hace es obtener el valor de la clave primaria relacionada con el host ya existe. En la siguientes figuras (20-21-22-23) podemos observar un ejemplo de cómo queda almacenado los valores en las distintas tablas.

id_time	dia	mes	ano	hora	minutos	segundos
1	20	12	2011	14	30	15

Figura 20 - Ejemplo de los Datos en la Tabla Dimension Tiempo

En la figura 220 podemos ver los valores de tiempo obtenidos cuando se inicia la ejecución del proceso ETL, recordemos que estos valores se toman una sola vez por cada ejecución del proceso, todos los valores que se vayan insertar en la tabla de hecho página en la corrida tendrán el mismo valor de id_time obtenido después de insertar los datos de tabla tiempo.

id_metadata	name	os	html_version	description	domain
1	www.blogprueba.com	null	DTD_XHTML_1.0_Strict	Blog_de_Prueba_para_la_Tesis_de_Grado	www.blogprueba.com

Figura 21 - Ejemplo de los Datos en la Tabla Dimension Metadata

id_sitio	lang	host	server	protocol	domain
1	EN	www.blogprueba.com	null	http	www.blogprueba.com

Figura 22 - Ejemplo de los Datos en la Tabla Dimension Sitios

Podemos observar en la figura 22, que el host www.blogprueba.com ya se encuentra en la tabla, si procesamos otra página que sea del mismo host no se agrega valores nuevos a la tabla si no que se toma el valor del id_sitios, para cuando se vaya insertar valores relacionados a ese host en la tabla de Hechos

id_metadata	id_time	id_sitio	id_mongo	uri	w3c_validation_status	http_status	ip	cantidad_flash	cantidad_video	cantidad_div	cantidad_son	cantidad_css	cantidad_tables	cantidad_jpg	cantidad_bmp	cantidad_gif	cantidad_png	cantidad_scripts
1	1	1	4f81d39db2db850fb6000002	http://www.blogprueba.com/index.html	null	null	null	0	0	10	0	1	0	3	0	0	0	0

Figura 23 - Ejemplo de los Datos en la Tabla de Hechos Páginas

En la figura 23 se muestra los datos como quedan al ser insertados en la tabla de hechos Páginas.

```
query.each do |i|
  mime = i.values_at("mime").to_s
  uri = i.values_at("uri").to_s
  contenido = i.values_at("contenido").to_s
################################
#si el mime contiene la palabra HTML entonces procede analizar el contenido HTML
  if (mime =~ /html/)
    contenido = Marshal.load(Base64.decode64(contenido))
    analizador(contenido)
    @id_mongo = i.values_at("_id").to_s
    if !@query2.include?(@id_mongo) #si el valor id de un dato de mongoDB ya se encuentra en el DW se obvia

     if @query.include?(URI.parse(uri).host) #Si un host de una direccion ya se encuentra en la tabla dim_sitios se obvia
       @id_sitio = @query[@query.index(URI.parse(uri).host)-1]
       direccion = URI.parse(uri)
       #Se procede a realizar los distintos insert con los datos que se han obtenido
       POpen4::popen4("java -cp ./monetdb-1.22-jdbc.jar:./ querys ins_metadata #{direccion.host} #{@os} #{@html_version} #{@titulo} #{direccion.
       end
       @id_metadata = @id_metadata.to_i + 1
       POpen4::popen4("java -cp ./monetdb-1.22-jdbc.jar:./ querys ins_fact_pagina #{@id_metadata} #{@id_time} #{@id_sitio} #{@id_mongo} #{uri} #
       end
     else
       #si el host de la URL no se encuentra se inserta un nuevo valor a la tabla dim_sitios
       direccion = URI.parse(uri)
       POpen4::popen4("java -cp ./monetdb-1.22-jdbc.jar:./ querys insert_dimsitios #{@lang} #{direccion.host} #{@server} #{direccion.scheme} #{d
       end
       POpen4::popen4("java -cp ./monetdb-1.22-jdbc.jar:./ querys sel_dimsitios") do |stdout, stderr, stdin, pid|
         @query = stdout.read.strip.split
       end
       @id_sitio = @query[@query.index(URI.parse(uri).host)-1]

       #Se procede a realizar los distintos insert con los datos que se han obtenido
       POpen4::popen4("java -cp ./monetdb-1.22-jdbc.jar:./ querys ins_metadata #{direccion.host} #{@os} #{@html_version} #{@titulo} #{direccion.
       end
       @id_metadata = @id_metadata.to_i + 1
       POpen4::popen4("java -cp ./monetdb-1.22-jdbc.jar:./ querys ins_fact_pagina #{@id_metadata} #{@id_time} #{@id_sitio} #{@id_mongo} #{uri} #
       end
     end
    end
  else
```

Figura 24 - Código del Proceso ETL en el caso de que el contenido sea de tipo HTML

En el caso que el contenido en el mime no sea del tipo HTML(ver figura 26), el proceso ETL es el mismo que en caso anterior con la diferencia que no se procesa el contenido, al no ser contenido HTML nos referimos a que puede ser una imagen, un scripts, un contenido flash, un video, un CSS, entre otras. (ver figura 25)

id_metadata	id_time	id_sitio	id_mongo	uri	w3c_validation_status	http_status	ip	cantidad_fla_sh	cantidad_vid_eos	cantidad_div_os	cantidad_son_idos	cantidad_css_les	cantidad_tab_s	cantidad_jpg_s	cantidad_bmp_s	cantidad_gif_s	cantidad_png_s	cantidad_scr_ipts
3	1	1	4f81d39db2db850fb6000001	http://www.blogprueba.com/style.css	null	null	null	null	null	null	null	null	null	null	null	null	null	null

Figura 25 - Tabla de Hechos Páginas cuando el contenido procesado no es HTML

```
##################################
    #todo los siguiente hace lo mismo que el bloque anterior pero para otro elementos que no son HTML
    #Como las Imagenes, flash, css, etc
    @id_mongo = i.values_at("_id").to_s
    if !@query2.include?(@id_mongo)

      if @query.include?(URI.parse(uri).host)
        @id_sitio = @query[@query.index(URI.parse(uri).host)-1]
        direccion = URI.parse(uri)

        POpen4::popen4("java -cp ./monetdb-1.22-jdbc.jar:./ querys ins_metadata #{direccion.host} null null null #{direccion.host}") do |stdout, stde
        end

        @id_metadata = @id_metadata.to_i + 1

        POpen4::popen4("java -cp ./monetdb-1.22-jdbc.jar:./ querys ins_fact_pagina #{@id_metadata} #{@id_time} #{@id_sitio} #{@id_mongo} #{uri} null
        end
      else
        direccion = URI.parse(uri)
        POpen4::popen4("java -cp ./monetdb-1.22-jdbc.jar:./ querys insert_dimsitios null #{direccion.host} null #{direccion.scheme} #{direccion.host}
        end
        POpen4::popen4("java -cp ./monetdb-1.22-jdbc.jar:./ querys sel_dimsitios") do |stdout, stderr, stdin, pid|
          @query = stdout.read.strip.split
        end
        @id_sitio = @query[@query.index(URI.parse(uri).host)-1]

        POpen4::popen4("java -cp ./monetdb-1.22-jdbc.jar:./ querys ins_metadata #{direccion.host} null null null #{direccion.host}") do |stdout, stde
        end

        @id_metadata = @id_metadata.to_i + 1

        POpen4::popen4("java -cp ./monetdb-1.22-jdbc.jar:./ querys ins_fact_pagina #{@id_metadata} #{@id_time} #{@id_sitio} #{@id_mongo} #{uri} null
        end


      end

    end
```

