UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Ingeniería técnica en informática de gestión

Proyecto fin de carrera

Un estudio comparativo entre los sistemas gestores RDF

Autor: Jorge Hernández Matute

Director: Harith Al-Jumaily

A Sara y a mis padres

AGRADECIMIENTOS

En primer lugar, agradecer a mi familia por todo el apoyo que me han dado a lo largo de estos años y por siempre estar ahí cuando les he necesitado. Y sobre todo a mi abuelo Pepe que ya no está con nosotros, pero que se la ilusión que le haría este momento.

A mi novia Sara, a la cual he conocido durante estos años universitarios y que tanto apoyo y ayuda me ha dado.

A todos los amigos que he hecho durante estos años universitarios y que tan buenos momentos me han hecho pasar. Y sobre todo a mi gran amigo Oscar.

Y por último, no podía olvidarme de Harith, mi director de proyecto, por ayudarme a hacer esto posible.

CONTENIDO

Índice de ilustraciones

Índice de tablas

1. INTRODUCCIÓN

En este primer apartado del proyecto se realiza una breve descripción sobre el mismo, indicando cual es el problema tratado en este proyecto y los objetivos a los que se quieren llegar a su conclusión.

Al final de este apartado se mostrara la estructura de la memoria del proyecto, indicando de los puntos de que consta y una breve introducción a cada uno de ellos.

1.1Presentación del problema

Parece que ya ha pasado mucho tiempo y que de la Web que conocimos a principios de los 90 ya poco queda de ella, solo hay que ver cómo han cambiado las páginas de entonces con las de ahora, ya sea por su aspecto físico, por cómo están estructuradas, por la cantidad de información de la que disponen, por su elevado número…

Pero hay un punto en el que la Web actual no dista mucho de la Web del comienzo, este punto es la “naturaleza de la información”. Ya que la mayoría de la información que nos encontramos actualmente en una página Web este representada en forma de texto. Una persona que lee esta información en forma de texto en una Web puede comprenderla fácilmente, pero por lo contrario es muy complicado crear procedimientos automáticos que obtengan su información semántica (relaciones que se establecen entre significados).

Esto se ve mejor con el siguiente ejemplo, actualmente existen muchas Web dedicadas a la comparación de vuelos de manera que podamos encontrar que oferta se aproxima más a nuestras necesidades. Estas Webs básicamente lo que hacen es redirigir a las consultas que reciben en ellas a la de las compañías aéreas intentando obtener la información semántica de cada portal de las compañías. Esto implica que se tiene que hace un mapeo de cada uno de los portales de las distintas compañías lo cual es algo complejo, que además muchas de las veces no nos darán la mejor solución y además el

análisis de cada uno de estos portales será muy sensible a las modificaciones que en él se produzcan.

La solución para este problema sería que todas las compañías de vuelo hablaran el mismo idioma, es decir, que para cada vuelo ofertado tuviéramos la misma información, que existiera un mismo formato para la información de un vuelo como por ejemplo: origen, destino, fecha de salida, fecha de llegada y precio. De esta manera no haría falta hacer un mapeo de cada compañía ni un programa que analizara la información para una sola, ya que ahora al tener todas la misma información bastaría que un solo programa que obtuviera estos datos y se los mostrara al usuario. Con esto además se solucionarían problemas como los cambios en una Web afectaran, ya que el formato seguirá siendo el mismo ahora estándar, o añadir nuevas compañías al sistema de comparación de vuelos. Pues esto es justo lo que pretende la Web semántica.

RDF es una de las tecnologías esenciales de la Web semántica con la que se quiere convertir las declaraciones de recursos de una página Web en expresiones del tipo sujeto-predicado-objeto (denominadas tripletas como se verá más adelante) y así definir metadatos[5] en la Web. De esta manera se quiere conseguir que los documentos Web tengan significado y los ordenadores sean capaces de interpretar la información contenida en ellos. RDF fue propuesto por la World Wide Web Consortium (W3C)[1].

Las tripletas RDF normalmente son serializadas mediante archivos XML[2] (Extensible Markup Language).Pero este método de almacenamiento presenta inconvenientes cuando se trabaja con volúmenes grandes de datos. Por este motivo desde hace unos años existen una serie de productos que permiten almacenar tripletas RDF en bases de datos. En la actualidad existen en el mercado tanto bases de datos relacionales como otras herramientas que utilizan modos de almacenamiento específicos para las tripletas RDF, estas herramientas son los denominados gestores RDF. Los gestores RDF son herramientas que nos van a permitir almacenar y interactuar con tripletas RDF.

En este proyecto fin de carrera se van a analizar distintos gestores RDF que en la actualidad hay en el mercado. Se analizaran distintos aspectos de cada uno de estos para realizar una comparación entre ellos y ver qué ventajas y desventajas tiene el uso de cada uno frente a los demás. Los aspectos a analizar serán el modelo de datos, la arquitectura de los gestores, la seguridad, el rendimiento durante inserciones de tripletas

RDF y el rendimiento de los gestores durante la realización de consultas realizadas sobre ellos.

Los gestores RDF que serán estudiados en este proyecto son:

- Oracle: básicamente es una herramienta cliente/servidor para la gestión de Bases de Datos. Dentro de Oracle están los paquetes SDO_RDF que van a ser los encargados de permitir trabajar con las tripletas RDF dentro de Oracle y son los que convierten a Oracle en un gestor RDF.
- Kowari: es un tipo de base de datos Java, optimizada para el almacenamiento y la gestión de los metadatos.
- Mulgara: es otra base de datos Java, creada por algunos de los que desarrollaron Kowari en 2006, que permite el almacenamiento y la gestión de metadatos.
- Sesame: es un Framework[3] Java (Estructura de soporte, marco de trabajo) en la cual se puede almacenar y realizar consultas sobre datos RDF.

1.2 Objetivos

Una vez que se ha hablado sobre el porqué del proyecto se exponen los objetivos que se han fijado para su realización:

- Estudio del ámbito de la tecnología en la que se va desarrollar el proyecto. Esta tecnología básicamente es RDF que es un lenguaje de descripción de recursos para la Web y los sistemas gestores que van a permitir trabajar con las tripletas RDF.
- Definir claramente cada uno de los puntos a analizar de los gestores RDF, para posteriormente analizar ese punto de forma detallada en cada gestor.
- Realizar una comparación entre los gestores llevados a estudio de cada uno de los puntos analizados, indicando que diferencias hay entre ellos, las desventajas

y ventajas que pueden tener cada uno de los gestores sobre los demás en un determinado punto.

- Uso y manejo de los gestores RDF llevados a estudio para la inserción masiva y manipulación de tripletas RDF en ellos. En este punto se analizaran como se produce la inserción de datos en cada uno de los gestores, que problemas pueden aparecer durante este proceso y como después de la inserción se pude trabajar con los datos insertado en ellos.

- Realizar pruebas de rendimiento, tanto en inserciones como en consultas en los gestores RDF, comprando los resultados obtenidos en cada uno de ellos para los mismos datos insertados y las mismas consultas realizadas.

1.3Estructura del documento

En este apartado se va a indicar la estructura capitular de este proyecto y una descripción de cada uno de los puntos de los que consta:

- **Capitulo 2.** Recuperación de Información y la tecnología RDF: expone de forma breve el ámbito de la tecnología RDF sobre la que se trabaja en este proyecto :

    - Introducción sobre la recuperación de la información, estado de la Web actual y que pretende conseguir la web semántica en el ámbito de la recuperación de la información.

    - Definición de las bases de datos RDF y representación de los recursos Web mediante tripletas RDF.

- Capitulo **3. Sistemas Gestores RDF**: en este apartado se analizan las características o aspectos más teóricos de los gestores RDF, comparando los cuatro gestores RDF llevados a estudio en este proyecto en cada uno de estos puntos:

    - *Arquitectura*: en este apartado se va analizar el esqueleto de cada uno de los gestores RDF. Se va a estudiar las capas que forman cada uno de ellos y ver que aportan y para qué sirven cada una de ellas.

- *Modelo de datos:* en este apartado se analizara con que estructuras se almacenan los datos, el uso o no de índices y las ventajas que se obtienen por la utilización de índices a la hora de realizar búsquedas …
- *Seguridad:* en este apartado se analizaran los medios e infraestructuras para la seguridad que nos proporciona cada uno de los gestores RDF que se van a analizar.
- *Lenguajes de consulta*: son los lenguajes que van a permitir realizar consultas sobre los gestores RDF y recuperación de la información que se solicite de estos. Sera el método de comunicación con el gestor.

- **Capitulo 4. Resultados de experimentos**: esta es la parte experimental del proyecto en la que se crean repositorios para almacenar tripletas RDF en cada uno de los sistemas gestores para posteriormente realizar consultas sobre ellos y poder medir su rendimiento tanto en las consultas como la inserción de las tripletas.

- **Capitulo 5. Conclusiones y líneas futuras**: conclusiones del resultado de la comparación de cada uno de los puntos analizados de los gestores RDF, conclusión general del estudio y exposición personal del autor sobre la evolución del trabajo. También se comentaran posibles líneas de trabajo futuro que se podrán realizar teniendo como base este proyecto.

- **Anexos:**
  - *Sintaxis de consultas:* significado y sintaxis en cada uno de los lenguajes de consulta que se utilizan en el proyecto, de las consultas utilizadas en el capítulo 4 del proyecto.
  - *Herramientas para la toma de datos de rendimiento:* herramientas utilizadas en el capítulo 4 para la recopilación de datos acerca del rendimiento de los gestores RDF durante las operaciones de inserción de tripletas RDF y consultas.
  - *SQL LOADER:* es una importante utilidad que permite cargar datos externos (en ficheros de texto por general) en tablas de una base de datos en Oracle.

2. RECUPERACIÓN DE INFORMACIÓN Y LA TECNOLOGÍA RDF

Es importante realizar un estudio previo al ámbito tecnológico en el que se va desarrollar el proyecto para entender bien cuál es el estado actual y que se pretende conseguir con la realización de este proyecto.

El ámbito de este proyecto es la recuperación de la información, en este apartado se verá en qué consiste, que métodos se usan actualmente para la recuperación de la información, que se pretende conseguir con la Web-semántica, y más concretamente con RDF y por último la necesidad de los gestores RDF.

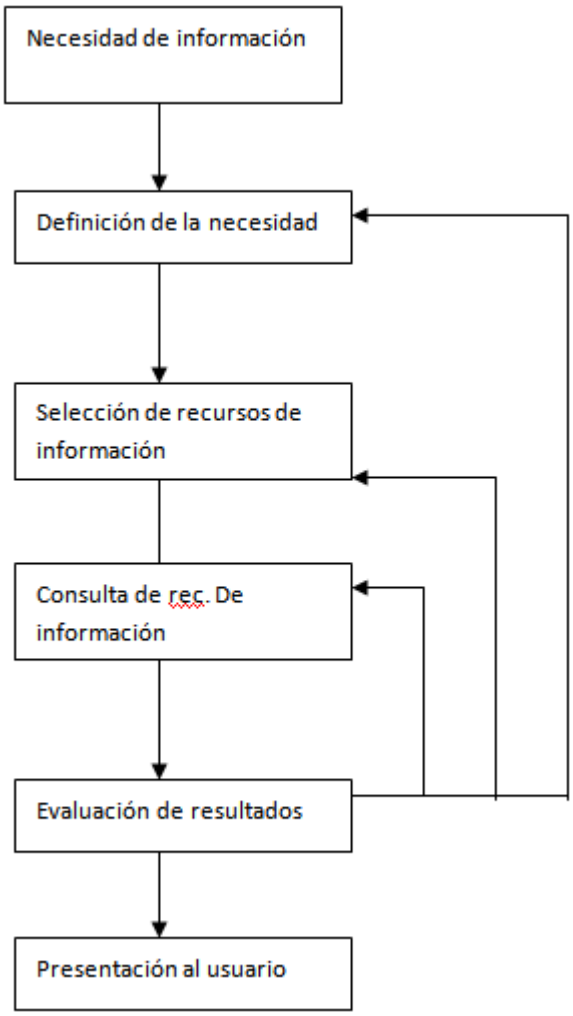
2.1 Recuperación de la información

Cuando un usuario se plantea la necesidad de obtener nueva información sobre un asunto o materia de su interés, está manifestando una carencia. Peter Ingwersen (profesor de la universidad de Copenhague en sistemas de recuperación de la información) deduce la existencia de un problema personal de espacio, por la diferencia del estado actual del conocimiento del usuario, y del estado que sería necesario para solucionar algún tipo de necesidad planteada. La respuesta a este tipo de situaciones es un conjunto de actividades que desarrolla el individuo para salir del estado anómalo, o para solucionar su problema de espacio, actividades que están íntimamente relacionadas con la adquisición de nueva información, y con el proceso comunicativo pertinente.

Diremos que la recuperación de la información es el conjunto de tareas mediante las cuales el usuario localiza y accede a los recursos de información que son pertinentes para la resolución de un problema planteado

En el siguiente esquema (Ilustración 2-1) se muestra el proceso genérico de recuperación de información:

Ilustración 2-1: Recuperación información



En principio, la recuperación de información engloba las acciones encaminadas a identificar, seleccionar y acceder a los recursos de información útiles al usuario.

Un punto importante en este apartado es distinguir entre recuperación de datos (RD) y recuperación de información (RI), para ello vemos los siguientes puntos:

- Según la forma de responder a la pregunta: En RD se utilizan preguntas altamente formalizadas, cuya respuesta es directamente la información deseada. Mientras que en RI las preguntas resultan difíciles de trasladar a un lenguaje formalizado y la repuesta es un conjunto de documentos que pueden contener o no la respuesta que nosotros buscamos (grado de incertidumbre), esto es lo que pasa cuando buscamos información en un buscador Web como puede ser Google.

- Según la relación entre el requerimiento al sistema y la satisfacción del usuario: En RD la relación es determinista entre la pregunta y la satisfacción. Mientras que en RI es probabilista debido al gran grado de incertidumbre que hay.
- Según el criterio de éxito: En el RD el criterio a emplear es la corrección y la exactitud, mientras que en RI el único criterio de valor es la satisfacción del usuario, basada en un criterio personal de utilidad.
- Según la rapidez de respuesta: En RD depende del soporte físico y de la perfección del algoritmo de búsqueda y de los índices. En RI depende de las decisiones y acciones del usuario durante el proceso de interrogación, es algo que no depende tanto del buscador sino de cómo decide buscar el usuario, de las decisiones que el toma durante este proceso, estas decisiones harán que la información deseada puede ser encontrada o no.

Como se ha visto en el punto anterior en la recuperación de la información se plantean problemas como la incertidumbre a la hora de encontrar la información deseada, problemas de ambigüedad...

Para solventar este problema a la hora de la recuperación de la información surgen los sistemas expertos que son aplicaciones capaces de realizar las tareas de un experto humano en un área restringida. Con los sistemas expertos se busca una mejor calidad y rapidez en las respuestas dando así lugar a una mejora de la productividad del experto.

Estos sistemas se componen de una base de datos, una base de reglas y un motor de inferencia. La base de datos almacena el conjunto de datos o documentos sobre los que se desea realizar una serie de acciones. La base de reglas contiene un conjunto de reglas lógicas que el sistema debe utilizar para desarrollar razonamientos, así como las normas que permiten combinar las reglas. Por último el motor de inferencia es el encargado de ejecutar las órdenes del usuario, utilizando como criterios las reglas, y como marial de partida el contenido de la base de datos, hasta alcanzar una conclusión simulando el razonamiento de un experto humano. A continuación vamos a ver las ventajas y las limitaciones de estos sistemas:

Ventajas:

- *Permanencia*: A diferencia de un experto humano un SE (sistema experto) no envejece, y por tanto no sufre pérdida de facultades con el paso del tiempo.
- *Duplicación*: Una vez programado un SE lo podemos duplicar infinidad de veces.
- *Rapidez*: Un SE puede obtener información de una base de datos y realizar cálculos numéricos mucho más rápido que cualquier ser humano.
- *Bajo costo*: A pesar de que el costo inicial pueda ser elevado, gracias a la capacidad de duplicación el coste finalmente es bajo.
- *Entornos peligrosos*: Un SE puede trabajar en entornos peligrosos o dañinos para el ser humano.
- *Fiabilidad*: Los SE no se ven afectados por condiciones externas, un humano sí (cansancio, presión, etc.).
- *Consolidar varios conocimientos*

Limitaciones:

- *Sentido común*: Para un SE no hay nada obvio. Por ejemplo, un sistema experto sobre medicina podría admitir que un hombre lleva 40 meses embarazado, a no ser que se especifique que esto no es posible.
- *Lenguaje natural*: Con un experto humano podemos mantener una conversación informal mientras que con un SE no podemos.
- *Capacidad de aprendizaje*: Cualquier persona aprende con relativa facilidad de sus errores y de errores ajenos, que un SE haga esto es muy complicado.
- *Perspectiva global*: Un experto humano es capaz de distinguir cuales son las cuestiones relevantes de un problema y separarlas de cuestiones secundarias.
- *Capacidad sensorial*: Un SE carece de sentidos.
- *Flexibilidad*: Un humano es sumamente flexible a la hora de aceptar datos para la resolución de un problema.
- *Conocimiento no estructurado*: Un SE no es capaz de manejar conocimiento poco estructurado.

Lo que diferencia a estos sistemas de un sistema tradicional de recuperación de la información es que estos últimos solo son capaces de recuperar aquello que existe

explícitamente, mientras que un sistema experto debe ser capaz de generar información no explicita razonando con los elementos que le dan.

Una vez vista este visión general sobre la recuperación de la información, según incumbe a este proyecto se dirá que es la ciencia de la búsqueda de información en documentos, búsqueda de los mismos documentos, la búsqueda de metadatos que describan documentos, o, también la búsqueda en bases de datos, ya sea a través de Internet, intranet, para textos, imágenes, sonidos o datos de otras características, de manera pertinente y relevante.

En la actualidad el mayor medio donde se encuentra información es la Web y los buscadores, tales como Google, Lycos y Copernic son algunas de las aplicaciones más populares de la recuperación de la información. Básicamente hay que construir un Vocabulario, que es una lista de términos en lenguaje natural, un algoritmo que incluya las reglas lógicas de la búsqueda {tabla de verdad} y una valoración de los resultados o cantidad de información lograda o posible.

2.2 Web actual

Como ya se ha dicho antes hoy en día dispones de una gran cantidad de posibilidades y de información gracias a la aparición de Internet. En Internet cualquier persona puede colgar documentos y ponerlos a disposición de cualquier usuario. Por lo tanto es necesario sistemas que nos permiten la búsqueda de la información que necesitamos en un momento dado dentro de toda esta cantidad de información.

La Web actual tiene las siguientes características:

- Funciona como una biblioteca digital hipermedia.
- Es una biblioteca de documentos (páginas Web) interconectados por hiperenlaces.
- Puede funcionar usando una base de datos, y servir como una plataforma de aplicaciones.
- Es una plataforma para elementos multimedia.
- Cada uno de los documentos tiene un modo de identificarse único (URLs[8]).

El problema que se plantea es que en un principio los lenguajes utilizados para la confección de páginas Web estaban pensados para los humanos, para su lectura y comprensión. Pero claro aunque nosotros entendamos esta información, nuestro ordenador no, y por lo tanto no es capaz de procesar esta información automáticamente. Si yo por ejemplo accedo a una tienda on-line para comprar algún producto mi ordenador no es capaz de procesar esta información automáticamente y ayudarme de esta manera en mi objetivo. Lo mismo ocurre si se tiene que planificar un viaje, para poder hacerlo se tiene que navegar por infinidad de páginas, Web ya sean comparadores de vuelos, buscadores…, buscando las mejores ofertas de vuelos, buscando alojamiento… Con un buscador actual al introducir lo que se quiere no se obtienen unos resultados muy exactos sobre lo que se está buscando y es labor de las personas buscar dentro de las diferentes páginas que nos ha proporcionado el buscador. Para verlo más claro podemos se tiene el siguiente ejemplo en él se busca un vuelo y alojamiento para realizar un viaje:

Si se buscan vuelos para Praga en un buscador actual se tienen los siguientes resultados:

Ilustración 2-2: Búsqueda vuelos a Praga en un buscador Web

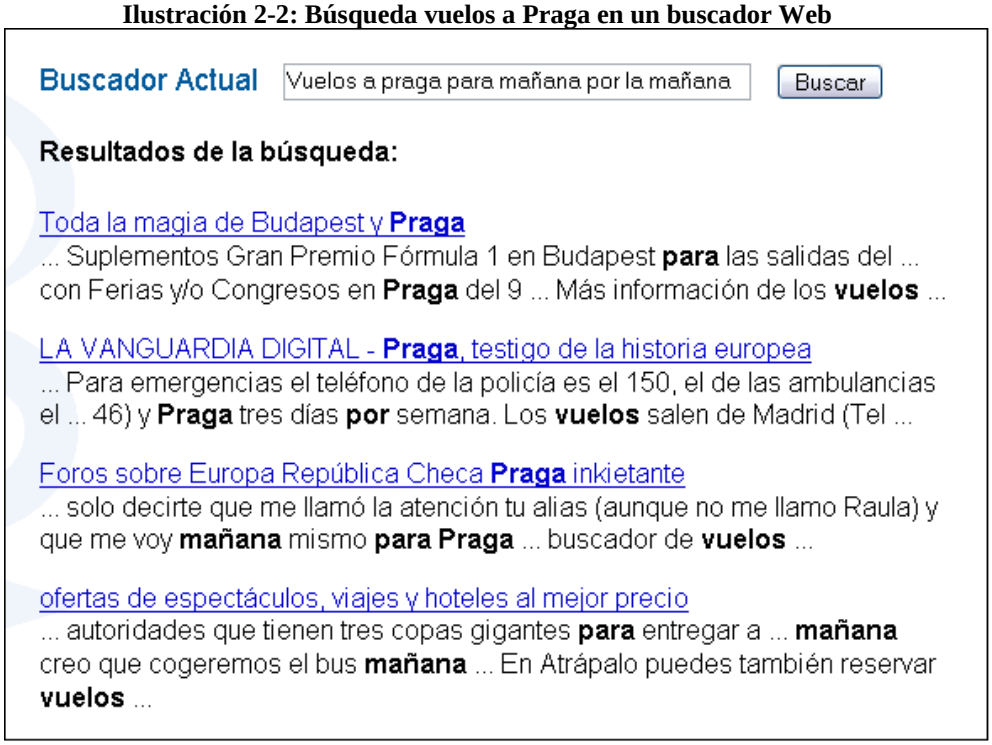
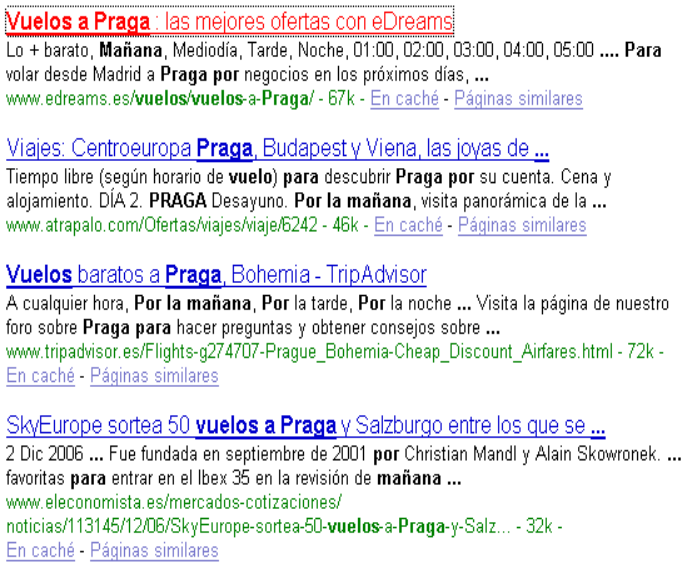
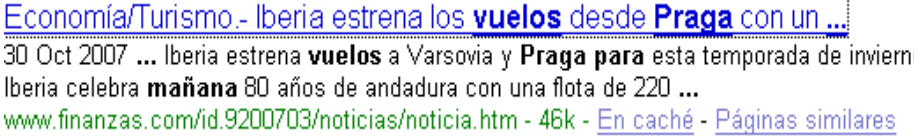


Ilustración 2-3: Búsqueda vuelos a Praga en Google



Como se puede ver tanto en la Ilustración 2-2 y en la Ilustración 2-3, lo que se consigue con esta búsqueda no es exactamente un vuelo para mañana a Praga sino una serie de enlaces a páginas Web donde quizás se puedan buscar vuelos para Praga. Pero para esto se tendrá que mirar entre las diferente opciones que muestra el buscador para poder comparar precios … Además se añade la problemática que al no ser la información que se introduce en el buscador tratada semántica mente, se encuentran resultados que no se corresponden con los esperados, como es este caso:

Ilustración 2-4: Enlace obtenido en la búsqueda de vuelos aPraga



El enlace de la Ilustración 2-4 conduce a una página Web donde se hace referencia a una noticia de Europa Press de economía y turismo, que ni mucho menos tiene que ver con lo que se está buscando.

En la actualidad los sistemas de recuperación de información más extendidos son los buscadores Web, los cuales tienen un uso cada más y más entendido para cualquier tipo de información que se necesita en un momento dado (ya sea un restaurante donde poder cenar esta noche, información acerca de la Web, el tiempo que va hacer hoy…).

Los buscadores Web son páginas Web en la que se ofrece consultar una base de datos en la cual se relacionan direcciones de páginas Web con su contenido. Su uso facilita enormemente la obtención de un listado de páginas Web que contienen información sobre el tema que nos interesa. Todos los buscadores Web tienen algo en común, que es que todos nos devuelven un conjunto de páginas Web sobre la consulta que hemos realizado.

Hoy en día todos conocemos un gran número de buscadores como pueden ser Microsoft, Live Search, Lycos…

También existen otro tipo de buscadores que son los metabuscadores. La diferencia de este tipo de buscadores con los citados anteriormente es que estos carecen de sus propias bases de datos, estos utilizan las de otros buscadores para realizar sus búsquedas, mostrando al usuario una combinación de las páginas Web mejores que ha devuelto cada buscador. Su principal ventaja es que el ámbito de búsqueda es mucho mayor, además se puede conseguir una mayor variedad de resultados, ya que los resultados que para un buscador tienen mayor notoriedad no tienen que ser los mismo que para otro, debido a que cada buscador tiene una estrategia distinta a la hora de mostrar los resultados de una búsqueda. Pero no todo son ventajas ya que estos son más lentos respecto a los buscadores Web y no es posible realizar búsquedas muy específicas ya que al tener cada buscador sus propios operadores y sintaxis en un metabuscador no se pueden aplicar.

Ejemplos de metabuscadores son Metacrawler y Copernic Agent Basisc. Este último a diferencia de los demás no es una página Web sino software.

2.3 Web-semántica

Pero esta Web actual esta falta de significado y semántica, como se ha visto en los ejemplos anteriores muchas es veces es difícil encontrar la información que se desea, ya que el manejo de esta información es complicado. De este déficit en la Web actual surge la Web semántica.

La Web Semántica es una Web extendida, dotada de mayor significado en la que cualquier usuario en Internet podrá encontrar respuestas a sus preguntas de forma más rápida y sencilla gracias a una información mejor definida. Al dotar a la Web de más significado y, por lo tanto, de más semántica, se pueden obtener soluciones a problemas habituales en la búsqueda de información gracias a la utilización de una infraestructura común, mediante la cual, es posible compartir, procesar y transferir información de forma sencilla. Esta Web extendida y basada en el significado, se apoya en lenguajes universales que resuelven los problemas ocasionados por una Web carente de semántica en la que, en ocasiones, el acceso a la información se convierte en una tarea difícil y frustrante.

La Web ha cambiado profundamente la forma en la que las personas se comunican, hacen negocios y realizan su trabajo. La comunicación prácticamente con todo el mundo en cualquier momento y a bajo coste es posible hoy en día. Se pueden realizar transacciones económicas a través de Internet. Se tiene acceso a millones de recursos, independientemente de la situación geográfica e idioma. Todos estos factores han contribuido al éxito de la Web. Sin embargo, al mismo tiempo, estos factores que han propiciado el éxito de la Web, también han originado sus principales problemas: sobrecarga de información y heterogeneidad de fuentes de información con el consiguiente problema de interoperabilidad.

La Web Semántica ayuda a resolver estos dos importantes problemas permitiendo a los usuarios delegar tareas en software. Gracias a la semántica en la Web, el software es capaz de procesar su contenido, razonar con este, combinarlo y realizar deducciones lógicas para resolver problemas cotidianos automáticamente.

Supóngase que la Web tiene la capacidad de construir una base de conocimiento sobre las preferencias de los usuarios y que, a través de una combinación entre su capacidad de conocimiento y la información disponible en Internet, sea capaz de atender de forma exacta las demandas de información por parte de los usuarios en relación, por ejemplo, a reserva de hoteles, vuelos, médicos, libros, etc.

Si esto sucediera el usuario al realizar una búsqueda se obtendría un resultado exacto de lo que está buscando, pero esto no es lo que sucede en la actualidad, como se pudo ver antes en el ejemplo en el que se buscaban vuelos a Praga.

La forma en la que se procesará esta información no sólo será en términos de entrada y salida de parámetros sino en términos de su SEMÁNTICA. La Web Semántica como infraestructura basada en metadatos aporta un camino para razonar en la Web, extendiendo así sus capacidades.

No se trata de una inteligencia artificial mágica que permita a las máquinas entender las palabras de los usuarios, es sólo la habilidad de una máquina para resolver problemas bien definidos, a través de operaciones bien definidas que se llevarán a cabo sobre datos existentes bien definidos.

Para obtener esa adecuada definición de los datos, la Web Semántica utiliza esencialmente RDF, SPARQL, y OWL (son estándares definidos por la W3C), mecanismos que ayudan a convertir la Web en una infraestructura global en la que es posible compartir, y reutilizar datos y documentos entre diferentes tipos de usuarios.

El World Wide Web Consortium, abreviado W3C, es un consorcio internacional que produce estándares para la World Wide Web (o la "Web"). Está dirigida por Tim Berners-Lee, el creador original de URL (Uniform Resource Locator, Localizador Uniforme de Recursos), HTTP (HyperText Transfer Protocol, Protocolo de Transferencia de HiperTexto) y HTML[6] (Lenguaje de Marcado de HiperTexto) que son las principales tecnologías sobre las que se basa la Web [22].

Este consorcio fue creado por Tim Berners-Lee el 1 de octubre de 1994y en octubre de 2007 ya contaba con 435 miembros. Este consorcio dispone de oficinas en España desde 2003 en el parque científico tecnológico de Gijón.

A día de hoy el W3C ha publicado más de ciento diez estándares desde que se fundó, llamados recomendaciones del W3C. En palabras de Tim Berners-Lee el objetivo del W3C es: "*Guiar la Web hacia su máximo potencial a través del desarrollo de protocolos y pautas que aseguren el crecimiento futuro de la Web*". A continuación se muestra algunos de los estándares que ha creado:

- SPARQL Query Language for RDF: es lenguaje de consulta sobre RDF, que permite hacer búsquedas sobre los recursos de la Web Semántica utilizando distintas fuentes datos.

- Web Services Addressing 1.0 – Metadata: proporciona mecanismos independientes del transporte para direccionar servicios web y mensajes. Esta especificación permite a los sistemas de mensajería habilitar la transmisión de mensajes a través de redes que incluyen nodos de procesamiento, tales como administradores de extremos, cortafuegos y pasarelas, de un modo independiente del transporte.
- SOAP Message Transmission Optimization Mechanism: es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.
- OWL Web Ontology Language: es un mecanismo para desarrollar temas o vocabularios específicos en los que asociar esos recursos. Lo que hace OWL es proporcionar un lenguaje para definir ontologías[22] estructuradas que pueden ser utilizadas a través de diferentes sistemas. Las ontologías, que se encargan de definir los términos utilizados para describir y representar un área de conocimiento, son utilizadas por los usuarios, las bases de datos y las aplicaciones que necesitan compartir información específica, es decir, en un campo determinado como puede ser el de las finanzas, medicina, deporte, etc. Las ontologías incluyen definiciones de conceptos básicos en un campo determinado y la relación entre ellos.
- HTML 4.0 Specification: esta especificación define el Lenguaje de Formato de Documentos para Hipertexto (*HyperText Markup Language, HTML*), el lenguaje de publicación de la World Wide Web
- Resource Description Framework (RDF): proporciona información descriptiva simple sobre los recursos que se encuentran en la Web y que se utiliza, por ejemplo, en catálogos de libros, directorios, colecciones personales de música, fotos, eventos, etc.
- RDFa in XHTML: Syntax and Processing: es otra tecnología que ofrece la Web Semántica para enriquecer los contenidos de la Web tradicional es RDFa. Mediante RDFa se pueden representar los datos estructurados visibles en las páginas Web (eventos en calendarios, información de contacto personal, información sobre derechos de autor, etc.), a través de unas anotaciones semánticas incluídas en el código e invisibles para el usuario, lo que permitirá a las aplicaciones interpretar esta información y utilizarla de forma eficaz. Por

ejemplo, una aplicación de calendario podría importar directamente los eventos que encuentra al navegar por cierta página Web, o se podrían especificar los datos del autor de cualquier foto publicada, así como la licencia de cualquier documento que se encuentre. Para extraer el RDF se podría utilizar GRDDL, una técnica estándar para extraer la información expresada en RDF desde documentos XML, y en particular, de las páginas XHTML.