Figura 26 - Código del Proceso ETL cuando el mime no es HTML

- **Pruebas:** Para la iteración 2 se realizaron las siguientes pruebas:

Numero de Caso de Prueba: 10	**Numero de Historia de Usuario:** 4
Descripción: Comprobar funcionamiento del analizador de documentos	
Resultados Esperados: Al utilizar el analizador de documento en una página web se tiene que mostrar por pantalla las distintas cantidades de elementos como son las imágenes, scripts, etc.	**Resultados Obtenidos:** Al utilizar el analizador de documentos en una página de prueba se mostro por pantalla la cantidades de elementos correctos, como por ejemplo: cantidad JPEG 5, cantidad de BMP 0, cantidad de flash: 1.

Numero de Caso de Prueba: 11	**Numero de Historia de Usuario:** 5
Descripción: Prueba de los distintos querys utilizados por el proceso ETL	
Resultados Esperados: Se espera el correcto funcionamiento de los querys como son los distintos Select y los Insert, necesario para el funcionamiento del proceso ETL	**Resultados Obtenidos:** Al ejecutar los distintos querys con datos de pruebas reales, se ejecutaron todos de manera correcta y los datos almacenado eran los que se esperaban.

Numero de Caso de Prueba: 12	**Numero de Historia de Usuario:** 7
Descripción: Prueba general del proceso ETL	
Resultados Esperados: El correcto funcionamiento del proceso ETL usando como datos de entradas el contenido de la base de datos en MongoDB.	**Resultados Obtenidos:** Los datos ingresados por el proceso ETL en el Almacén de Datos eran los esperados, según los datos de entradas.

5.1.4 ITERACIÓN 3

- **Planificación:** Como ultima iteración tenemos la interfaz de usuario, en esta interfaz el usuario podrá ver distintas versiones de una página según la URL que ingrese en la pantalla principal, para seleccionar entre las distintas versiones el usuario seleccionara la versión que desea ver seleccionándola a través de un calendario, al seleccionar la versión deseada el cliente navegara la página seleccionada.

Numero de Iteración: 3	**Numero de Historia:** 8, 9, 10, 11 y 12
Fecha de Inicio: 15/01/2012	**Fecha de Fin:** 01/03/2012
Descripción: Creación de la Interfaz de Usuario	**Tipo:** Desarrollo

- **Historias de Usuario:**

Numero: 8	**Nombre:** Desarrollo de la página WEB
Usuario: Carlos Blanco	**Iteración Asignada:** 3
Tipo: Nueva	**Tiempo Estimado:** 5 días
Descripción: Se desarrolla la página web (HTML) que va a ser vista por el cliente.	

Numero: 9	**Nombre:** Creación del ambiente de trabajo en Ruby on Rails (Aplicación Web)
Usuario: Carlos Blanco	**Iteración Asignada:** 3
Tipo: Nueva	**Tiempo Estimado:** 5 días

Descripción: Se crea el ambiente de trabajo usando Ruby on Rails, los distintos controladores y páginas se irán desarrollando en el transcurso de las necesidades que surjan, las vistas estarán en base a la interfaz anteriormente implementada.

Numero: 10	**Nombre:** Desarrollar Calendario para la Aplicación Web
Usuario: Carlos Blanco	**Iteración Asignada:** 3
Tipo: Nueva	**Tiempo Estimado:** 5 días
Descripción: Se desea implementar un calendario que en él se muestre las distintas versiones de la páginas, partiendo de la URL ingresada por el usuario. Las distintas versiones se mostraran en el día, mes y año según la fecha en la cual fueron procesadas por el proceso ETL.	

Numero: 11	**Nombre:** Extraer archivos necesarios de la Página Web
Usuario: Carlos Blanco	**Iteración Asignada:** 3
Tipo: Nueva	**Tiempo Estimado:** 30 días
Descripción: Cuando el usuario seleccione la versión que desea ver en el calendario, se procede a crear los archivos necesarios para mostrar el contenido de la página web, la información necesaria para estos archivos se extraerá desde la Base de Datos en MongoDB.	

Numero: 12	**Nombre:** Crear graficas para mostrar las métricas
Usuario: Carlos Blanco	**Iteración Asignada:** 3
Tipo: Nueva	**Tiempo Estimado:** 5 días
Descripción: En la aplicaciones web se mostrara unas graficas de barras en que se mostrara lo siguiente: 1- Cantidades de versiones totales de la página web en cada mes y en total en el año, partiendo del URL ingresado por el usuario. 2- Al seleccionar una versión desde el calendario, se mostrara una grafica en la que muestra la cantidad de elementos contenida en el archivo HTML, Ejemplo: Cantidad de imágenes, Flash, etc.	

- **Diseño:** Para desarrollar la interfaz de usuario se va a utilizar Ruby on Rails, con este Frameworks se va a tener distintas vistas que representan las distintas páginas web, los respectivos controladores que es el código encargado de manejar las distintas acciones de las distintas páginas web. Entre las páginas que se van a desarrollar tenemos como páginas principal el index, donde el usuario ingresa la URL de la página a consultar, al ingresar una URL valida se procede a mostrar la página de pre calendario donde se le va a mostrar los años en la cual hay versiones de la página, después de seleccionar un año se muestra un calendario donde muestra todas las versiones correspondiente al año que el usuario había seleccionado, finalmente selecciona una versión de la página en el calendario y finalmente puede visualizar la páginaque había seleccionado y navegar por ella con todos sus elementos.

- **Codificación:** En la codificación se va a explicar las distintas vistas con sus respectivos controladores, el orden en que se explica será el mismo orden en el que el usuario va navegando por el sistema.