Pues como se ha dicho a partir de estos estándares se construye la Web semántica. ¿Pero qué ejemplos existen de Web Semántica? Tres de los ejemplos más conocidos de aplicación de Web Semántica son RSS, FOAF y buscadores semánticos.

- RSS es un vocabulario RDF basado en XML que permite la catalogación de información (noticias y eventos) de tal manera que sea posible encontrar información precisa adaptada a las preferencias de los usuarios. Los archivos RSS contienen metadatos sobre fuentes de información especificadas por los usuarios cuya función principal es avisar a los usuarios de que los recursos que ellos han seleccionado para formar parte de esa RSS han cambiado sin necesidad de comprobar directamente la página, es decir, notifican de forma automática cualquier cambio que se realice en esos recursos de interés seleccionados. Un ejemplo de la aplicación de RSS se puede encontrar en las Noticias de la Oficina Española del W3C como canal RSS.
- FOAF es un proyecto de Web Semántica, que permite crear páginas Web para describir personas, vínculos entre ellos, y cosas que hacen y crean. Se trata de un vocabulario RDF, que permite tener disponible información personal de forma sencilla y simplificada para que pueda ser procesada, compartida y reutilizada. Dentro de FOAF se puede destacar FOAF-a-Matic, que se trata de una aplicación Javascript que permite crear una descripción FOAF de uno mismo. Con esta descripción, los datos personales serán compartidos en la Web pasando a formar parte de un motor de búsqueda donde será posible descubrir información a cerca de una persona en concreto y de las comunidades de las que es miembro de una forma sencilla y rápida.

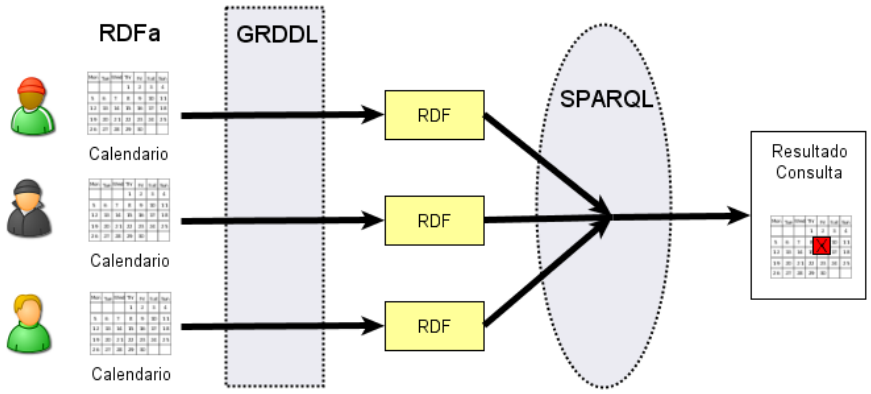
Para verlo de una forma más practica vamos a ver como utilizando esta tecnología se podría establecer una reunión entre tres personas, que tienen publicados en sus sitios Webs respectivos los calendarios con sus citas y eventos. Estos datos están expuestos en paginas XHTML de forma gráfica, pero además se incluye información en RDFa:

Una herramienta permite extraer, mediante GRDDL, los datos de sus calendarios en un formato homogéneo y fácil de tratar (RDF), para poder procesarlo posteriormente.

Se realiza una consulta sobre la disponibilidad de las personas para un cierto día a una hora concreta. Los datos consultados están en formato RDF y la consulta se podría realizar mediante SPARQL (Lenguaje de consultas para RDF).

La herramienta procesa y analiza el resultado obtenido, concluyendo si las personas están disponibles en el instante que se había elegido previamente.

Ilustración 2-5: ejemplo aplicación web semántica



Los buscadores semánticos son un ejemplo más de aplicaciones basadas en Web Semántica. El objetivo es satisfacer las expectativas de búsqueda de usuarios que requieren respuestas precisas.

2.4. Buscadores semánticos.

Los buscadores semánticos son aquellos que hacen una búsqueda atendiendo al significado del grupo de palabras que se introducen el buscador y no basándose en las actuales etiquetas, como puede hacer un buscador como Yahoo. Digamos para entenderlo mejor que son buscadores inteligentes. En la actualidad su uso no está muy extendido, pero ya existen algunos con resultados destacables y que sin duda son el futuro en la búsqueda de información.

Pero el problema que existe en la actualidad es que casi todos los buscadores, que se presentan como semánticos no son más que buscadores ontológicos, es decir, buscan ficheros en RDF, micro formatos y algunos en OWL, pero son herramientas inadecuadas para usuarios finales. Estos buscadores son útiles para expertos y probablemente para, una vez procesada toda la información, poder ofrecer servicios nuevos. Hay muy pocos buscadores semánticos de propósito general que ya funcionan [21]. A continuación se muestran algunos ejemplos de los buscadores semánticos que hay en la actualidad:

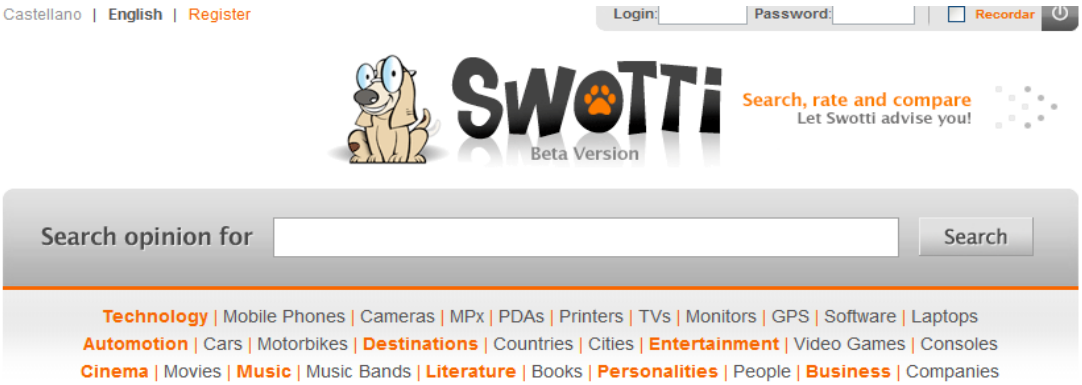
- Swotti: rastrea opiniones sobre productos, rastrea toda la red intentando considerar sola la información que recoja la opinión de los usuarios. Para ellos se apoya en la Web semántica.

  Swotti trabaja apoyándose en la Web semántica ya que se apoya en una tecnología capaz de identificar los adjetivos y verbos que definen aquello que estamos buscando, y que por tanto permiten deducir si el comentario es positivo o negativo. Cuando hacemos una búsqueda en Swotti obtenemos no sólo resultados, sino sobre todo una valoración cualitativa. Y eso puede aplicarse a personas, marcas, productos, empresas, ciudades…

  El producto aún está en una fase muy beta y no siempre obtienes el número de resultados que quisieras, pero apunta maneras muy interesantes. Por ejemplo, si buscas “iPhone” parece ser que lo más valorado son aspectos como la velocidad, el diseño, la batería o la cámara, mientras que lo más negativo son el peso, el tamaño o el altavoz. Aunque sus creadores insisten en posicionarse como un buscador de productos, para acompañar decisiones de compra, a mi me

gustan estas herramientas para aplicarlas a conceptos, sectores, instituciones, políticos… aunque ciertamente muchas de estas cosas a menudo son sólo productos. Sea como fuere, Swotti está desarrollando sus algoritmos semánticos por familias temáticas, y de momento ha empezado a trabajar en familias como la electrónica de consumo, los videojuegos, la informática, el cine, la literatura, la música, el turismo, la automoción y cosas similares.

Ilustración 2-6: swotti



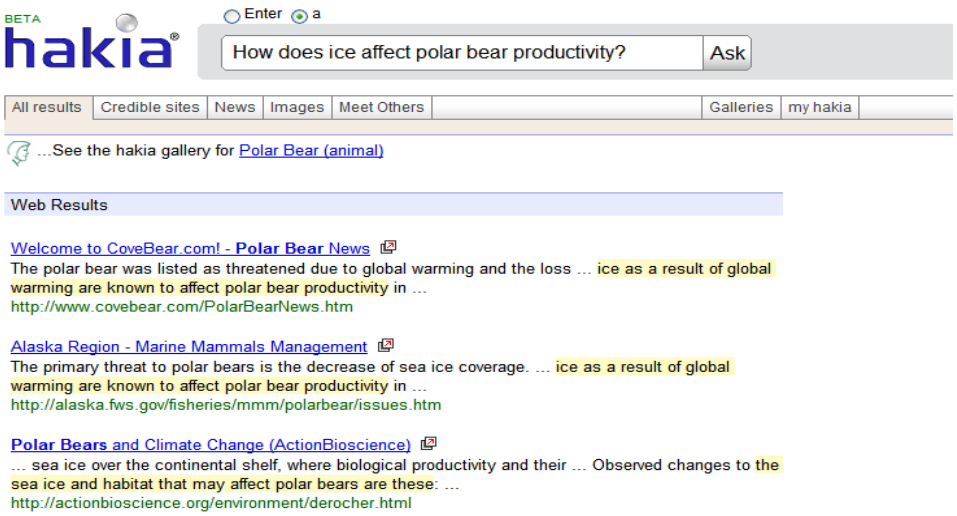
- Hakia: buscador que responde a preguntas escritas en ingles. Esta web esta concebida por un experto en inteligencia artificial y del procesamiento natural del lenguaje, el profesor Riza Berkan. que comenzó desarrollando varios buscadores semánticos verticales, tiene ahoras un Buscador Semántico de propósito general. La calidad de los resultados cumple simultáneamente tres criterios:

  - Fuentes dignas de confianza (verticales).
  - Información reciente.
  - Relevante para la consulta.

  El motor de búsqueda actualmente funcionan en modo beta, pero el analisis y desarrollo continúa avanzando. Aunque Hakia de momento solo soporta ingles, pero están trabajando para dar cobertura al portugues, español, italiano, polaco y turco. Según el Dr. Riza Berkan, fundador y director general de Hakia, están llegando al final de la primera fase de desarrollo. Hakia es

claramente un Buscador Semántico de propósito general, pero sus efectos, a fecha de hoy, no son aún espectaculares, realmente parece un buscador corriente.

Ilustración 2-7:Hakia



- SWOOGLE: no es un buscador destinado a un usuario común, este buscador está concebido para el rastreo de documentos de Web semánticos cuyos formatos pueden ser OWL, RDF y DAML.

  SWOOGLE no está orientado a proporcionar documentos HTML que son los documentos que requieren los usuarios finales, es un buscador de ontologías. Ha sido desarrollado de la Universidad de Maryland, Baltimore County (UMBC), en concreto por el grupo de investigación del departamento de Ciencias de Computación e Ingeniería Electrica (CSEE), conocido como UMBC eBiquity.

  Recoge más de 850K de documentos Web semánticos recolectados de la Web, bien buscando directamente en ficheros RDF y OWL o a través de páginas Web (HTML) que pueden contener documentos SW. Más de 10.000 ontologías disponibles en la Web (1.0, y 2.0), almacenadas (de momento, no en tripleta) sen una base de datos MySQL en forma de URIs[8], pero también permite buscar en los términos de cada vocabulario/esquema/ontología.

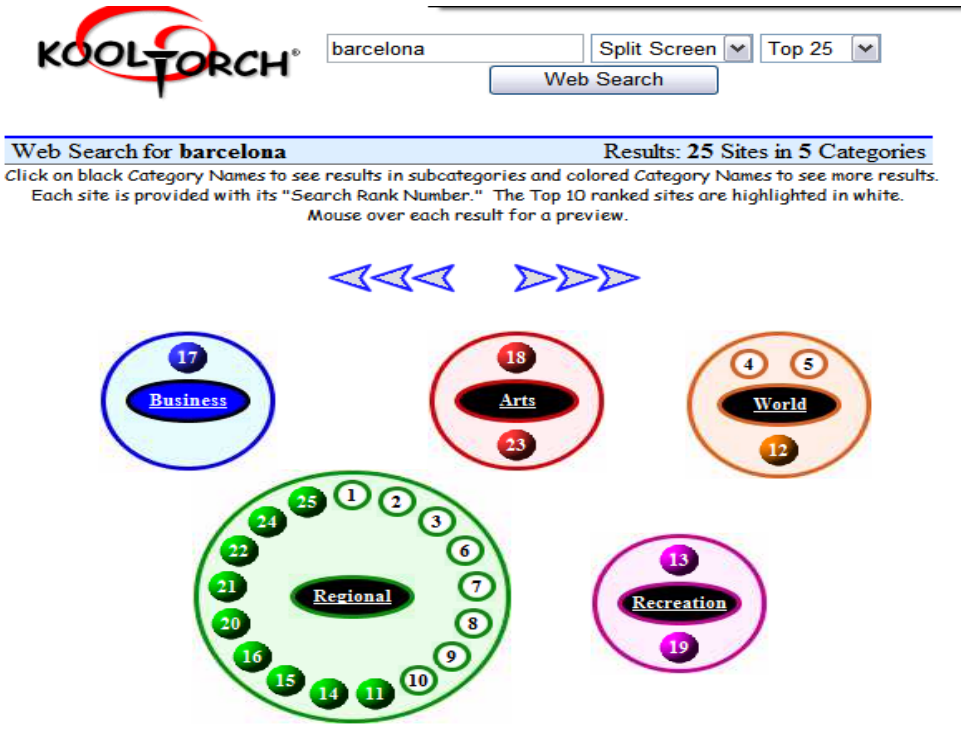
Ahora mismo, SWOOGLE es una herramienta utilísima para los desarrolladores de la Web semántica y para los agentes de software basado en estas tecnologías:

- Para estudiar la magnitud y el crecimiento de la Web semántica
- Para recopilar y buscar clases y propiedades (términos de la Web Semántica, SWTs) o las ontologías en que se conforman
- Para apoyar herramientas de carácter semántico.

Todavía este buscador no está dirigido al usuario final para encontrar recursos Web, sino que es más bien un parabuscador para buscar, clasificar e incluso validar documentos y vocabularios de la Web Semántica.

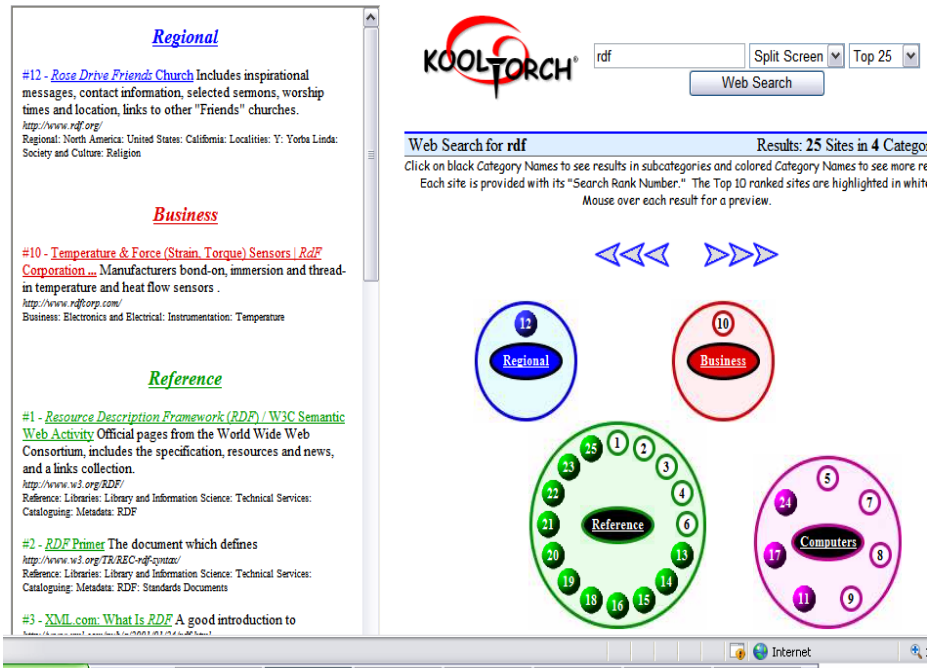
- Kooltorch: es uno de los buscadores semánticos más laureados y prometedores. Ofrece de una forma muy visual y estructurada sus resultados, tal y como se puede ver en la siguiente imagen:

Ilustración 2-8:Kooltorch



Los resultados están agrupados en categorías basados en rasgos comunes. Cada categoría aparece en el interior de un círculo, rodeada de círculos más pequeños que contienen rasgos comunes a esa categoría.

Ilustración 2-9:Kooltorch resultados búsqueda



También existen otros sitios Web como DBpedia donde se pueden realizar consultas sobre metadatos, marcados con un lenguaje de marcado como es RDF y realizando esta consultas en SPRAQL.

2.5 DBpedia

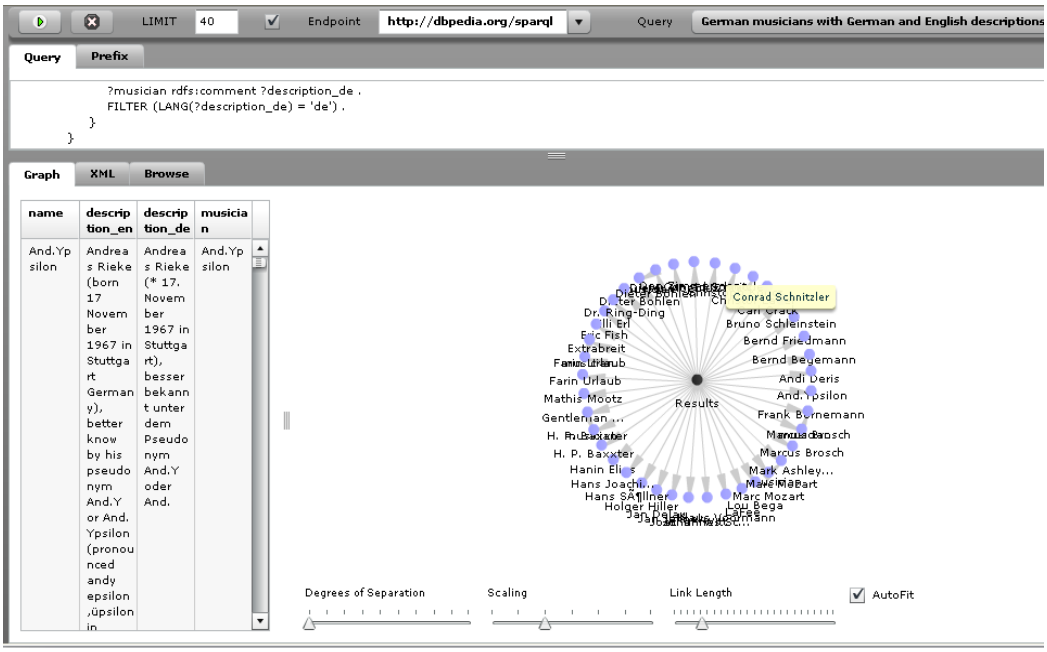
DBpedia es un proyecto para la extracción de datos de Wikipedia para proponer una versión Web semántica (DBpedia contiene una conversión automática a RDF de parte de la información existente en Wikipedia). Por ejemplo, una búsqueda en Wikipedia de series cómicas de televisión en Nueva York produce un resultado poco útil que muestra la relevancia de todos los aciertos encontrados. Sin embargo, con DBpedia, aparece un listado de todas las series de televisión de Nueva York como si de una consulta SQL[10] de base de datos. Este proyecto está realizado por la Universidad de Leipzig, Universidad Libre de Berlín y la compañía OpenLink Software [23].

La información se almacenan con el Resource Description Framework (se extrae como ya se ha dicho de la Wikipedia) y después se pueden realizar consultas a la base de datos a través de SPARQL. El motor de extracción de datos se realiza con PHP 5, un software libre publicado bajo el GNU General Public License. Su código fuente se distribuye: se alberga en sourceforge y disponible a través de Subversion.

Existe un *endpoint* (servicio web) público para hacer consultas SPARQL sobre DBpedia. A través de esta web se pueden realizar consultas a la base de datos y se encuentran ejemplos de lo que serian consultadas realizadas a los datos extraídos de la Wikipedia:

```
SELECT ?name ?description_en ?description_de ?musician WHERE
{?musician skos:subject
<http://dbpedia.org/resource/Category:German_musicians> . ?musician
foaf:name ?name .OPTIONAL { ?musician rdfs:comment ?description_en .
FILTER (LANG(?description_en) = 'en') .}OPTIONAL {?musician
rdfs:comment ?description_de .FILTER (LANG(?description_de) = 'de') .}}
```

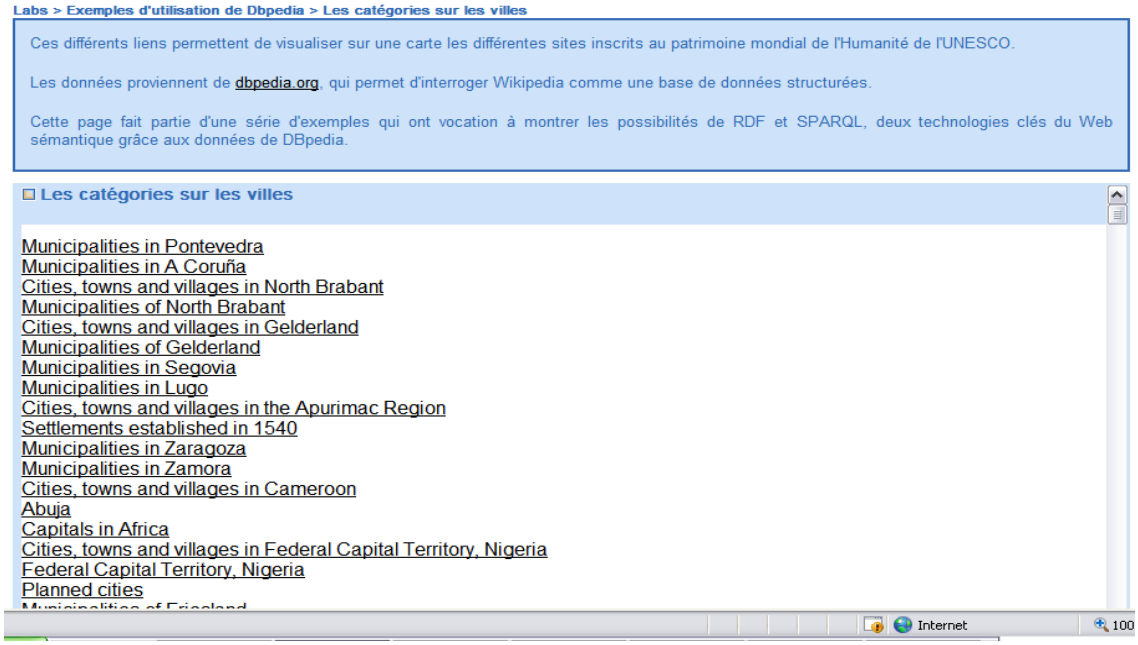
Ilustración 2-10: consulta en DBpedia



Otro ejemplo es una consulta en la que buscan los sitios inscritos dentro de patrimonios nacionales en la UNESCO:

http://www.lespetitescases.net/semweblabs/dbpedia/dbpedia-categorie-ville.php

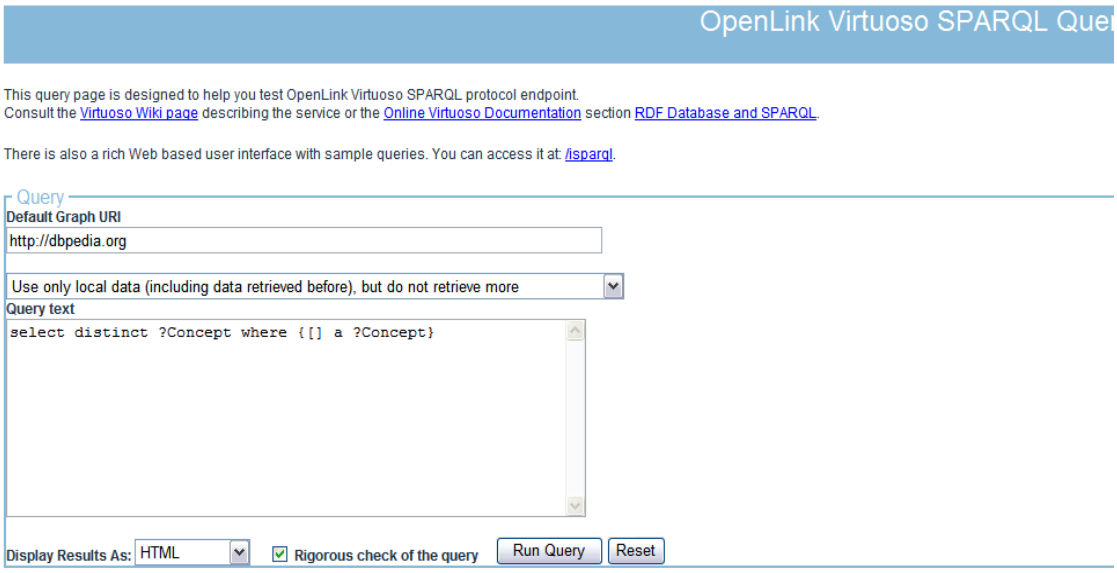
Ilustración 2-11: consulta UNESCO



Los datos están almacenados en el motor de bases de datos de Viruoso Opensource y las consultas se realizan con SPARQL:

http://dbpedia.org/sparql

Ilustración 2-12: Motor consulta DBPEDIA



Además de esta página se ha tomado el fichero que contiene los datos que serán introducidos en los gestores RDF del proyecto para poder realizar la parte practica del mismo.

2.6 RDF ("Resource Description Framework")

Como ya se ha dicho en puntos anterior RDF es el lenguaje que la W3C propone para definir metadatos en la Web y ser el estándar universal. Estas siglas significan Marco de Descripción de Recursos (Resource Description Framework). RDF básicamente es un framework para intercambiar y describir metadatos. De esta manera RDF permite el intercambio de información entre distintas aplicaciones sí que exista una pérdida de significado [1].

Con RDF se va a poder representar metadatos sobre recursos de la Web, como puede ser el autor, la fecha de creación de una página Web… Incluso pueden representar información acerca de cosas que pueden ser identificadas en la Web aunque estas no puedan ser recuperadas directamente en la Web, como puede ser el caso de artículos que se venden en una tienda on-line, sus precios especificaciones…

Este modelo, que pretende representar los recursos de la Web, se basa en la idea de convertir las declaraciones de los recursos en expresiones con la forma sujeto-predicado-objeto (conocidas en términos RDF como tripletas). Ahora vamos a analizar las tres partes que forman as tripletas en RDF:

- Sujeto: será cualquier objeto Web identificable unívocamente por un URI (es una cadena corta de caracteres que identifica inequívocamente un recurso). Puede ser de un documento HTML, una parte de una página Web, una colección de páginas, un sitio Web completo, y en general, cualquier recurso entendido como objeto de información.
- Predicado: es la propiedad o relación que se quiere establecer con el sujeto (el recuro) y de esta manera describirlo.
- Objeto: es el valor de la propiedad o el recurso con el cual se quiere establecer la relación.

RDF está basado en la idea identificar cosas usando URIs. RDF usa *URIref* (URI references) para nombrar al sujeto, a la propiedad o al valor de algún enunciado RDF. Un URIref es un URI junto con un identificador al final. Por ejemplo, el URIref → http://www.ejemplo.org/index.html#seccion2 consta de las siguientes partes: el URI→ http://www.ejemplo.org/index.html y (separado por el carácter #), el

identificador seccion2. Esto se puede ver más claro con un ejemplo: Supóngase el que una persona crea una página en Internet (www.mipagina.com/index.html). Si ese persona describe este hecho en un lenguaje como el español crearía una sentencia como esta:

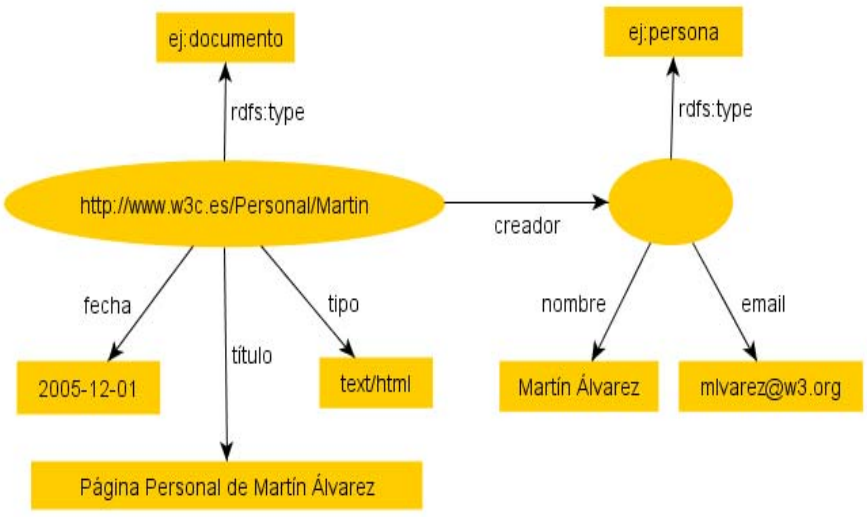
www.mipagina.com/index.html tiene un creador que es Jorge

En esta frase se puede distinguir claramente los 3 elementos citados anteriormente, el objeto que sería la pagina Web, la propiedad que describe esta sentencia que sería "creador" y el objeto que sería Jorge (que es el valor de la propiedad creador). El sujeto o recuso en este caso es identificado por la URL[9] (que es un tipo particular de URI), el predicado por la palabra creador y el objeto por otra palabra Jorge [2].

Como se ve con el ejemplo se puede describir recursos realizando este tipo de frases o sentencias e identificarlos con URIs.

Estas sentencias se pude representar gráficamente mediante un diagrama de nodos y arcos. Donde los nodos serán el sujeto y el objeto y el predicado será el arco, empezando este siempre en el sujeto y terminando en el objeto. Un ejemplo de esta representación es:

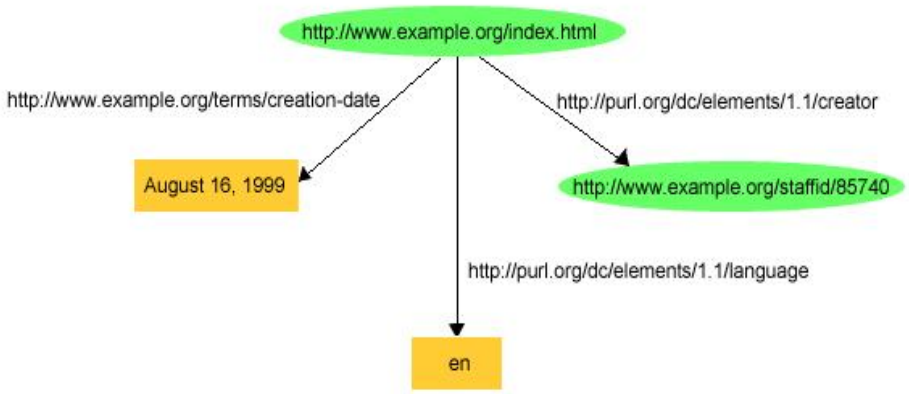
Ilustración 2-13: ejemplo 1 de diagrama de nodos y arcos



En la Ilustración 2-13 se ve como gráficamente se representa lo mismo que antes. El recuro http://www.w3c.es/Person/marti(el sujeto) tiene una propiedad que es tipo (el predicado) y esta tiene un valor que es text/html (el objeto).

Otro ejemplo de esta representación gráfica de las sentencias RDF sería:

Ilustración 2-14: ejemplo 2 de diagrama de nodos y arcos



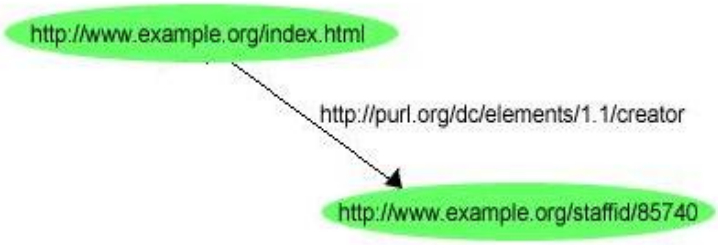
Como se puede ver en la Ilustración 2-14 los sujetos y los predicados van a ser identificados mediante URIs y los objetos podrán ser literales o valores concretos ya que representa el valor de la propiedad de ese recurso.

Esta es la forma de representar gráficamente las sentencias RDF pero también se puede hacer mediante las tripletas. Cada sentencia representada en el grafico RDF equivale a una tripleta formada por sujeto-predicado-objeto. De esta manera el grafo del ejemplo 2 quedaría representado mediante tripletas de la siguiente manera:

1ª Tripleta:

<http://www.example.org/index.html>
<http://purl.org/dc/elements/1.1/creator>
<http://www.example.org/staffid/85740>.
Que correspondería a esta parte del grafo:

Ilustración 2-15: Representación gráfica de una tripleta RDF

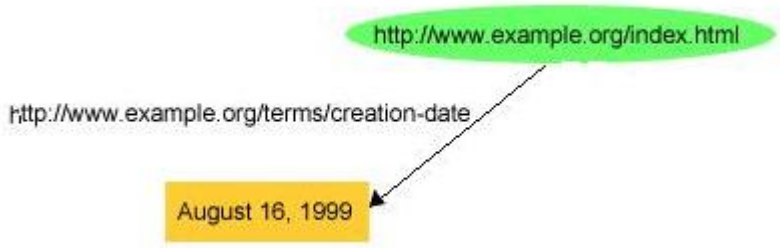


2ª Tripleta:

<http://www.example.org/index.html>

<http://www.example.org/terms/creation-date> "August 16, 1999".