  El usuario comienza con la página index que sería nuestra página inicial en el sistema, en esta página se comprobara primero si el URL que está ingresando el usuario existe en el Almacén de Datos, si existe se procede a la siguiente página del sistema que sería la de pre calendario. A continuación se muestra dos imágenes (Figura 27 y 28) que representan la vista y el controlador del index.

```
<body>
<div id="wrapper">
  <div id="header">
    <div id="logo">
      <h1><a href="#">Tesis de Grado</a></h1>
    </div>
    <div id="menu">
      <ul>
        <li class="first current_page_item"><a href="http://localhost:3000/tesis_web/">Inicio</a></li>
      </ul>
      <br class="clearfix"/>
    </div>
  </div>
  <div id="page">
    <div id="content">
      <div class="box">
        <h2>Ingrese la Direccion de la pagina Web a Consultar</h2>

        <p>
          <% form_tag(:action => 'index') do %>
              <label for="direccion">Ingrese La Direccion de la Pagina a Consultar:</label>
              <input id="direccion" value="http://" name="direccion" type="text"/> <!-- campo donde el usuario ingresa el url-->
              <input name="commit" type="submit" value="Aceptar"/>
              <font size="3" color="#FF0000"><%= @error %></font> <!-- Muestra el mensaje de error si la pagina que ingreso el usuario no existe -->

          <% end %>
        </p>
      </div>

      <br class="clearfix"/>
    </div>

    <br class="clearfix"/>
  </div>
</div>
</body>
```

Figura 27- Vista del index.html.erb

```
def index
  require 'rubygems'
  require 'popen4'

#################################
  #Crea un ID numerico para el usuario actual usando un Random
  if session[:user_id].nil?
    session[:user_id] = rand(9999999999999)
  end
#################################

#################################
  #guarda la direccion ingresadaba por el usuario en la variable $direccion
  #cheque que esa pagina existe y si es asi, va a la pagina de pre_calendario y si no muestra un mensaje de error
  $direccion = params[:direccion]
  POpen4::popen4("java -cp ../monetdb-1.22-jdbc.jar:../ querys cantidad_fact_pagina #{$direccion}") do |stdout, stderr, stdin, pid|
    @existe_pagina = stdout.read.strip.to_i
  end
  if $direccion
    if (@existe_pagina > 0)
      @error = ""
      redirect_to('/tesis_web/pre_calendario')
    else
      @error = "La Pagina Ingresada No Existe"
    end
  end
#################################
end
```

Figura 28 - Controlador de la página index.html.erb

```
} else if (args[0].equals("sel_factpagina")) {

    rs = st.executeQuery("SELECT COUNT(id_mongo) FROM fact_pagina");
    rs.next();
    int cantidad = rs.getInt(1);
    rs = st.executeQuery("SELECT id_mongo FROM fact_pagina");

    ResultSetMetaData md = rs.getMetaData();
    for (int i = 0; rs.next() && i < cantidad; i++) {
        System.out.print(rs.getString(1) + "\t");
```

Figura 29 - Query Utilizado en el controlador de index.html.erb

El query mostrado en la Figura 29 su función es la de retornar un entero, si el entero es mayor a cero eso quiere decir que existe incidencias del URL ingresado por el usuario en el Almacén de Datos.

Después de comprobarse si hay incidencia del URL ingresado por el usuario en el Almacén de Datos, se continua con la página pre_calendario (ver figura 30), en esta página lo que se muestra son los años en el que existe versiones de la página que desea ver el usuario, para que seleccione un año y continúe con la página de calendario.

```
<body>
<div id="wrapper">
  <div id="header">
    <div id="logo">
      <h1><a href="#">Tesis de Grado</a></h1>
    </div>
    <div id="menu">
      <ul>
        <li class="first current_page_item"><a href="http://localhost:3000/tesis_web/">Inicio</a></li>
      </ul>
      <br class="clearfix"/>
    </div>
  </div>
  <div id="page">
    <div id="content">
      <div class="box">
        <h2>Seleccione el año que desea consultar</h2>
        <p>
        <!--Segun la cantidad de años total, muestra los distintos años como url para continuar al calendario-->
        <!--segun el año seleccionado-->
        <table align="center" border="0" width="100%">
          <% @years.each do |i| %>
          <tr>
            <td align='center'>
              <a href="/tesis_web/calendario/<%= i %>"><%= i %></a>

            </td>
          </tr>
          <% end %>
        </table>

        </p>
      </div>

      <br class="clearfix"/>
    </div>

    <br class="clearfix"/>
  </div>
</div>
</body>
```

Figura 30 - Vista del pre_calendario.html.erb

```
  def pre_calendario
    require 'rubygems'
    require 'popen4'

##################################
    #Selecciona los distintos campos de fecha de la tabla dim_tiempo, segun la direccion web ingresada por el usuario
    POpen4::popen4("java -cp ../monetdb-1.22-jdbc.jar:../ querys sel_fecha #{$direccion}") do |stdout, stderr, stdin, pid|
      @sel_fecha = stdout.read.strip
    end
##################################

##################################
    #crea un arreglo de años para almacenar los disintos años en que existe una version de la pagina ingresada por el usuario
    @years = Array.new
    @sel_fecha.each do |i|
      @fecha_split = i.split(" ") #Los valores que retorna son 7, el cuarto elementos es el que contiene el año
      if @years.include?(@fecha_split[3])
        puts "el año esta"
      else
        @years << @fecha_split[3] #si el año no se encuentra en el arreglo, se almacena
      end
##################################

    end
```

Figura 31 - Controlador de pre_calendario.html.erb

```
else if (args[0].equals("sel_fecha")) {
  rs = st.executeQuery("SELECT COUNT(id_time) FROM dim_tiempo WHERE id_time IN (SELECT id_time FROM fact_pagina where uri like" + "'%" + args[1] + "%')");
  rs.next();
  int cantidad = rs.getInt(1);
  rs = st.executeQuery("SELECT * FROM dim_tiempo WHERE id_time IN (SELECT id_time FROM fact_pagina where uri like" + "'%" + args[1] + "%')");
  ResultSetMetaData md = rs.getMetaData();

  for (int i = 0; rs.next() && i < cantidad; i++) {
     System.out.print(rs.getString(1) + " " + rs.getString(2) + " " + rs.getString(3) + " " + rs.getString(4) + " " + rs.getString(5) + " " + rs.getString(6) + " " + rs.getString(7) + "\n");
  }
```

Figura 32 - Query utilizado en el controlador de pre_calendario.html.erb

En el query de la figura 32 se retorna todos los elementos que se encuentre en la tabla de dimensión tiempo que coincidan con el id_time que obtenemos de la tabla de hecho según la URL ingresada por el usuario.

Después que el usuario selecciona un año, se muestra la página de calendario, en la que se muestra todas las versiones almacenada en el Almacén de Datos para el año que se selecciono anteriormente. Las versiones se muestran según el dia, mes y hora en la que fue almacenada en el Almacén de Datos. Por ejemplo si una versión fue almacenada el 1º de Enero del 2012 a las 13 horas, en el calendario se mostrara un evento el dia 1º de Enero del 2012, el evento tendrá por nombre la hora en la que fue almacenada la versión. El usuario para continuar tiene que darle click en alguno de estos eventos y a continuación se le abrirá una nueva página donde se le muestra la página que él desea ver.

```
<script type='text/javascript'>

    $(document).ready(function() {

//        Se almacena en el calendario los dintitos eventos para enero.
//        tambien se almacena el año a mostrar
//        lo mismo se hace con cada mes en el calendario
        $('#calendar').fullCalendar({
            month:0,
            year:<%=@year%>,
            events: [
                <% if !@ene.empty? %>
                <%=@ene%>
                <%end%>
            ]
        });

    });

    $(document).ready(function() {

        $('#calendar1').fullCalendar({
            month:1,
            year:<%=@year%>,
            events: [
                <% if !@feb.empty? %>
                <%=@feb%>
                <%end%>
            ]
        });

    });
    $(document).ready(function() {
        $('#calendar2').fullCalendar({
            month:2,
            year:<%=@year%>,
            events: [
                <% if !@mar.empty? %>
                <%=@mar%>
                <%end%>
            ]
        });

    });
```

Figura 33 - Script donde se almacena los eventos en el calendario

En la figura 33 se puede observar el script donde se crean los distintos eventos en el calendarios, estos eventos son creados en el controlador y pasados a través de una variable que es diferentes para cada mes del años, estas variables contiene la estructura de datos necesaria para que el calendario funcione sin ningún problema y muestre los distintos eventos en todo el año.

```
    <!--se muestra cada mes del ca
    <div id='calendar'></div>
  </td>
  <td>
    <div id='calendar1'></div>
  </td>
</tr>
<tr>
  <td>
    <div id='calendar2'></div>
  </td>
  <td>
    <div id='calendar3'></div>
  </td>
</tr>
<tr>
  <td>
    <div id='calendar4'></div>
  </td>
  <td>
    <div id='calendar5'></div>
  </td>
</tr>
<tr>
  <td>
    <div id='calendar6'></div>
  </td>
  <td>
    <div id='calendar7'></div>
  </td>
</tr>
<tr>
  <td>
    <div id='calendar8'></div>
  </td>
  <td>
    <div id='calendar9'></div>
  </td>
</tr>
<tr>
  <td>
    <div id='calendar10'></div>
  </td>
  <td>
    <div id='calendar11'></div>
  </td>
</tr>
```

Figura 34 - Vista del calendario.html.erb

En la figura 34 se muestra en código HTML donde se mandan a imprimir los distintos meses del año.

```
def calendario
  require 'rubygems'
  require 'popen4'
  #Obtiene los disntinos valores de la tabla dim_tiempo segun el año y  la direccion ingresada por el usuario
  POpen4::popen4("java -cp ../monetdb-1.22-jdbc.jar:../ querys sel_fechas_calendario #{$direccion} #{params[:id]}") do |stdout, stderr, stdin, pid|
    @sel_fecha = stdout.read.strip
  end

  #variables necesarias para el calendario
  @ene = ""
  @feb= ""
  @mar = ""
  @abr = ""
  @may = ""
  @jun = ""
  @jul = ""
  @ago = ""
  @sep = ""
  @oct = ""
  @nov = ""
  @dic = ""
  @year = ""
  #Aqui se crea los dintintos eventos para el calendario
  @sel_fecha.each do |i| #para cada elementos del arreglo obtenido del query anterior hacemos lo siguiente
    @eventos = i.split   #arreglo en que se va a trabajar
    if @eventos[2] == '1' #si el tercer elemento del arreglo es 1 quiere decir que hay eventos de enero
      @eventos[2] = 1 - 1
      #concatenamos en enero los ditintos valores para que se muetre en calendario como: hora, minutos, segundo en title
      #tambien se usa el dia y mes para saber que dia se va a crear el evento
      #y al final se crea un url para cuando se de click en el evento se siga a la siguiente pagina, pasando por url
      #los siguiente datos Id_time de la tabla dim_tiempo y  url que se encuentra en direccion
      @ene.concat("{title: '#{@eventos[4]}:#{@eventos[5]}:#{@eventos[6]}', start: new Date(#{@eventos[3]}, #{@eventos[2]}, #{@eventos[1]}),
      url: 'http://localhost:3000/tesis_web/procesar_pagina/#{@eventos[0]}?url=#{$direccion}'},")
```

Figura 35 - Parte del código del controlador de calendario.html.erb

El Query utilizado en la Figura 35, tiene la misma función que el query mostrado en la Figura 32, con la diferencia que los resultados se acotan al año que el usuario selecciono en la página de pre_calendario.

También en la página de calendario se muestra un grafico de barra en la que se muestra la cantidad de versiones por mes en el año seleccionado por el usuario, este grafico se usa para demostrar esas métricas al usuario. A continuación se mostrara unas imágenes (Figura 36 y 37) que muestra el código usado para este grafico.

```
<div class="box">
  <h2>Resultados para la Pagina: <%= $direccion %></h2>
  <!--se muestra la grafica con la cantidad de versiones de cada mes segun año y el URL-->
  <%= @graph %>
```

Figura 36 - Porción del Código donde se imprime el Grafico 1

En la figura 40 se muestra la porción de código en donde imprime el grafico en la vista del calendario.html.erb, este grafico se muestra como un grafico de barra usando flash y Java Script, el código para obtener el valor de las distintas variables para el grafico se ve en la figura 39

```
###################################
    #Se obtiene los datos necesarios para la grafica que muestra la cantidad de versiones por mes en un año dado
    @datos_grafico = Array.new

    for i in (1..12) #se cuenta la cantidad de versiones de la pagina en cada mes y se concatena en un arreglo
      POpen4::popen4("java -cp ../monetdb-1.22-jdbc.jar:../ querys grafico_1 #{$url_completa} #{i} #{@year}") do |stdout, stderr, stdin, pid|
        @grafico_1 = stdout.read.strip
      end
      @datos_grafico << @grafico_1.to_i
    end
###################################

###################################
    #se suma cada elemento del arreglo para saber la cantidad total de versiones en un año de la pagina web
    @cantidad_total = 0
    @datos_grafico.each do |suma|
      @cantidad_total = @cantidad_total + suma
    end

###################################
    #se crea la variable necesaria para el grafico con los datos que necesita, pasados por url
    # primero el maximo valor obtenido en el arreglo para saber el tamaño maximo de valores en el EJE y
    #segundo los datos del arreglo y por ultimo la cantidad total del arreglo
    @graph = open_flash_chart_object(450, 150, "/tesis_web/grafico_barra_1/#{@datos_grafico.max}?datos=#{@datos_grafico}&total=#{@cantidad_total}")
###################################
```

Figura 37 - Código del Grafico 1 en el controlador de calendario.html.erb

```
else if (args[0].equals("grafico_1")) {
  rs = st.executeQuery("SELECT COUNT(*) FROM fact_pagina WHERE uri = " + "'" + args[1] + "'" + " and id_time in (SELECT id_time FROM dim_tiempo WHERE mes = " + args[2] + " AND ano = " + args[3] + ")");
  rs.next();
  int cantidad = rs.getInt(1);
  System.out.print(cantidad);
```

Figura 38 - Query utilizado en el Grafico 1

En el query mostrado en la figura 38 se obtiene la cantidad de versiones una página para un mes y año determinado, el resultado se retorna como un entero.

```
  def grafico_barra_1
##################################
    #definicion para el grafico de cantidad total de versiones de cada pagina
    title = Title.new("Cantidad Total de Versiones almacenadas: #{params[:total]}")
    bar = Bar.new
    bar.values = params[:datos].to_s.split(//).map(&:to_i) #datos obtenidos para cada mes analizado
    x_axis = XAxis.new
    x_axis.labels = ["Ene", "Feb", "Mar", "Abr", "May", "Jun", "Jul", "Ago", "Sep", "Oct", "Nov", "Dic"]

    y_axis = YAxis.new
    y_axis.set_range(0, params[:id], 1)  #establece el rango del grafico en el eje Y, comenzando de 0 hasta el valor maximo
                                         #obtenido del arreglo de datos
    chart = OpenFlashChart.new
    chart.set_title(title)
    chart.add_element(bar)
    chart.x_axis = x_axis
    chart.y_axis = y_axis
    render :text => chart.to_s
##################################
  end
```

Figura 39 - Método que define las variables usadas para la Grafica 1

Finalmente después que el usuario seleccione la versión de la página en el calendario se abre una ventana con la página completa que el usuario deseaba ver, incluyendo todos sus elementos como son las imágenes, flash, etc.