Ilustración 2-16: tripleta RDF

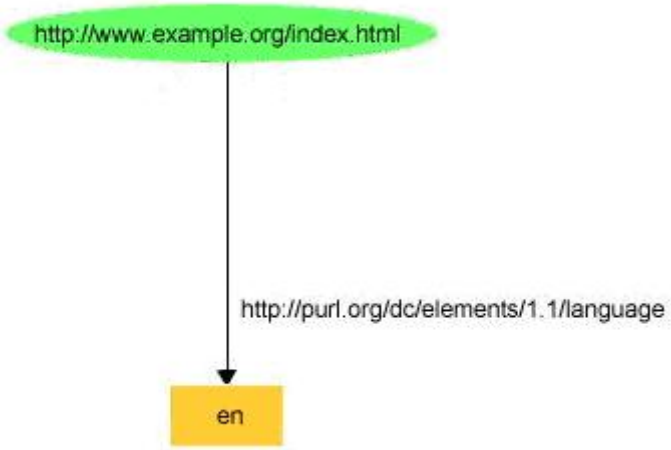


3ª Tripleta:

<http://www.example.org/index.html>

<http://purl.org/dc/elements/1.1/language> "en".

Ilustración 2-17: Tripleta RDF



Como se ve, esta forma de representación tiene la desventaja que las sentencias pueden ser muy largas y muy repetitivas (las URIs pueden ser muy largas), por eso se usa lo que se llama Qnames y que sólo son abreviaturas para los URIref's, no son más que espacios de nombres (los espacios de nombres (namespace) se usan para agrupar nombres y de esta manera evitar ambigüedad a la hora de usarlos (son etiquetas). El ejemplo anterior quedaría de esta manera:

Prefix rdf:, namespace URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#

prefix rdfs:, namespace URI: http://www.w3.org/2000/01/rdf-schema#

prefix dc:, namespace URI: http://purl.org/dc/elements/1.1/

prefix owl:, namespace URI: http://www.w3.org/2002/07/owl#

prefix ex:, namespace URI: http://www.example.org/ (or
http://www.example.com/
prefix xsd:, namespace URI: http://www.w3.org/2001/XMLSchema#

prefix exterms:, namespace URI: http://www.example.org/terms/

prefix exstaff:, namespace URI: http://www.example.org/staffid/

Una vez se tienen declarados los prefijos ya se podría declarar las tripletas, que quedan de la siguiente manera:

ex:index.html dc:creator exstaff:85740 .
ex:index.html exterms:creation-date "August 16, 1999" .
ex:index.html dc:language "en" .

Como se ve esta otra manera de declarar las tripletas puede resultar mucho más cómoda cuando se tengan que declarar muchas, reduciendo mucho su tamaño además ahorrando también tiempo y una forma de organizar la información mejor y más clara.

Dos importantes espacios de nombres que ayudan a describir las propiedades principales de este modelo son;

URIref ==> *http://www.w3.org/2000/01/rdf-schem*a# que es el núcleo del vocabulario del esquema RDF denominado informalmente '**rdfs'**.

URIref ==> *http://www.w3.org/1999/02/22-rdf-syntax-ns#* que es el espacio de nombres principal de RDF, denominado informalmente como '**rdf**'.

Como se ha visto hasta ahora toda la información de recursos se representa mediante sentencias, sin embargo no todo se puede expresar en una sola sentencia, por lo que se requiere que se describa cada una de las propiedades del objeto en términos de otros enunciados. Por ejemplo escribir:

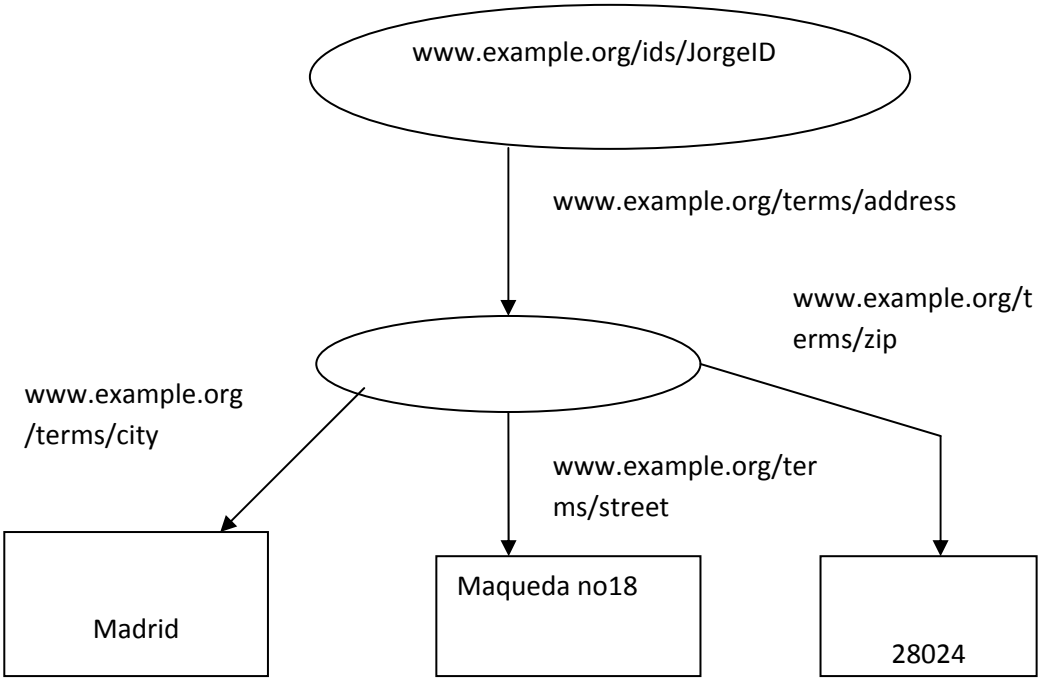
"Jorge tiene la dirección Maqueda no.18 C.P 28024, Madrid"

Requiere especificar por separado cada uno de los componentes de la dirección (si queremos decir la dirección de otra forma a la que está escrita). Entonces lo que tendría que escribir varios enunciados como:

1) Jorge tiene la dirección *X*
2) X se refiere a la calle Maqueda no.18
3) X tiene el código postal 28024
4) X se refiere a la ciudad de Madrid

Para esta definición se tendrá la siguiente representación:

Ilustración 2-18: Representación gráfica de la sentencia



El identificador que en el ejemplo se ha denominado X, no es más que un nodo intermedio que sirve para representar este tipo de sentencias y que se denominará nodo en blanco (blank node). Los nodos blancos son llamados "recursos anónimos" y se representan añadiendo el prefijo "_:" (el identificador de los nodos blancos) a algún otro identificador que nosotros inventemos. Por ejemplo, para lo visto en el ejemplo anterior:

_:direcciónJorge

_:direccionJorge exterms:street "Maqueda no.18”

Por último dentro del predicado pueden aparecer en lugar de una URIs como las que se han visto hasta ahora, alguna que hace referencia a un tipo de datos. Estos tipos de datos pertenecen al esquema XML: datatype, estos datos pueden definir los datos que se quieren utilizar dentro del esquema XML, ya sea este una marca de idioma, un integer, un literal, una fecha…

Como la mayoría de las tripletas RDF son serializadas por archivos XML, dentro de ellas es fácil encontrar atributos que pertenecen al esquema XML de tipos de datos.

Vamos a mostrar a continuación algunos de estos tipos que nos podemos encontrar lo largo de este proyecto:

"Democratic Party (US)"@en .

"211136"^^<http://www.w3.org/2001/XMLSchema#Integer>. →Para un integer

"1987-07-08"^^<http://www.w3.org/2001/XMLSchema#Date> →para definir formato fecha

http://www.w3.org/2001/XMLSchema#related.resources

http://www.w3.org/TR/xmlschema-2/#purpose

3. SISTEMAS GESTORES RDF

En este capítulo se analizan las características o aspectos más teóricos de los gestores RDF, comparando los cuatro gestores RDF llevados a estudio en este proyecto.

3.1 Arquitectura de los gestores RDF

En este apartado se va analizar el esqueleto de cada uno de los gestores RDF. Se va a estudiar las capas que forman cada uno de ellos y ver que aportan y para qué sirven cada una de ellas.

Una vez que se conozcan cada una de las capas que componen los gestores se analizara que aportan cada una de ellas en lo que se refiere al almacenamiento y manipulación de las tripletas RDF. Se quiere conocer como se almacenan los datos dentro del gestor RDF y ver de qué manera se consigue interactuar con los datos almacenados.

La arquitectura es una parte importante del análisis de los gestores RDF ya que al ser la base sobre la que se asienta, puede servir para ver las desventajas o ventajas en cada uno de ellos en la forma de tratar a las tripletas RDF que se inserten en cada uno de los gestores.

3.1.1 Oracle

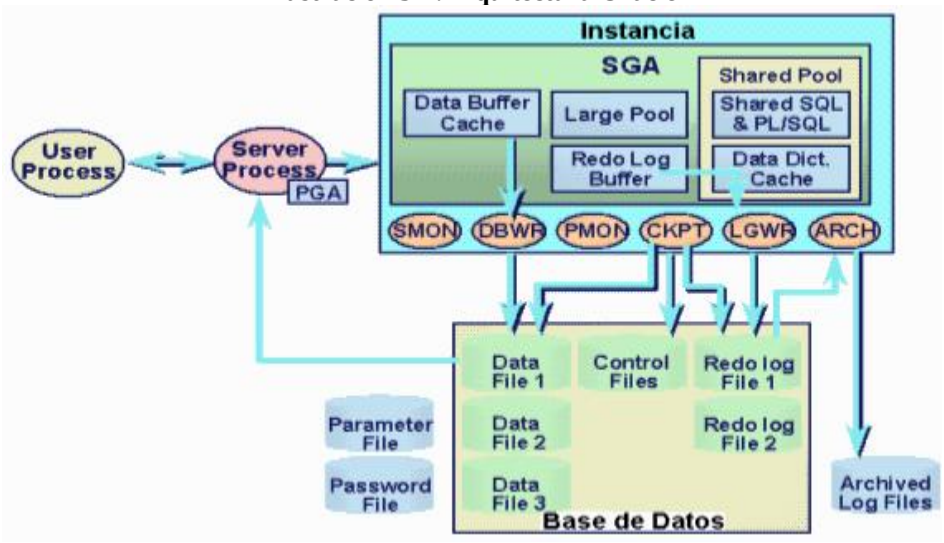
Oracle básicamente es una herramienta cliente/servidor para la gestión de Bases de Datos. Para trabajar en Oracle utilizamos SQL que ayuda a tratar y gestionar la base de datos, junto con los paquetes SDO_RDF que van a ser los encargados de permitir trabajar con las tripletas RDF dentro de Oracle [8].

La versión de Oracle con la que se trabaja en este proyecto es Oracle Database 10g Release 2. Oracle es una base de datos del tipo relacional, que es una base de datos que cumple con el modelo relacional, que a su vez es un modelo de datos basado en la lógica de predicado y en la teoría de conjuntos. Su idea fundamental es el uso de

"relaciones", estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados "tuplas".

En la ilustración 3-1 que se muestra a continuación se ve un diagrama de la arquitectura de Oracle:

Ilustración 3-1:Arquitectura Oracle



Pero hablar de la arquitectura en Oracle de una forma detallada no es el propósito de este apartado, pero si el ver como a partir de esta estructura se puede trabajar con tripletas RDF dentro de Oracle. Para información más detallada de la arquitectura de Oracle consultar las referencias [24] y [25]:

Dentro de Oracle interesan especialmente los paquetes SDO_RDF que son los que van a permitir trabajar con tripletas RDF. Estos paquetes van a permitir crear tablas donde van a ir las tripletas y el modelo RDF al cual van a pertenecer éstas.

Todas las tripletas RDF son analizadas y almacenadas en el sistema como entradas en tabla bajo el esquema MDSYS (es un esquema que prescribe el almacenaje, la sintaxis, y la semántica de los tipos de datos apoyados). La tripleta RDF va a ser tratada dentro de Oracle como un objeto de la base de datos, como resultado de ello, un solo archivo RDF que contiene varias tripletas RDF tiene como resultado múltiples objetos en la base de datos. A su vez todos los sujetos y objetos de las tripletas se asignan a nodos de una red, y los predicados se asignan a una red de vínculos que tiene como nodo final y nodo inicial, un sujeto y un objeto, respectivamente. Los tipos de nodos en una red RDF serán nodos en blanco, literales, URI y tipos soportados.

Para almacenar datos RDF dentro de Oracle se tienen que cumplir los siguientes requerimientos:

- Un sujeto debe ser un nodo en blanco o una URI.
- Un predicad debe ser una URI.
- Un objeto puede ser cualquier tipo, como URI, nodos en blanco o literales. (Sin embargo valores nulos no son soportados).

3.1.2 Kowari

Kowari es un programa de código abierto, que es una base de datos java (está montada sobre java) transnacional[11] (son aquellas bases de datos que en caso de que una operación de escritura no se completará con éxito por algún motivo, podrían volver atrás). Kowari implementa muchos de los conceptos de W3C Web Semántica y realiza las de declaraciones de los datos de la forma sujeto-predicado-objeto de manera similar a la estándar que es W3C Marco de Descripción de Recursos (RDF) [4].

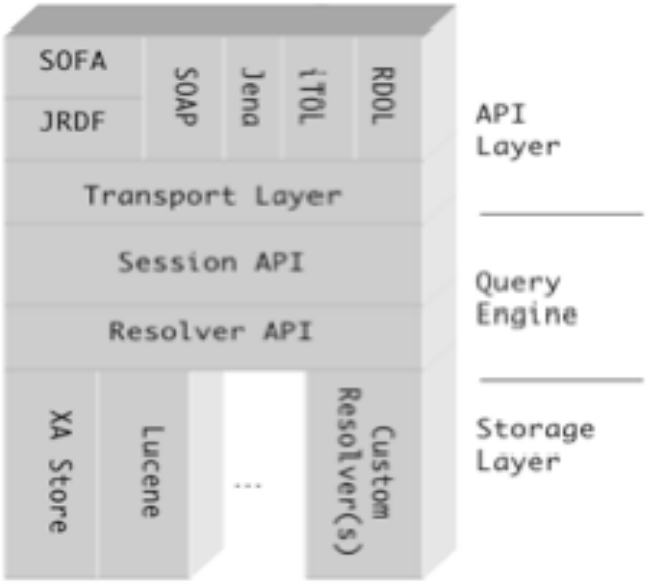
Kowari puede correr en cualquier sistema operativo que soporte Java 1.4 o versiones posteriores. Kowari puede ser desplegado en PCs estándar basados en una arquitectura de 32 bits (por ejemplo, Intel ® Pentium ® II o III procesadores y 64 MB de memoria) o la utilización de servidores de rango medio 32 o arquitecturas de 64 bits (por ejemplo, Sun EnterpriseTM, CobaltTM o NetraTM gama De los servidores). Kowari se implementa en el lenguaje de programación Java y es 100% Java. Depende de paquetes del estándar de Java de Sun MicrosystemsTM.

Kowari es un nuevo tipo de base de datos, optimizada para el almacenamiento y la gestión de los metadatos. Una base de datos más general permite almacenar cualquier tipo de datos, al estar Kowari creada explícitamente para trabajar con metadatos, se pretende lograr una gran eficiencia a la hora de trabajar con ellos.

Kowari trabaja con el lenguaje de consulta ITQL parecido a SQL pero con algunas diferencias debidas a la forma de almacenar los datos que tiene Kowari.

En la ilustración 3-2 se puede ver la arquitectura de este sistema:

Ilustración 3-2:arquitectura Kowari



La capa de arriba proporciona acceso API[12], que como se puede ver la capa API apoya muchos de los estándares de acceso API de la Web semántica. Estos estándares básicamente son: SOFA[13], Java RDF[14], iTQL[15], RDQL[16] y SOAP[17].

Justo debajo esta el motor de consulta y por último el sistema de almacenamiento. Entre ellos esta Resolver API que es la capa de abstracción que debe este nombre por la resolución de las restricciones (una consulta es desglosada en restricciones individuales) de las sentencias RDF por la capa de almacenamiento cuando estas son requeridas por el motor de consultas. Existen dos tipos de Resolvers APIs: las internas y las externas. Las internas son aquellas donde las fuentes de datos están controladas por Kowari (por ejemplo los datos nativos de Kowari los XA Statement Store) y los externos, aquellos que no son controlados por Kowari, como pueden ser los datos que provienen de una base de datos relacional.

Los datos que almacenamos son representados en la capa de almacenamiento de forma numérica de manera que estos datos son locales a Kowari y no pueden ser usados de forma externa, por eso antes de ser de vueltos al usuario tienen que pasar de esta representación numérica a una forma de representación global entendible para el usuario (esto será el proceso de globalización). (Nota: Se darán más detalles de esta capa en el siguiente apartado modelo de datos).