En este proceso se crean los archivos desde la base de datos en MongoDB, partiendo los MongoID que se obtuvieron según la versión seleccionada por el usuario, teniendo los MongoID sabemos que archivos en específico se van a seleccionar para la versión de la página web que se desea ver. A continuación se van a mostrar unas imágenes (Figura 40 y 41) con los distintos códigos usados para este proceso.

```
<FRAMESET ROWS=225,*>

  <!--En el frame de arriba se muestra la grafica de cada elemento de la pagina web-->
  <!--En el frame de abajo se muestra la pagina completa que selecciono el usuario-->
  <FRAME SRC="/tesis_web/frame/<%=params[:id]%>" NAME="arriba" >
  <FRAME SRC="<%="/#{session[:user_id]}/#{@pagina}"%>" NAME="abajo">

</FRAMESET>
```

Figura 40 - Vista de procesar_página.html.erb

En la Figura 40 podemos observar que la vista de procesar_página la dividimos en dos frames, en el frame de arriba se mostrara la Grafica 2 que se explicara más adelante y en el Frame de abajo se abrirá la versión de la página que selecciono el usuario.

```
conection = Mongo::Connection.new
db = conection.db('iticve_db')
coll = db.collection('items')
#se obtiene los mongoid de la pagina segun el host de la direccion en $direccion y  el id_time segun la seleccion del calendario
POpen4::popen4("java -cp ../monetdb-1.22-jdbc.jar:../ querys sel_mongoid #{@host} #{params[:id]}") do |stdout, stderr, stdin, pid|
  @sel_idmongo = stdout.read.strip.split
end
#se crea los distintos directorios necesarios usando como nombre para el directorio principal el id del usuario
if !File.directory?("./public/#{session[:user_id]}")
  Dir.mkdir("./public/#{session[:user_id]}")
  Dir.mkdir("./public/#{session[:user_id]}/img")
  Dir.mkdir("./public/#{session[:user_id]}/css")
  Dir.mkdir("./public/#{session[:user_id]}/media")
  Dir.mkdir("./public/#{session[:user_id]}/scripts")
end
@sel_idmongo.each do |i| #para cada mongoid que retorno el query se hace lo siguiente
  arr = coll.find({"_id" => BSON::ObjectId(i)}).to_a #se pasa a un arreglo cada elemento obtenido del mongodb segun su id
  if arr[0].values_at("mime").to_s =~ /html/  #si el mime obtenido contiene la palabra HTML se hace lo siguiente
    #se crea el archivo usando el nombre del archivo al final de cada URL y se crea un documento nokigiri llamado web_doc
    file = File.new("./public/#{session[:user_id]}/#{arr[0].values_at("uri").to_s.split("/").last}", "w")
    web_doc = Nokogiri::HTML(Marshal.load(Base64.decode64(arr[0].values_at("contenido").to_s)))
    #Se procede a cambiar las direcciones absoluta a direcciones relativas para cada elemento
    #que lo necesite en el html
    web_doc.css("a").each do |link|
      if !link.attributes["href"].nil?
        archivos = link.attributes["href"].value.to_s.split("/") #se onbtine el url que esta en el valor de href
        link.attributes["href"].value = "#{archivos.last}"  #se cambia el valor de href por uno nuevo
                                                            #mateniendo el mismo nombre del archivo pero con carpeta diferente
      end
    end
    #lo mismo que para el tag <a> se aplica para los demas tag en el contenido HTML
    web_doc.css("link").each do |link|
      if !link.attributes["href"].nil?
        archivos = link.attributes["href"].value.to_s.split("/")
        link.attributes["href"].value = "./css/#{archivos.last}"
      end
    end

    web_doc.css("img").each do |link|
      if !link.attributes["src"].nil?
        archivos = link.attributes["src"].value.to_s.split("/")
        link.attributes["src"].value = "./img/#{archivos.last}"
      end
    end
```

Figura 41 -Código de cambio de dirección absoluta por dirección relativas

En la figura 41 podemos observar como comienza este proceso, primero se crea una lista con todos los MongoID para poder extraer de la base de datos de MongoDB los elementos necesarios para mostrar la versión que selecciono el usuario anteriormente, luego se crean los diferentes directorios para almacenar el contenido de la página web, el directorio raíz va a tener como nombre un id de sesión usuario que es un numero al random que cambia cada vez que el usuario incia el sistema abriendo la aplicación web en una sección, así aseguramos que los archivos de los distintos usuarios que estén usando el sistema al mismo tiempo no se combinen, de subcarpetas va a tener un directorio para cada grupo de elementos web como por ejemplo: para las imágenes se va a tener una carpeta llamada img, etc.

Cuando se terminen de crear los directorio extraemos el contenido para los distintos MongoID que estamos trabajando (recordemos la estructura de cada datos de MongoDB en el Cuadro 11), cuando el elemento mime es igual a HTML, decodificamos el contenido usando base64.decode64 para obtener el contenido de manera legible del HTML, ese contenido HTML lo convertimos en un documento Nokogiri, que es una librería que nos permite parsear los documentos HTML, así podemos cambiar de manera sencilla las direcciones absolutas por direcciones relativas, lo podemos observar en el siguiente ejemplo:

```
if !link.attributes["href"].nil?
        archivos = link.attributes["href"].value.to_s.split("/")
        link.attributes["href"].value = "./CSS/#{archivos.last}"
 end
```

En este ejemplo cambiamos la dirección de donde se encuentra originalmente una hoja de estilo (CSS) por una dirección en donde se va encontrar en el servidor de manera local.

```
else if (args[0].equals("sel_mongoid")) {
  rs = st.executeQuery("SELECT COUNT(id_mongo) FROM fact_pagina where uri like" + "'%" + args[1] + "%' AND id_time = " + args[2]);
  rs.next();
  int cantidad = rs.getInt(1);
  rs = st.executeQuery("SELECT id_mongo FROM fact_pagina where uri like" + "'%" + args[1] + "%' AND id_time = " + args[2]);
  ResultSetMetaData md = rs.getMetaData();

  for (int i = 0; rs.next() && i < cantidad; i++) {
      System.out.print(rs.getString(1) + "\n");
  }
```

Figura 42 - Query utilizado en el proceso de la Figura 45

En el query de la figura 42 utilizado para el controlador de procesar_página, obtenemos la lista de MongoID según el URL ingresado por el usuario y se acota el

resultado al pasarle también al query la fecha y hora que selecciono el usuario en el calendario.

Después de cambiar las direcciones absolutas a direcciones relativas en el documento HTML, se crean los archivos HTML en el directorio creado al principio del proceso.

Cuando el archivo a procesar no es un HTML, se crea los archivos en sus carpetas correspondiente después que se decodifique el contenido con base64.decode4, el nombre de cada archivo lo obtenemos de la URL almacenada en la estructura en MongoDB, de ese URL tomamos el ultimo valor y tenemos el nombre del archivo a crear. Por ejemplo: http://www.prueba.com/index.html el nombre del archivo a crear es index.html así es igual con los demás elementos. A continuación se mostrara en la Figura 43 la porción de código para crear los demás elementos que no son HTML.

```
    if arr[0].values_at("mime").to_s =~ /css/
      file = File.new("./public/#{session[:user_id]}/css/#{arr[0].values_at("uri").to_s.split("/").last}", "w")
      if file
        file.syswrite((Marshal.load(Base64.decode64(arr[0].values_at("contenido").to_s))).gsub("url(images/", "url(/#{session[:user_id]}/img/"))
      else
        puts "No se Puede Abrir el Archivos"
      end
      file.close
    end
    if arr[0].values_at("mime").to_s =~ /image/
      file = File.new("./public/#{session[:user_id]}/img/#{arr[0].values_at("uri").to_s.split("/").last}", "w")
      if file
        file.syswrite(Marshal.load(Base64.decode64(arr[0].values_at("contenido").to_s)))
      else
        puts "No se Puede Abrir el Archivos"
      end
      file.close
    end
    if arr[0].values_at("mime").to_s =~ /flash/
      file = File.new("./public/#{session[:user_id]}/media/#{arr[0].values_at("uri").to_s.split("/").last}", "w")
      if file
        file.syswrite(Marshal.load(Base64.decode64(arr[0].values_at("contenido").to_s)))
      else
        puts "No se Puede Abrir el Archivos"
      end
      file.close
    end
    if arr[0].values_at("mime").to_s =~ /javascript/
      file = File.new("./public/#{session[:user_id]}/scripts/#{arr[0].values_at("uri").to_s.split("/").last}", "w")
      if file
        file.syswrite(Marshal.load(Base64.decode64(arr[0].values_at("contenido").to_s)))
      else
        puts "No se Puede Abrir el Archivos"
      end
      file.close
    end
    #si el URL ingresado por el usuario no contiene ninguna pagina se le concatena el index y la extension necesaria
    if @pagina.nil?
      arreglo = Dir.entries("./public/#{session[:user_id]}/");
      arreglo.each do |i|
        if i.include? "index"
          @pagina = i
          break
        end
      end
    end
  end
end
```

Figura 43 - Porción del código donde se crea los archivos que no son HTML

Anteriormente que procesar_página contenía dos frames, el de arriba para el Grafico 2 y el de abajo donde se va a mostrar la versión de la página seleccionada por el usuario.

En el frame de arriba se muestra una grafica que muestra la cantidad total de los elementos seleccionados como métricas para mostrar el usuario, este

grafico mostrara la cantidad total de JPEG, BMP, flash, scripts, etc. De la página que usuario selecciono.

Estos datos son obtenidos en el Almacén de Datos. A continuación se mostrara unas imágenes (Figura 44 y 45) con los códigos relacionado con este grafico.

```
<html>
<head>
  <title>Tesis </title>
  <link href="/stylesheets/style.css" media="screen" rel="Stylesheet" type="text/css">
  <script type="text/javascript" src="/javascripts/swfobject.js"></script>
</head>

<body>
<p></p>
<p></p>
<!--Se muestra la grafica que contiene la cantidad de elementos que estan en la pagina web-->
<div align="center"><%= @graph %></div>
</body>
</html>
```

Figura 44 - Vista para el Frame de Grafico 2

```
def frame
##################################
    #Como para el grafico para mostrar la cantidad de versiones de una pagina se hace lo mismo para mostrar
    #la cantidad de elementos que tiene una pagina como: imagenes, flash, etc
    POpen4::popen4("java -cp ../monetdb-1.22-jdbc.jar:../ querys grafico_2 #{$url_completa} #{params[:id]}") do |stdout, stderr, stdin, pid|
      @grafico_2 = stdout.read.strip.split.map(&:to_i)
    end
    @cantidad_total = 0
    @grafico_2.each do |suma|
      @cantidad_total = @cantidad_total + suma
    end

    @graph = open_flash_chart_object(450, 150, "/tesis_web/grafico_barra_2/#{@grafico_2.max}?datos=#{@grafico_2}&total=#{@cantidad_total}")
  end
##################################
```

Figura 45 - Controlador para el Frame de Grafico 2

El proceso es muy parecido para el Grafico 1, lo que varia es el query que se usa para tomar los datos necesarios para el grafico 2, que se mostrara a continuación en la Figura 50.

```
else if (args[0].equals("grafico_2")) {

  rs = st.executeQuery("SELECT cantidad_flash, cantidad_videos, cantidad_jpg, cantidad_bmp, cantidad_gif, cantidad_png, cantidad_scripts  FROM fact_pagina WHERE uri = " + "'" + args[1] + "'" + " AND id_time = " + args[2]);
  ResultSetMetaData md = rs.getMetaData();

  for (int i = 0; rs.next() && i < 1; i++) {
     System.out.print(rs.getString(1) + " " + rs.getString(2) + " " + rs.getString(3) + " " + rs.getString(4) + " " + rs.getString(5) + " " + rs.getString(6) + " " + rs.getString(7) + "\n");
  }
```

Figura 46 - Query para obtener los datos necesarios para el Grafico 2

El query retorna la cantidad_flash, cantidad_jpg, cantidad_bmp, cantidad_gif, cantidad_png y cantidad_scripts almacenadas en la tabla de hechos Páginas en el Almacén de Datos, este resultado se filtra usando el URL que ingreso el usuario al principio del sistema y usando también id_time según la fecha y hora que está relacionada con la selecciono del usuario en el calendario.

En la figura 47 se muestra el código donde se define las distintas variables necesarias para el correcto funcionamiento del Grafico 2.

```
  def grafico_barra_2
#################################
    #lo mismo que en grafica_1 pero para el grafico que muestra la cantidad de elementos de una pagina
    title = Title.new("Cantidad Totales para Elementos de la Pagina")
    bar = Bar.new
    bar.values = params[:datos].to_s.split(//).map(&:to_i)
    x_axis = XAxis.new
    x_axis.labels = ["Flash", "Videos", "JPG", "BMP", "GIF", "PNG", "Scripts"]

    y_axis = YAxis.new
    y_axis.set_range(0, params[:id], 1)

    chart = OpenFlashChart.new
    chart.set_title(title)
    chart.add_element(bar)
    chart.x_axis = x_axis
    chart.y_axis = y_axis
    render :text => chart.to_s
#################################
  end
```

Figura 47 - Variables necesarias para el Grafico 2

CONCLUSIONES

Al concluir este trabajo de grado se logró los objetivos planteados en el capítulo 1 de este documento, ah continuación se explicara un poco sobre las metas cumplidas:

Para el diseño del Almacén de Datos se utilizo la metodología de Kimball que nos plantea el diseño en 4 pasos, al aplicar estos pasos se pudo diseñar la tabla de hechos, las tablas de dimensiones, las métricas y la granularidad necesaria para cumplir con uno de los objetivos anteriormente planteados. Los distintos atributos de las tablas incluyendo las métricas se obtuvieron de los documentos web almacenados en la Base de Datos NoSql orientada a documento MongoDB.