Respecto al motor de consulta, este usa un modelo de objetos abstracto que está ligado a una sintaxis concreta en las consultas. Esta sintaxis permite de forma fácil la integración de las APIs y la utilización de diferentes lenguajes de consulta. Por defecto el motor de consultas de Kowari ejecutara comandos que estén en ITQL (que será el lenguaje de consulta que utilizaremos nosotros).

Cuando una consulta llega al motor de búsqueda lo primero que hace este es estimar el tamaño que tendrá el resultado de esa consulta mediante estimaciones que hace a partir de las restricciones que se imponen en la consulta. Además a partir de estas estimaciones se hace un plan inicial para la consulta, que a lo largo de esta según los resultados será modificado en busca de un plan más eficiente. Por último utilizando operaciones de "join" (operaciones de unión) con los resultados de cada una de las restricciones que aparezcan en la consulta se devolverá el resultado final al cliente que solicito la consulta.

3.1.3 Mulgara

Mulgara es un programa de código abierto, que es una base de datos java (está montada sobre java) al igual que Kowari. Mulgara es una nueva base de datos para metadatos que surge después que la empresa que compro Kowari en 2006 dejara estancado el desarrollo de este proyecto, la comunidad de personas que trabajaba en Kowari (no todas, un grupo de personas que estaba involucrada en Kowari) decidió crear Mulgara. Mulgara sigue implementando muchos de los conceptos de W3C Web Semántica y realiza las de declaraciones de los datos de la forma sujeto-predicado-objeto de manera similar a la de W3C Marco de Descripción de Recursos (RDF) estándar [5].

Mulgara no es una base de datos basada en el modelo de las bases de datos relacionales (como sucede en Oracle), debido al gran número de operaciones Joins (Es una combinación de dos o más tablas de una base de datos relacional. Consiste en obtener el producto (multiplicación) de todas las tuplas de una tabla con las de la otra, para posteriormente evaluar aquellas cuyo campo en común sea igual generando como resultado una nueva tabla que tiene como tuplas (renglones) que cumplen con la

condición establecida) que aparecen cuando se trabaja con metadatos. Pero tampoco es una base de datos orientada a objetos ya que en este tipo de bases de datos se crea un objeto para cada sentencia y para este propósito se necesitaría mucha memoria y el tiempo de proceso sería lento. Mulgara como ya se ha dicho trata a los metadatos como sentencias sujeto-predicado-objeto, los desarrolladores de Mulgara piensan que al ser un modelo especifico para trabajar con metadatos el rendimiento y la escalabilidad será superior que el de una base de datos relacional u orientada a objetos.

Mulgara utiliza como lenguaje de consulta ITQL al igual que Kowari. Su estructura es muy similar a la estructura de SQL, lenguaje utilizado para hacer consultas en bases de datos relacionales, pero con algunas diferencias debido a la forma en que se almacenan los datos en Mulgara.

Además Mulgara puede ser utilizada, al igual que las bases de datos relacionales, como un repositorio de datos para aplicaciones software. Para ello dispone de una API abierta que soporta distintos lenguajes de programación y protocolos. De manera que, distintos tipos de usuarios interactúen con Mulgara dependiendo de sus necesidades:

- Usuarios finales que interactúan indirectamente a través de aplicaciones que utilizan Mulgara como repositorio.
- Los administradores de sistema que utilizan ITQL para modificar la base de datos, realizar copias de seguridad …
- Los programadores para realizar la integración entre sus propias aplicaciones y Mulgara.

Como se comento antes, Mulgara es creada por gran parte de la comunidad de creadores que desarrollaron Kowari por lo tanto la idea básica de la arquitectura de Mulgara es la misma que Kowari. En este proyecto se ha utilizado la versión 2.0.0 de Mulgara (publicada el 23 de julio de 2008) en ella se han producido algunas mejoras respecto a Kowari, se procede a comentar algunas de estas mejoras:

- Puede funcionar con java 5 y java 6.
- Mejor sincronización con el servidor final. Esto soluciona algunas excepciones que aparecían cuando distintos clientes accedían a la misma conexión

- Los decimales no ya no tienen que empezar con el carácter “+” cosa que antes si sucedía
- Se soportan literales cuyo tipo es xsd:date.
- Aceptación de mas códigos de idioma
- Posible uso de consultas del tipo Construct
- Nuevo comando export, que permite exportar los datos en dos tipos de formatos, RDF y N-triples.
- Mejoras en el manejo del cierre del servidor
- Nueva Connection API (para establecer una llamada con servidor especificado)
- Lenguaje de marcada para literales sin tipo (untyped literals)
- Preparación para futuro soporte SPARQL[18]

3.1.4 Sesame

Sesame es un Framework Java (Estructura de soporte, marco de trabajo) en la cual se puede almacenar y realizar consultas sobre datos RDF. El marco es totalmente abierto respecto a los formatos de los ficheros RDF (n-triples, n3...), en cuanto a lenguajes de consulta (Sparql, Serql[19]), en cuanto a los métodos de almacenamiento... En el caso de Sesame el diseño y la implementación son independientes de cualquier dispositivo de almacenamiento, de forma que permite su empleo en una variedad de dispositivos de almacenamiento tales como bases de datos relacionales, almacenamiento de triplas o bases de datos orientadas a objeto [6].

Pero este marco en si no es suficiente, también es necesario la implementación de una serie de APIs (interfaz de comunicación entre diferentes componentes de Software). Esto hace que Sesame pueda trabajar con diferentes lenguajes de consulta como Sparql y Serql, almacenar RDFs tanto en memoria como en disco,...

Sesame almacena sus datos RDF en los denominados repositorios. Un repositorio es un contenedor de almacenamiento para RDF. Esto puede significar simplemente un objeto de Java (o un conjunto de objetos de Java) en memoria, o puede significar una base de datos relacional. De cualquier forma, es importante que cada operación en Sesame ocurra relacionada con un repositorio: Cuando se añaden datos

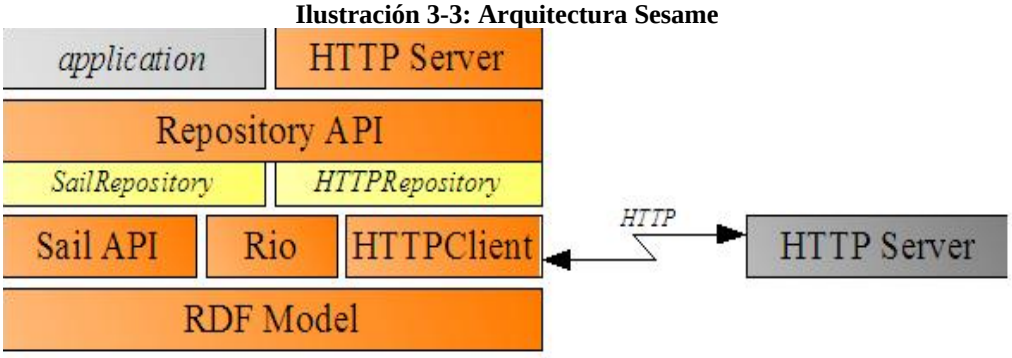
RDF, se añaden a un repositorio, cuando se realiza una consulta se realiza sobre un repositorio particular.

En este proyecto se va a trabajar con la versión 2.1.2 de Sesame que básicamente tiene las siguientes características:

- Está orientado a Java 5.
- Soporte a las transacciones y rollback[20].
- Soporte para SPARQL.
- Un REST-ful protocolo HTTP que incluye soporte para el protocolo SPARQL y los resultados de una consulta SPARQL en formato XML.

Para ver la arquitectura de Sesame se dispone de la ilustración 3-3 en la que se pueden ver las diferentes APIs que utiliza Sesame y los componentes más destacados. Aclarar que cada componente o API depende de los componentes o APIs que están debajo de él.

Ilustración 3-3: Arquitectura Sesame

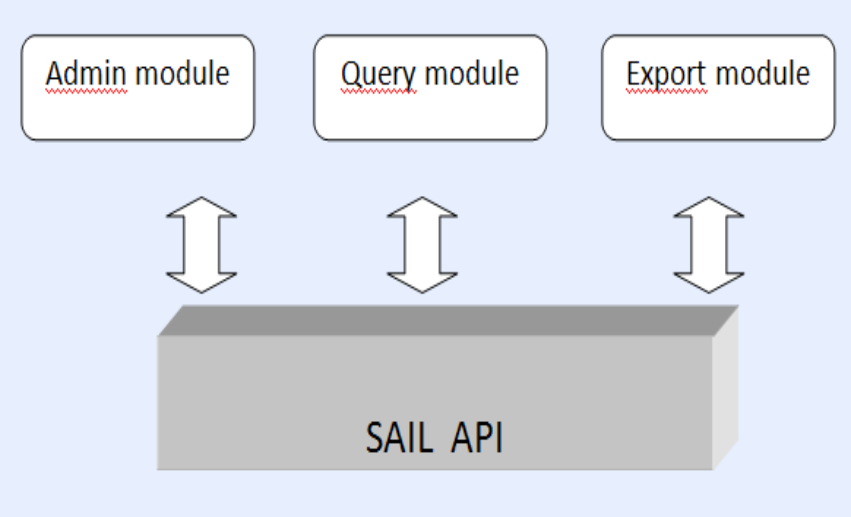


- RDF Model: es el encargado de definir todas las interfaces que están definidas en el modelo RDF y de la definición de todas las entidades básicas de RDF (esto son las URIS, los blank nodes, literales y sentencias).
- RIO: es un conjuntos de parsers (analizador) para la serialización (codificación de un objeto) de diferentes formatos de RDF (RDF/XML, Turtle, N-Triples,N3).

- Sail API: esta API va ser la que permita almacenar los datos RDF de diferentes maneras y definir la inferencia (modo de razonamiento).

- Módulos funcionales de Sesame: En este caso son 3 los módulos funcionales que se comunican con la Sail API (son clientes suyos) y , como se puede ver en la siguiente ilustración:

Ilustración 3-4: Módulos funcionales Sesame

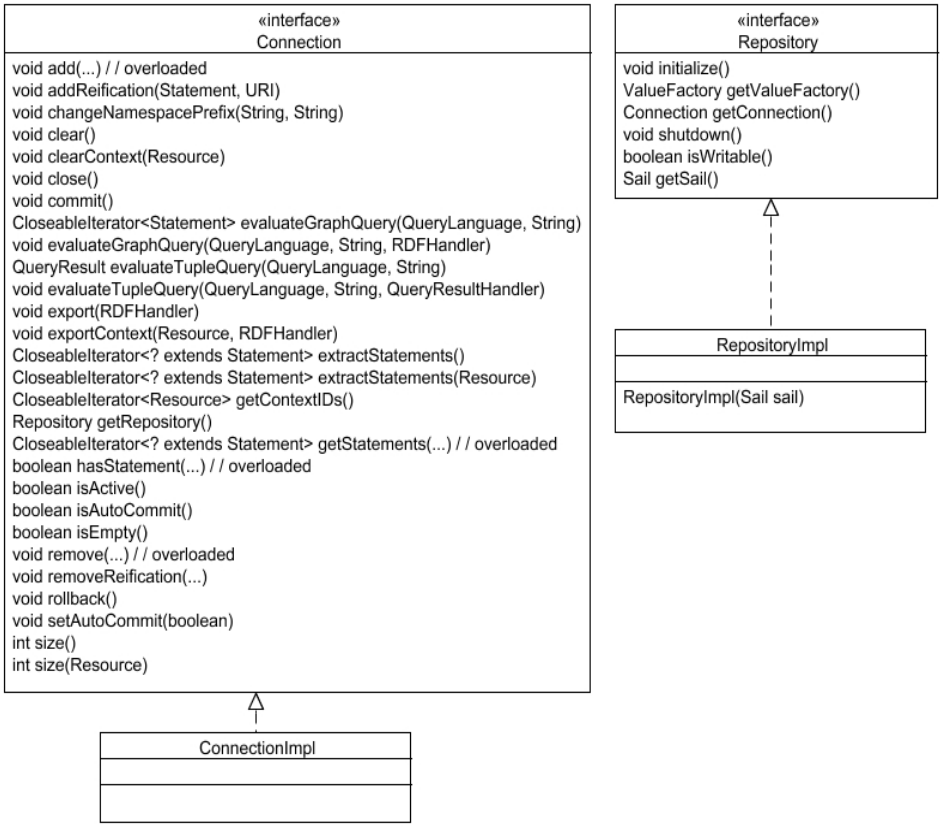


  - *Admin module:* este modulo tiene dos funciones principalmente, una la de añadir datos RDF a un repositorio y la otra limpiar un repositorio. Su funcionamiento es muy sencillo, este modulo mediante una analizador (parser en este caso ARP RDF) analiza los datos a insertar, después de analizarlo mediante la comunicación con la Sail API se insertar los datos o se comunicara si ha existido un error si esto la inserción no se ha podido llevar a cabo.
  - *Query module:* este modulo es el utilizado para realizar consultas sobre los repositorios.
  - *Export module:* este modulo se usa para poder exportar los datos que se tienen almacenados en un repositorio. Esta exportación se puede realizar en los siguientes formatos: n3, Turtle, n-triples, RDF/XML, TriX y TriG.

- Repository API: Ofrece un gran número de procedimientos para el uso/manejo de datos RDF. Estos métodos se utilizan para cargar los ficheros de de datos,

consultas y para la extracción y manipulación de datos. Existen varias implementaciones de esta API, en la ilustración 3-5 vemos dos: SailRepository y HTTPRepository (este orece una comunicación tranparente entre cliente y servidor, sobre la comunicación con un servidor de sésamo mediante HTTP). A continuación se muestra el diagrama de clases:

Ilustración 3-5: Diagrama de clases Repository API



- HTTP Server: Sesame puede ser utilizado como un servidor, este software de servidor viene en forma de dos aplicaciones Web Java Sesame (HTTP) server y OpenRDF Workbench.

Sesame Server provee de Acceso HTTP a los repositorios Sesame y puede utilizarse como acceso para otras aplicaciones. Pero esta aplicación no provee de ninguna funcionalidad para usuarios, para ello está OpenRDF Workbench.

OpenRDF Workbench proporciona un interfaz Web para poder realizar consultas, actualizar datos y explorar los repositorios del Sesame Server.

3.1.5 Conclusiones arquitectura

La tabla 3-1 es un resumen en el que se muestran las principales características de cada uno de los gestores RDF.

Tabla 3-1: Características generales gestores RDF

	Oracle	**Kowari**	**Mulgara**	**Sesame**
Tipo de base de datos	Relacional	Propio tipo orientado a trabajo con metadatos	Propio tipo orientado a trabajo con metadatos	Soporta orientadas a objetos y relacionales
Sistemas soportados	Windows, Unix, Solaris, AIX (PPC64) Clusterware) y HP-UX .	Microsoft Windows NT, Windows 2000 y XP UNIX® and Linux, Solaris, Mac OS X y IRIX	Microsoft Windows NT, Windows 2000 y XP, UNIX and Linux, Solaris, Mac OS X y IRIX	Windows y Unix, Solaris, Mac OS y Irix
Lenguaje de consulta	SQL	ITQL	ITQL	SPARQL Y SERQL
Requerimientos sistema	1024 MB de RAM Disco de al menos 7 GB	Instalación J2SE 1.4	Instalación J2SE 1.4 o superior	Instalación J2SE 1.4 o superior

Llegado este punto se puede analizar una de las principales diferencias que existen entre Oracle y el grupo que forman Mulgara, Kowari y Sesame. Estas últimas han creado su propia capa de almacenamiento para almacenar las tripletas RDF mientras que Oracle no ha creado ninguna capa sino que lo ha hecho a través de la capa Oracle's Network Database (que está a su vez construido en la capa de aplicaciones), esto va a provocar que la escalabilidad en Oracle según tengamos una cierta cantidad de datos en nuestra base de datos va ser mucho peor que en las otras. Quizás para una cantidad pequeña de datos, pero cuando la base de datos contiene un gran número de datos como

sucede este proyecto la escalabilidad y el rendimiento de la base de datos no va ser todo lo bueno que se podría esperar de Oracle. Esto se podrá colaborar una vez llegado al punto el que se miden los resultados de los gestores.

Respecto a los otros gestores RDF (Sesame, Mulgara y Kowari), todos tiene una capa pensada para la comunicación con otras aplicaciones dependiendo del ambiente en el que se implante estos gestores. En la siguiente tabla se muestran dichos métodos:

Tabla 3-2: Métodos de comunicación

Métodos comunicación Mulgara y Kowari	Métodos comunicación Sesame
-SOAP -Jena -ITQL, RDQL -SOFA -JRDF	-Vía Http para contexto Web -SOAP -RMI(es un mecanismo ofrecido en Java para invocar un método remotamente, solamente comunica servidores codificados para Java.