Al tener el diseño del Almacen de datos, se implementa en la Base de Datos Orientada a Columnas MonetDB, se utilizo el lenguaje SQL para los distintos querys necesario para la implementación del Almacen de Datos.

Con el Almacen de Datos implementado se procede a alimentar los distintos atributos de la tabla de hechos y las tablas de dimensiones utilizando distintos datos obtenidos de los documentos web almacenados, para lograr esto se desarrollo el proceso ETL, que se encarga de todo el proceso para alimentar al Almacen de Datos.

Finalmente se desarrollo la aplicación web utilizando el Framework Ruby on Rails que usa el Lenguaje Ruby como base; la aplicación web sirve para que los usuarios en general puedan ver las distintas versiones de los documentos web almacenados y poder navegar en estos documentos, también pueden ver las distintas métricas almacenadas en el Almacen de Datos a través de unas graficas que se muestran.

Hay que destacar que La metodología XP ayudó bastante para el desarrollo de este proyecto, ya que permitió organizar desarrollo del sistema en diferentes iteraciones, en la cual cada iteración tenía su fase de planificación, diseño, codificación y pruebas. La fase de

prueba fue fundamental para mejorar mucho el código de algunas iteraciones, también ayudo a mejorar otros aspectos del proyecto.

Para finalizar se espera que este modulo ayude al proyecto del Centro de Computación Paralela y Distribuida (CCPD) logre su meta principal es la preservación digital de los documentos web.

RECOMENDACIONES

En el presente trabajo de grado se trabajó en la base de datos que proviene de MongoDB esta base de datos se llama iticve_db, los datos que se insertan en esa base de datos son muy limitados, no almacenan hojas de estilos, frash, scripts entre otros. Se recomienda totalmente incorporar más funcionalidades al spider para que almacene mucho más elementos relacionados con las páginas web en el MongoDB.

Otra importante recomendación es la de tratar de poner operativa la librería que conecta Ruby con MonetDB que es el manejador de Almacenes de Datos, ya que usando la librería Popen4 que ejecuta los querys usando java y luego los resultados los almacena como string en Ruby, se pierde ciclos del procesador por la cual puede afectar el rendimiento del sistema cuando se va a ejecutar querys pesados.

En la función de procesar_página, en donde se cambia las direcciones absolutas a direcciones relativas se cambiaron las direcciones de los elementos más comunes de un documento HTML, como son las hojas de estilos, los scripts, los flash, etc. Se recomienda trabajar más en esta parte para agregar mucho más elementos HTML.

Las carpetas donde se almacenan los archivos de las páginas que son extraídas desde MongoDB por petición del usuario no se borran, se recomienda crear un script que elimine estas carpetas con alguna condición, para evitar que se sobrecargue el disco duro del servidor con todas estas carpetas creadas cuando el usuario selecciona una versión a ver de una página.

Como el principal principio de todo el sistema es el de guardar páginas web especialmente de Venezuela se deberá desarrollar unos módulos para que se almacene nada mas páginas hechas por venezolano o almacenadas en Venezuela.

BIBLIOGRAFÍA

Alvarez, W. (28 de Diciembre de 2008). Obtenido de http://damncorner.blogspot.com/2008/12/la-aventura-de-ruby-on-rails-el-inicio.html

Archive, I. (2001). Obtenido de http://archive.org/about/faqs.php

Bi-Argentina. (10 de Marzo de 2008). Obtenido de http://www.bi-argentina.com.ar/que-es-un-data-warehouse/

Bombadil. (02 de Enero de 2012). Obtenido de http://bosqueviejo.net/2012/01/02/las-gemas-de-ruby/

Buigues, A. (14 de Enero de 2012). Obtenido de http://anabuigues.com/2010/01/14/data-warehouse-y-las-bases-de-datos-operacionales/

Curto, J. (19 de Noviembre de 2007). Obtenido de http://informationmanagement.wordpress.com/2007/11/19/diseno-de-un-data-warehouse-estrella-y-copo-de-nieve/

Curto, J. (16 de Octubre de 2007). Obtenido de http://informationmanagement.wordpress.com/2007/10/16/diseno-de-un-data-warehouse-definiciones/

Fonollosa, C. L. (Diciembre de 2005). Obtenido de http://www.ub.edu/bid/15lluec2.htm

Herrera, C. (30 de 10 de 2007). Obtenido de http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=datawarehouse#sdfootnote1anc

Inteligencias de Negocios. (2009). *Inteligencias de Negocios.* Obtenido de http://www.leroot.com/site/index.php?option=com_content&view=article&id=63&Itemid=69

Jeffries, R. (08 de Noviembre de 2001). *What is Extreme Programming?* Obtenido de http://xprogramming.com/xpmag/whatisxp

Martinez, D. (21 de Julio de 2010). Obtenido de http://blog.danielmartinez.info/?p=27

MonetDB. (s.f.). Obtenido de http://www.monetdb.nl/Assets/embedded.pdf

Padicat. (2011). Obtenido de http://www.padicat.cat/es/conocenos

Paramio, C. (26 de Abril de 2011). Obtenido de http://www.genbetadev.com/bases-de-datos/el-concepto-nosql-o-como-almacenar-tus-datos-en-una-base-de-datos-no-relacional

Paramio, C. (10 de Mayo de 2011). Obtenido de http://www.genbetadev.com/bases-de-datos/una-introduccion-a-mongodb

Paredes Villarreal, F. A. (2007). Obtenido de http://homepages.mty.itesm.mx/al792696/Extremeprogramming.doc

Patricio, L. (2006). *Metodologías Ágiles para el desarrollo de software: Extreme Programming.* Universidad Politécnica de Valencia.

Pérez, F. F. (23 de Febrero de 2011). Obtenido de http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=introduccion_bases_de_datos_nosql

qbitácora. (19 de Abril de 2010). Obtenido de http://qbitacora.wordpress.com/2010/04/19/wayback-machine-archivando-internet/

Rancel, M. R. (2011). Obtenido de http://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=73:ruby-on-rails-plataforma-de-software-libre-para-desarrollos-web&catid=46:lenguajes-y-entornos&Itemid=163

Rivadera, G. R. (2010). *La Metodología de Kimball para el Diseño de almacenes.*

Santos, R. D. (2006). Obtenido de http://www.maestrosdelweb.com/editorial/rubyonrails/

Santos, R. D. (s.f.). *Maestros del Web.* Obtenido de http://www.maestrosdelweb.com/editorial/rubyonrails/

Sinnexus. (2007). Obtenido de http://www.sinnexus.com/business_intelligence/datawarehouse.aspx

Softpedia. (2007). Obtenido de http://www.softpedia.com/es/programa-MonetDB-69281.html

Swiatecka, A. (31 de Enero de 2008). Obtenido de http://www.slideshare.net/anansi/preservacin-digital-2

Ubuntu Life. (13 de Abril de 2010). Obtenido de http://ubuntulife.wordpress.com/2010/04/13/introduccion-a-la-base-de-datos-nosql-mongodb-instalacion-primeros-pasos-y-ejemplo-de-conexion-con-java/

Xqbot. (3 de Julio de 2009). Obtenido de http://es.wikipedia.org/wiki/RubyGems

APÉNDICE

5.2 Modelo del Almacén de Datos

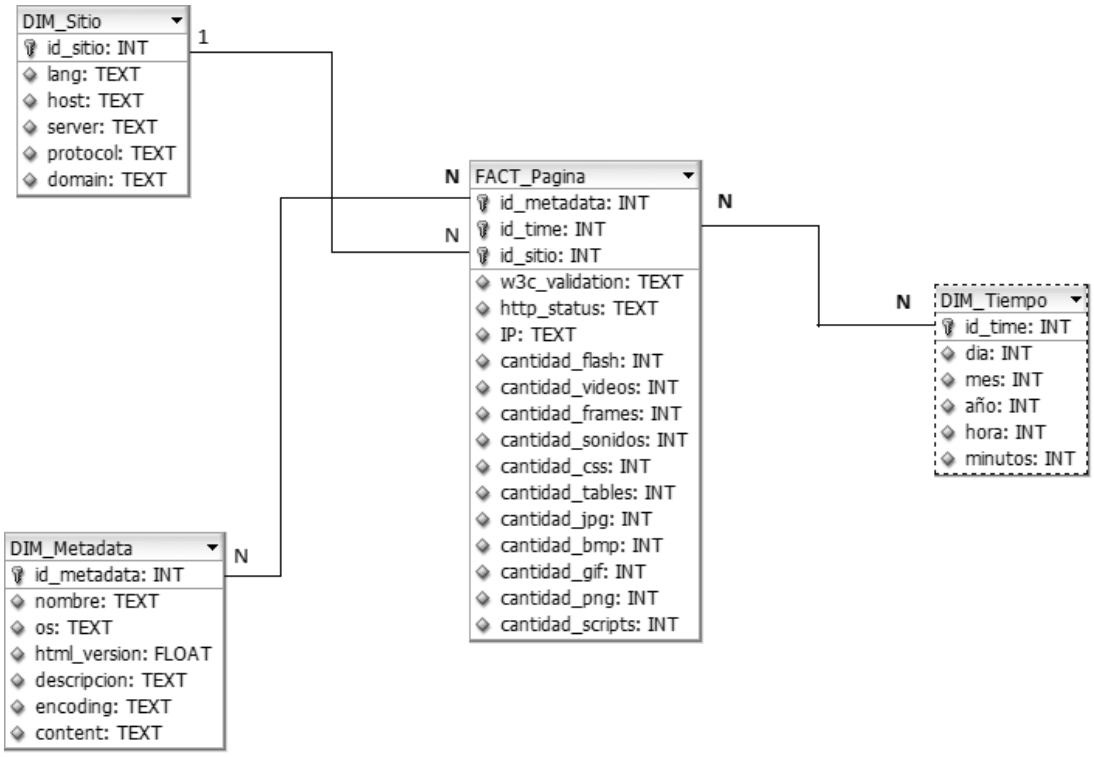

Figura 48 - Modelo del Almacén de Datos

Se puede observar en la Figura 48, que el Almacén de Datos sigue un modelo de estrella.

5.3 Diccionario de Datos del Almacén de Datos

Tabla de Hechos FACT_página	
Campos	**Descripción**
id_metadata	Clave foranea de la tabla DIM_Metadata
Id_time	Clave foranea de la tabla DIM_Tiempo
Id_sitio	Clave foranea de la tabla DIM_Sitio
W3c_validation	Se almacena Si o No si la página tiene una validación W3c en otro caso se almacena NULL.
http_status	Se almacena SI o No si la página esta online si no se sabe se almacena NULL.
IP	Se almacena la dirección IP de la página si se conoce en otro caso se almacena NULL.
cantidad_flash	Se almacena la cantidad de flash que contenga la página.
cantidad_videos	Se almacena la cantidad de videos que contenga la página.
cantidad_frames	Se almacena la cantidad de frames que contenga la página.
cantidad_sonidos	Se almacena la cantidad de sonidos que contenga la página como (mp3, midi, etc.)
cantidad_css	Se almacena la cantidad de hojas de estilos (CSS) que contenga la página.
cantidad_tables	Se almacena la cantidad de tables que contenga la página.
cantidad_jpg	Se almacena la cantidad de imágenes JPEG que contenga la página.
cantidad_bmp	Se almacena la cantidad imágenes BMP de que contenga la página.
cantidad_gif	Se almacena la cantidad imágenes gif de que contenga la página.
cantidad_png	Se almacena la cantidad de imágenes png que contenga la página.
cantidad_scripts	Se almacena la cantidad de scripts que contenga la página.

Cuadro 12 - Descripción de la Tabla de Hechos FACT_Página

Tabla de Dimensiones DIM_Sitio	
Campos	**Descripción**
Id_sitio	Clave primaria de la tabla DIM_Sitio
lang	Almacena en qué idioma esta la página en otro caso almacena NULL.
host	Almacena el host en que esta almacenada la página.
server	Almacena el nombre del servidor en donde está la página en otro caso almacena NULL.
protocol	Almacena el protocola que usa la página.
domain	Almacena el dominio en que esta la página.

Cuadro 13 - Descripción de la Tabla de Dimensiones DIM_Sitio

Tabla de Dimensiones DIM_Metadata	
Campos	**Descripción**
Id_metadata	Clave primaria de la tabla DIM_Metadata
nombre	Almacena el titulo de la página, en otro caso almacena NULL.
OS	Almacena el Sistema Operativo del servidor donde está alojada la página, en otro caso almacena NULL.
html_version	Almacena la versión de HTML de la página, en otro caso almacena NULL.
descripción	Almacena la descripción de la página, en otro caso almacena NULL.
encoding	Almacena la codificación de la página, en otro caso almacena NULL.
content	Almacena el tipo de contenido de la página, en otro caso almacena NULL.

Cuadro 14 - Descripción de la Tabla de Dimensiones DIM_Metadata

Tabla de Dimensiones DIM_Tiempo	
Campos	**Descripción**
Id_time	Clave primaria de la tabla DIM_Tiempo
dia	Almacena el dia en que fue almacenada la información de la página en el Almacén de Datos
mes	Almacena el mes en que fue almacenada la información de la página en el Almacén de Datos
ano	Almacena el año en que fue almacenada la información de la página en el Almacén de Datos
hora	Almacena la hora en que fue almacenada la información de la página en el Almacén de Datos
minutos	Almacena los minutos en que fue almacenada la información de la página en el Almacén de Datos

Cuadro 15 - Descripción de la Tabla de Dimensiones Tiempo