Este grupo de gestores también dispone de un motor de consultas específico para este tipo de datos. Por un lado está el Query Engine de Mulgara y el módulo de consultas de Sesame. Donde la mayor diferencia que existe entre ambos es que cuando Sesame recibe una consulta este realiza un “pasing” (análisis sintáctico de la consulta) que crea un árbol de consulta que a su vez pasa a un optimizador de consultas que lo que consigue es un modelo equivalente a la consulta pero más eficiente. EL modelo optimizado es seguidamente evaluado, siguiendo la estructura de árbol generada, cada objeto representa una unidad básica en la consulta original y se evalúa a si mismo extrayendo información por medio de “SAIL” cuando lo necesita. La gran ventaja de este enfoque es que los resultados se retornan de la misma forma subsiguiente, en vez de construir primero la consulta completa en memoria.

3.2 Modelo de datos en los gestores RDF

En esta apartado se analizará con que estructuras se almacenan los datos, el uso o no de índices, ventajas que obtenemos por la utilización de índices…

Un modelo de datos es un conjunto de conceptos que sirven para describir la estructura de una base de datos: los datos, las relaciones entre los datos y las restricciones que deben cumplirse sobre los datos. Los modelos de datos contienen también un conjunto de operaciones básicas para la realización de consultas (lecturas) y actualizaciones de datos.

Son la parte que describa el modelo de datos, se clasificarán a éstos de dos formas: modelos conceptuales y modelos físicos.

Los modelos físicos proporcionan conceptos de cómo se almacenan los datos en nuestra computadora, las estructuras que utilizan los índices (arboles AVL…)

Lo modelos conceptuales nos muestran la base de datos mediante entidades, atributos y relaciones, dando una idea de las relaciones y la composición de los datos que estamos almacenando en nuestra base de datos.

3.2.1 Oracle

Como se ha comentado en anteriores apartados los paquetes SDO_RDF son los que permiten trabajar con tripletas RDF dentro de Oracle. Estos paquetes van a permitir crear tablas donde van a ir las tripletas y el modelo al cual van a pertenecer [10].

El procedimiento SDO_RDF.CREATE_RDF_NETWORK va ser el encargado de dar soporte RDF a nuestra base de datos en Oracle. Este procedimiento es el encargado de crear las tablas y objetos necesarios para trabajar con RDF dentro de Oracle.

Una vez aplicado el procedimiento anterior se pueden crear las tablas donde se van a insertar las tripletas RDF. Las tablas donde se almacenan tripletas RDF constan de un identificador y de la tripleta RDF compuesta por sus tres elementos (sujeto, predicado y objeto).

El siguiente paso para poder trabajar con RDF es crear un modelo de datos con el procedimiento SDO_RDF.CREATE_RDF_MODEL especificando un nombre para el modelo, así como la tabla y la columna para mantener la referencia con los datos RDF. Además el sistema genera automáticamente un modelo de identificación. Se pude ver de forma más clara en la tabla 3-3:

Tabla 3-3: MDSYS.RDF_MODEL$

Nombre columna	Tipo de dato	Descripción
OWNER	VARCHAR2(30)	Propietario del modelo RDF
MODEL_ID NUMBER,	Unique model ID number	Numero generado automáticamente por el sistema
MODEL_NAME	VARCHAR2(25)	Nombre del modelo
TABLE_NAME	VARCHAR2(32)	Nombre de la tabla RDF a la que hace referencia
COLUMN_ NAME .	VARCHAR2(32)	Nombre de la columna SDO_RDF_TRIPLE_S en la tabla RDF que hace referencia a los datos RDF del modelo

También se crea la tabla *MDSYS.RDFM_model-name* que es una vista de las tripletas RDF asociadas con el modelo que se ha creado:

Tabla 3-4: MDSYS.RDFM_model-name

Nombre columna	Tipo de dato	Descripción
Link_ID	Number	Identificador de la tripelta
Start_node_id	Number	El identificador (value_ID) del sujeto de la tripleta
Link_type	Varcahr (200)	El tipo del predicado
End_node_id	Number	El objeto (value_ID) del sujeto de la tripleta
Active	Varchar(1)	El estado del link en la red.”y” si esta activa, “n” si no lo esta
Model_id	Number	El identificador del modelo al que la tripleta pertenece

La tabla 3-4 muestra solo las columnas más importantes de la tabla *MDSYS.RDFM_model-name*.

Y por último está la tabla *MDSYS.RDF_VALUE$* contiene información acerca de los sujetos, predicados y objetos que forman las tripletas RDF:

Tabla 3-5: MDSYS.RDF_VALUE$

Nombre de columna	Tipo de dato	Descripción
Value_ID	Number	Número identificador
Value_Name		Valor del texto para una de las partes de las tripletas, si fuera mayor de 4000 caracteres se alamcenaría en la coñumna Long_value
Value_type	Varchar2(10)	El tipo del texto almacenado en la columna value_name
Literal_type	Varchar2(4000)	El tipo para los literales(el tipo de información)
Language_type	Varchab2(80)	Idioma
Long_value	Clob	Si esta parte de la tripleta tiene es mayor que 4000 caracteres se almacena aquí si no tendrá un valor nulo.

Por último se comentaran los índices, que sirven para tener un acceso más rápido a los datos contenidos dentro de una tabla y tener un mejor rendimiento de la base de datos. Inmediatamente después de crear el índice, Oracle comienza a mantenerlo de acuerdo a las inserciones, actualizaciones y eliminaciones de registros de la tabla en la cual se ha implementado.

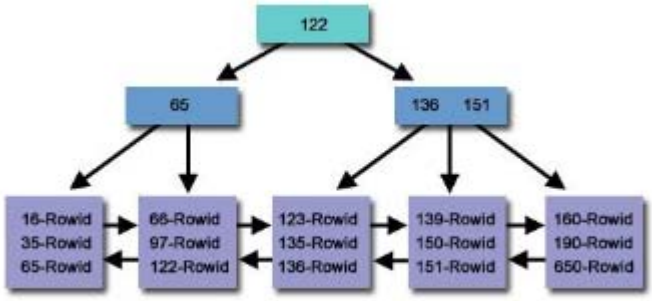
Cuando se crea un índice (de cualquier tipo) también se crea un segmento de datos para guardar esa información.

Oracle permite la creación de índices B* (similares a los B+). Ventajas de la estructura de B-tree:

• Todos los nodos hoja están a la misma profundidad. La recuperación de cualquier registro lleva aproximadamente el mismo tiempo.
• Permanecen automáticamente balanceados.
• Todos los bloques del árbol están llenos de media en ¾.
• Excelente desempeño para una amplia variedad de consultas, desde emparejamiento exacto (exact match) a búsquedas por rango de valores.
• Inserts, updates, y deletes son eficientes, manteniéndose el orden de clave.

Un ejemplo gráfico sería:

Ilustración 3-6: ejemplo arboles B*



Al crearse una tabla en Oracle se crean los siguientes índices:

- Un índice UNIQUE basado en arboles B* para tener las columnas que se hayan definidas como clave primaria de una tabla.
- Un índice UNIQUE basado en arboles B* para mantener la restricción de unicidad de cada grupo de columnas que se haya declarado como único utilizando la restricción constraint.
- Un índice basado en árboles B* para mantener las columnas que se hayan definidas como clave primaria y todas las filas de una tabla organizada por índice.
- Un índice basado en hashing para mantener las filas de un grupo de tablas (“clúster”) organizado por hash.

Aparte en este proyecto se van a crear 2 índices una para el sujeto y el predicado de nuestra tabla RDF (más adelante se podrá ver la sintaxis de creación de dichos índices).

3.2.2 Kowari y Mulgara

El almacenamiento de datos en Kowari (el XA Statement Store) se hace a partir de la versión nativa de Java 1.4 de almacenamiento de datos transaccional que almacena datos RDF en disco (Mulgara suporta la versión 1.5 y superiores a diferencia de Kowari).

Los datos se almacenan como cuádruplas. Estas cuádruplas están formadas por sujeto, predica, objeto y meta-nodo. Los tres primeros son estándares para las sentencias RDF (como ya se comento en la introducción) y el meta-nodo nos da información sobre el modelo de sentencia que aparece. Cada cuádrupla es única, por eso para una sentencia que aparece dos veces en un modelo, el valor del meta-nodo es diferente.

Kowari a la hora de almacenar estas sentencias RDF lo hace en seis órdenes diferentes, estos corresponden al mínimo número de caminos en que los 4 tipos de nodos (sujeto, predicado…) pueden ser recorridos, a la hora de de buscar cualquier sentencia con la que coincidan. Estos seis órdenes son los correspondientes ***índices*** que crea y con los que trabaja Kowari. Estas son los 6 índices: sujeto-predicado-objeto, sujeto-objeto-predicado, predicado-sujeto-objeto, predicado-objeto-sujeto, objeto-sujeto-predicado y objeto-predicado-sujeto.

Estos índices se basan en un híbrido entre árboles AVL y árboles B.

Para incrementar la velocidad y reducir la ***redundancia***, los nodos que componen sentencias son almacenados como integers de 64 bits, localizados en el llamado “Node pool” (es un nodo de agrupación). A su vez estos nodos corresponden a URIs y literales que son almacenados en el denominado “String pool”(es una colección de referencias). Con esto podemos decir que el almacenamiento de sentencias esta esencialmente compuesto de estos dos elementos (node pool y string pool) y nos provén de los requerimientos para almacenar sentencias RDF.

El String pool es construido usando un árbol AVL para almacenar y ordenar estos elementos. A partir del AVL se almacenan los índices dentro de archivos de piso (los archivos de piso: son normalmente los conjuntos de datos básicos que se utilizan para el almacenamiento de datos de configuración de aplicaciones y programas [26]) con cualquier tipo de dato RDF, entendiendo como dato RDF URIs, literales y todos los tipos de datos del esquema XML. Estos datos son ordenados en el árbol según la semántica de cada uno de ellos. Por último también se dispone de un índice para localizar cualquier entrada de acuerdo con el número del nodo.

Una ventaja de Kowari respecto al formalismo que utilizan otras bases de datos (por ejemplo SWRDB) es que consideran al predicado como cualquier otro elemento de la relación y no de forma distinta. Así que como ya hemos dicho cada sentencia RDF implica su indexación de forma paralela en seis índices diferentes, como vimos anteriormente.

Los árboles AVL son árboles binarios de búsqueda que están siempre equilibrados de tal modo que para todos los nodos, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha. Gracias a esta forma de equilibrio (o balanceo), la complejidad de una búsqueda en uno de estos árboles se mantiene siempre en orden de complejidad O (log n). El factor de equilibrio puede ser almacenado directamente en cada nodo o ser computado a partir de las alturas de los subárboles.

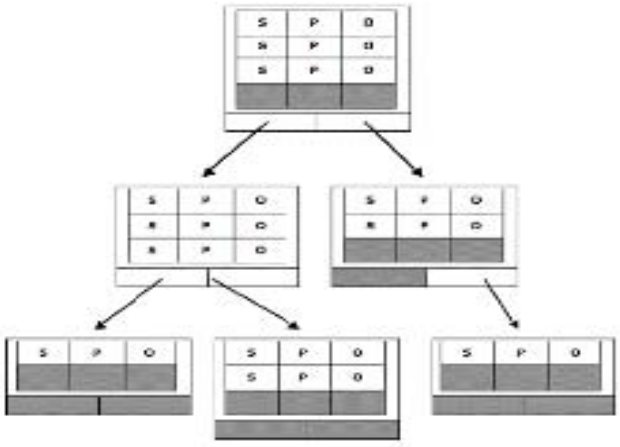
Los árboles AVL son particularmente útiles para almacenar RDF cuando las restricciones de una consulta son uno o varios elementos de la tripleta (a la cuádrupla si la información del modelo es añadida a la tripleta RDF). Las tripletas se pueden ser indexadas en orden y crear índices en paralelo. Una restricción de uno o varios elementos de la tripleta puede ser resulta en una simple línea de los índices.

A continuación se ve qué información tiene cada nodo de los árboles AVL utilizados en los índices, para que sea más sencillos se va a ignorar el meta-nodo y de esta manera reducir el número de índices a tres. Cada uno de estos 3 árboles tiene una clave de ordenación diferente, son estas tres: (sujeto, predicado, objeto), (predicado, objeto, sujeto) y (objeto, sujeto, predicado). Cada nodo de cada uno de estos índices tiene la siguiente información:

- Un conjunto de tripletas ordenado según la clave de ordenación para ese índice.
- El número de tripletas en l conjunto de ese nodo.
- Una copia de la primera tripleta.
- Una copia de la última tripleta.
- El identificador del subárbol izquierdo.
- El identificador del subárbol derecho.
- La altura del subárbol cuya raíz es este nodo.

En la ilustración 3-7 se muestra un ejemplo de inserción de tripletas RDF en un árbol AVL, en el cual se ha suprimido el meta-nodo para una mayor simplicidad:

Ilustración 3-7: Árbol AVL

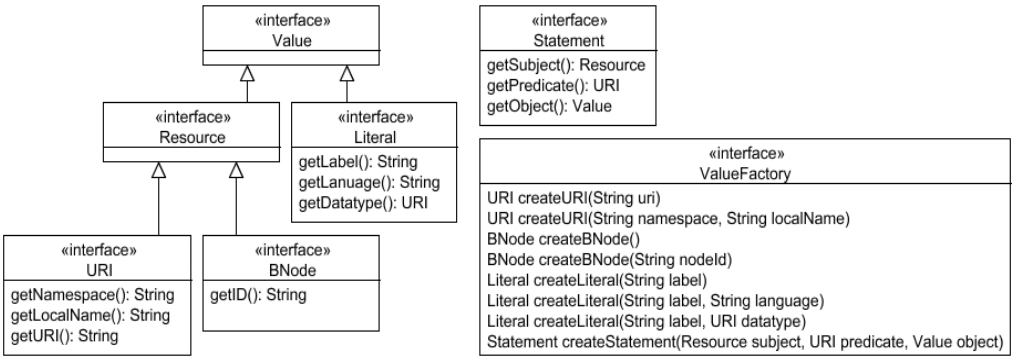


3.2.3 Sesame

En Sesame las tripletas RDF también son almacenadas como cuádruplas formadas por el sujeto, predicado y objeto de la tripleta, y un cuarto elemento que introduce Sesame que es el contexto [6].

Como ya se ha dicho el modelo de datos es el encargado de definir todas las interfaces que están definidas en el modelo RDF y de la definición de todas las entidades básicas de RDF (esto son las URIS, los blank nodes, literales y sentencias). En la ilustración 3-8 se puede ver el diagrama de clases que muestra las interfaces que están definidas en Sesame's RDF Model:

Ilustración 3-8: Modelo Sesame



Sesame almacena los datos en repositorios y permite elegir entre distintas formas de almacenamiento en estos. Estas formas de almacenamiento son las siguientes:

- *Memory*: una memoria basada en un repositorio RDF.
- *Memory-rdfs*: un repositorio en memoria principal con un RDF Schema de inferencia.
- *Memory-rdfs-dt*: un repositorio en memoria principal con un RDF Schema y una jerarquía directa de inferencia.
- *Native*: repositorio que utiliza una estructura de datos en disco.
- *Native-rdfs:* repositorio native pero con un RDF Schema de inferencia.
- *Native-rdfs-dt*: repositorio native pero con un RDF Schema y una jerarquía directa de inferencia.
- *Pqsql*: repositorio que almacena datos en una base de datos PostgreSQL.
- *Mysql*: repositorio que almacena datos en una base de datos MySQL.
- *Remote*: repositorio que se utiliza como proxy (un programa o dispositivo que realiza una acción en representación de otro) para un repositorio que está en un servidor Sesame.

Además Sesame a la hora de almacenar crea dos índices por defecto, para que el acceso a disco sea más rápido. Estos índices son árboles de tipo B que utilizan una clave formada por cuatro campos que son sujeto, predicado, objeto y contexto. El orden de estos campos en la declaración del índice será relevante en la manera en que si el primer campo es el sujeto y realzamos una consulta con un sujeto específico esto se realizara de

forma más rápida que si nos encontramos en el índice el sujeto como tercer o cuarto campo.

Sesame crea 2 índices por defecto cuando insertamos en disco, son SPOC cuya clave es sujeto-predicado-objeto-contexto y POSC cuya clave es predicado-objeto-sujeto-contexto. Pero cuando se crea un repositorio (un native repository) se tiene la opción de declarar más índices usando el parámetro Triple indexes. Para ello bastara con poner las iníciales de los campos (sujeto, predicado, objeto, contexto) dentro del citado parámetro creándose el nuevo índice. Crear más índices mejorara el rendimiento de las consultas pero esto también traerá consigo un coste en el mantenimiento de los índices (a la hora de insertar nuevas sentencias…) y un gasto de memoria adicional por cada índice creado.

3.2.4 Comentarios modelo de datos

Como se ha visto en los puntos las principales diferencias que existen en los modelos de datos son el tipo de índices que se utiliza. En Sesame se utilizan árboles B mientras que en Oracle árboles B* (similares a los B+ que son los que almacenan la información en los nodos hoja) y por último Kowari y Mulgara que utilizan AVL.

Respecto a los índices también existe la diferencia de el número de índices que se van a crear en cada base de datos. En Kowari y en Mulgara este número viene fijado ya al crearse la base de datos y es de seis y es algo que no se puede modificar. Mientras tanto en Oracle y en Sesame si se puede crear el número y el tipo de índices que se crean necesarios. Una primera idea fue crear también 6 índices pero esto se descarto en seguida viendo la lentitud en la inserción en Oracle por lo que se pensó en reducir este número a dos (también se probo no creando ningún índice pero la lentitud en la inserción no era problema de los índices como en apartados posteriores se comentara) y en Sesame se hizo lo mismo que además son dos los índices que se crean por defecto. En las conclusiones se podrá ver cuánto es de necesario la utilización de más o menos índices en el propósito del proyecto.

Respecto a la forma de almacenar los datos en las bases de datos todas lo hacen mediante cuádruplas formadas por sujeto, predicado, objeto y un cuarto elemento que

Sesame, Kowari y Mulgara lo generan automáticamente (llamado contexto), mientras que en Oracle va ser un número secuencial que identificara a la tripleta.

3.3 Seguridad en los gestores RDF

En este apartado se analizaran los medios e infraestructuras para la seguridad que nos proporciona cada uno de los gestores RDF que se van a analizar en este proyecto.

Será necesario comentar que medios y que herramientas utilizan los gestores RDF con los que se trabajan para tratar los siguientes puntos en cuanto la seguridad:

- Identificación y autorización de usuarios.
- Accesibilidad: la información debe estar disponible.
- Integridad: Permite asegurar que los datos no han sido falseados.
- Prevenir escuchas (eavesdropping), la falsificación de la identidad del remitente (phishing) y mantener la integridad del mensaje.
- Uso de técnicas de cifrado, que nos proporcione una capa de transporte entre cliente y servidor.
- Uso de auditorias donde se registran las operaciones realizadas por un usuario.
- Backups:
    - Backup Lógico: Este backup se refiere a hacer una copia lógica de las estructuras e información que están dentro de la base de datos. Se llama copia lógica porque lo que se copia es prácticamente la definición en un script de los objetos para luego poder crearlos cuando algo falle.
    - Backup Físico: se realizan copias de los ficheros físicos que componen la base de datos.

3.3.1 Oracle

Los métodos con los que cuenta Oracle son:

A. Métodos de autentificación de Oracle:

Oracle dispone de varios métodos de autentificación:

- VPD (Virtual private databases): esta tecnología puede restringir el acceso a determinadas filas de una determinada tabla.
- Seguridad basada en roles (se verá más adelante)
- Grant-execute security: aplicación de privilegios sobre procedimientos, que hacen que un usuario mediante un procedimiento tengo acceso a la base de datos pero solo en el ámbito del procedimiento.
- Servidores de autentificación segura para usuarios externos.
- Seguridad de acceso a los puertos: las aplicaciones de Oracle están prefijadas para escuchar por un puerto específico del servidor. Oracle Web Listener puede ser configurado para restringir el acceso.

B. Checklist:

Oracle proporciona un checklist de seguridad para la auditoria de la base de datos. Un checklist nos indica una serie de elementos que deben ser revisados para determinar si la base de datos que se analiza, cumple con las características de Seguridad aconsejadas por Oracle en este caso [27].

C. Cifrado:

Oracle da la posibilidad de que ciertos datos sensibles como pueden ser el número de las tarjetas de crédito, número de la seguridad social… puedan ser cifrados, con la utilización de unos simples comandos (Oracle Transparent Data Encryption).

Este cifrado no implica cambiar la sintaxis de SQL a la hora de insertar datos, sino que los datos son automáticamente cifrados por Oracle antes de que estos se escriban en el disco. Lo mismo pasa a la hora de una consulta, los datos son devueltos ya descifrados para que no exista ningún tipo de problema. Oracle Transparent Data Encryption es transparente a todas las aplicaciones que utilizan la capa de SQL.

Además esta encriptación de los datos no crea problemas con los índices que tengamos creados ya que Oracle Transparent Data Encryption en cripta el valor del índice saciado con una tabla, por lo que el rendimiento no se debe ver afectado a la hora de las búsquedas.

En este documento se puede ver qué pasos seguir para el cifrado de algún elemento de una tabla que tengamos en nuestra base de datos [28].

D. Oracle Label Security

Es otra opción que da Oracle para agregar más protección a la información sensible de nuestra base de datos. Brinda capacidades de seguridad en múltiples niveles a fin de proteger el acceso a los datos hasta las filas individuales de tablas. (Rol de policía)

E. Roles

Los roles es una importante herramienta dentro de Oracle para manejar privilegios (ya sea privilegios para ver determinadas tablas, borrar, crear…), estos se podrán aplicar sobre usuarios o sobre otros roles ya creados. Podemos crear un modelo de seguridad basado en roles.

F. Auditorias

Oracle también da la posibilidad de hacer auditorias cuyo objetivo fundamental es obtener información de las operaciones que cada usuario realiza sobre los objetos de una base de datos.

Oracle soporta tres tipos generales de Auditoria:

- Por sentencia: las sentencias auditables son, por ejemplo:
- INDEX (Create, alter, drop.), NOT EXISTS (Todas las sentencias SQL que fallan al no existir el objeto referenciado), PROCEDURE (Create function, create pakage,…), PUBLIC SYNONYM (Create, drop), ROLE (Create, alter, drop).

- Por Privilegios o Autorizaciones: privilegios como ALTER DATABASE, AUDIT SYSTEM, ALTER, SYSTEM, CREATE DATABASE LINK, CREATE ROLE…

- Objetos de un sistema: se parece a la auditoria por sentencia SQL, pero en este caso, se especifica un esquema y un objeto, es decir, una tabla, una vista y una función, etc.

Además Oracle dispone de dos utilidades que tiene Oracle para las auditorias son:

- DBMS. FGA (“Fine Grain Auditing”): utilidad para auditar operaciones de acceso selectivo sobre conjunto de datos, incluidas todas las operaciones de DML (Select, insert, delete y update).
- Log Miner. Utilidad incluida en Oracle, para procesar el REDOLOG/ARCHIVE LOG (cambio contenido y estructuras).

G. Backups

Oracle proporciona un par de métodos para hacer backups que son mediante Export/Import (backups lógicos) y mediante la Recovery Manager (RMAN) (backups físicos).

Export/Import es una utilidad de Oracle para realizar backups lógicos de Oracle (y luego poderlos restaurar). Esto significa que copian el contenido de la BD pero sin almacenar la posición física de los datos. Para realizar estas operaciones la base de datos tiene que estar abierta. Para crear el fichero de backup se utiliza la utilidad Export y para importar el contenido o recuperar la base de datos se realiza Import. Este tipo de backup de utiliza en los siguientes casos:

- Para realizar backups de bases de datos (pequeñas/medianas bases de datos).

- Detectar alguna corrupción en la base de datos, puesto que al hacer el Export se lee toda la base de datos.
- Para migrar una base de datos a otro servidor.
- Para corregir “Row Migration & Row Chaining” (línea demasiado grande para caber en un bloque de la base de datos, filas que se trasladaron a otros bloques y se hicieron demasiado grandes para entrar en los bloques originales).

Vamos a explicar que es exactamente el RMAN. Este producto tal como su nombre más o menos indica se encarga de la gestión de backups y restauración de data files, archivelogs y control files, además de poder ser usado para la recuperación completa o incompleta de una Base de datos.

En la tabla 3-6 se muestran que backups se pueden y cuáles no se pueden hacer gracias a los métodos citados anteriormente:

Tabla 3-6: TiposBackups Sesame

Tipo de backup	RMAN	Export
Backup en frio	Soportado - Requiere que la instancia esté montada	NO soportado
Backup en caliente	Soportado en forma automática	Requiere Undo para mantener consistencia
Backup Incremental	Soportado	NO soportado
Detección de bloques corruptos	Soportado. Se visualiza con las vistas V$BACKUP_CORRUPTION	Soportado. Se visualiza en el Log
Catálogo de backups	Soportado	NO soportado
Resguardo de archivos de inicio	Soportado	NO soportado
Comandos independientes S.O.	Soportado	Soportado

3.3.2 Kowari y Mulgara

Kowari y Mulgara no proporcionan ninguna infraestructura de seguridad salvo la posibilidad de la realización de backups mediante la Shell de ITQL. Mulgara y Kowari tienen la posibilidad de realizar backups de servidores enteros o de un modelo específico dentro de un servidor, la sintaxis básica para esta operación es:

backup server[#model] to [local|remote] file;

Donde:

- Serve: es la URI del servidor al que se le va a hacer el backup.
- #model : es opcional y se añade cuando en lugar de hacer un backup del servidor se hace un backup de un modelo concreto.
- local or remote: indica si el backup va ser guardado en local o en remoto, si no se especifica por defecto es en remoto.

- File: es la Uri del fichero donde se va a guardar el backup.

Si el backups es de un modelo, este es escrito en formato RDF, mientras que si es de un servidor este es escrito en formato N3 comprimido en gzip.

Se ha realizado una prueba en Mulgara y Kowari y estos son los resultados:

- Backup del servidor:

Mulgara:

Results: (1 query, 1.205,453 seconds) **Query Executed:**
backup <rmi://HP21997209933/server1> to
<file:/D:/Mulgara/servidorbackup.gz>;

Result Message: Successfully backed up rmi://HP21997209933/server1 to file:/D:/Mulgara/servidorbackup.gz.

Este archivo ocupa en disco en disco 126 MB.

Kowari:

Results: (1 query, 1.595,875 seconds) **Query Executed:**
backup <rmi://HP21997209933/server1> to
<file:/C:/kowari/servidorbackup.gz>;

Result Message: Successfully backed up rmi://HP21997209933/server1 to file:/C:/kowari/servidorbackup.gz.
Este archivo ocupa en disco 118 MB.

Para restaurar un backup del servidor se utiliza el comando “restore” mientras que si es de un modelo se utilizara el comando de carga normal “load” .

Para disponer de infraestructura de seguridad existe la versión comercial de Kowari que proporciona la capa de transporte segura (SSL / TLS), y almacenar el nivel de autenticación y autorización. SSL proporciona autenticación y privacidad de la información entre extremos sobre Internet mediante el uso de criptografía. Habitualmente, sólo el servidor es autenticado (es decir, se garantiza su identidad) mientras que el cliente se mantiene sin autenticar; la autenticación mutua requiere un despliegue de infraestructura de claves públicas (o PKI) para los clientes. Los protocolos permiten a las aplicaciones cliente-servidor comunicarse de una forma diseñada para prevenir escuchas (eavesdropping), la falsificación de la identidad del remitente (phishing) y mantener la integridad del mensaje.

SSL implica una serie de fases básicas:

- Negociar entre las partes el algoritmo que se usará en la comunicación
- Intercambio de claves públicas y autenticación basada en certificados digitales
- Cifrado del tráfico basado en cifrado simétrico

3.3.3 Sesame

Básicamente Sesame para el tema de seguridad proporciona dos opciones: la creación de usuarios y creación de contraseñas. Se pueden crear usuarios dando les permisos o no, tanto para operaciones de escritura como de lectura. Además se pueden crear contraseñas para el acceso a los servidores.

Existe otra opción que es exportar los datos que se tienen en la base de datos de manera que se haga una copia de seguridad de estos. Para ello utilizando Workbench se puede exportar los datos que se tienen en un repositorio en distintos formatos, como se muestra en la ilustración 3-9.

Ilustración 3-9: Exportación Sesame

Estos formatos de exportación son:

- N3: es una notación independiente de RDF, que representa las tripletas (como sujeto, predicado y objeto, las URIs se escriben entre ángulos y no existe una sintaxis específica para nodos en blanco. Un ejemplo de esta notación sería:

  <urn:isbn:0-345-67890-1> <http://purl.org/dc/elements/1.1/creator> autor .

- N-triples: es un subconjunto de la notación N3, donde se separan los elementos con un espacio o un carácter TAB, hay solo una tripleta por línea y las que empiezan por # son comentarios. También destacar que los nodos en blanco empiezan por _: seguido del nombre.

  # Mi libro en RDF <urn:isbn:0-345-67890-1> dc:creator _:autor .
  _:autor uns-dcic-prf:nombre "Pablo" .
  _:autor uns-dcic-prf:apellido "Fillottrani" .

- RDF/XML: un documento RDF/XML consiste de una especificación del elemento rdf:RDF, puede ser parte de un documento XML más grande, o ser el elemento raíz del documento. Este tipo de elementos contiene una serie de elementos rdf:Description, cada uno de es la descripción de un recurso. Si tiene

un atributo rdf:about entonces ése es el URI del recurso; si no lo tiene se trata de un nodo en blanco.

El contenido de un elemento rdf:Description, está formado por una serie de sentencias de propiedades, cada sentencia de propiedad engloba un par predicado/objeto: el predicado corresponde con el nombre del elemento y el objeto con su contenido. Dado que los predicados se identifican con URIrefs, es necesario usar espacios de nombres para crear nombres válidos en XML, si el contenido es texto el objeto representado es un literal y si el contenido es vacío entonces el objeto es otro recurso, identificado mediante el valor del atributo rdf:resource. Un ejemplo:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:uns-dcic-prf="http://www.cs.uns.edu.ar/~prf/#"
xmlns:dc="http://purl.org/dc/elements/1.1/">
<rdf:Description rdf:about="urn:isbn:0-345-67890-1">
<dc:creator>
<uns-dcic-prf:nombre>Pablo</uns-dcic-prf:nombre>
<uns-dcic-prf:apellido>Fillottrani</uns-dcic-prf:apellido>
</dc:creator>
</rdf:Description>
</rdf:RDF>
```

- TriX: este formato contiene una estructura más controlada y es más útil para cuando luego se quiere trabajar con otras tecnologías XML. Contiene información de contexto (llamada grafo).

- TriG: es una versión de TriX basada en esto. Contiene lo mismo.[33]

- Turtle: otro refinamiento (subclase) de N3 que probé un mecanismo para manejo de espacios de nombres, permite abreviaturas para triplas con el mismo sujeto, introduce también abreviaturas para colecciones, se puede agrupar triplas con el mismo sujeto, los nodos en blanco como objeto se notan con [] y los nodos en blanco como sujeto se notan incluyendo entre los corchetes el predicado y el objeto. Un par de ejemplos serian [29]:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

dc:creator [ uns-dcic-prf:nombre "Pablo" ; uns-dcic-prf:apellido
"Fillottrani" ] .
```

Si se utiliza Sesame como Base de datos RDF no hay mas medidas de seguridad que las citadas anteriormente, solo si Sesame es utilizado junto con otra base de datos pueden aplicarse los aspectos de seguridad que incorporen estas bases de datos.

Para más datos respecto a la seguridad se tendrá que adentrar en Apache [16] que es el contenedor Java donde va ir Sesame, pero como esto no incumbe al proyecto solo se comentara que el cifrado que usa Apache es SSL (Secure Sockets Layer [19]).

3.3.4 Conclusiones seguridad

En este apartado existe una gran diferencia entre Oracle y el resto de gestores RDF analizados debido principalmente a que Oracle en si es un sistema gestor de bases de datos relacional que junto a una serie de procedimientos puede almacenar y trabajar con tripletas RDF, mientras que el resto son almacenes específicos para tripletas RDF que a su vez soportan distintos gestores de bases de datos.

Así que la gran ventaja de trabajar con Oracle es que los métodos de seguridad ya viene integrados en él al ser un gestor de bases de datos, sin embargo, si utilizamos Mulgara o Sesame se tendrá que estudiar qué sistemas gestores bases de datos soportan (como PostgreSQL, MySql,…) y ver qué medios de seguridad pueden proporcionar estos sistemas, ya que Mulgara y Sesame por si solos poco aparte de la realización de backups o exportación de datos nos pueden aportar en cuanto a métodos y medidas de seguridad se refiere.

Tabla 3-7: Resumen Seguridad en los gestores RDF

	Oracle	Kowari	Mulgara	Sesame
Backups	-Backups lógicos: mediante los comandos Export/Import -Backups físicos: mediante la Recovery Manager(RMAN)	-Backup del servidor -Backup del modelo de datos	-Backup del servidor: copia de los datos en formato N3 comprimidos en Gzip. -Backup del modelo de datos: copia en formato RDF	-Extract data: extracciones de los datos que se tienen almacenados en un repositorio en distintos formatos.
Métodos de autentificación	-Creación de roles -Privilegios sobre procedimientos - Servidores de autentificación segura para usuarios externos -Seguridad de acceso a puertos	No dispone	No dispone	-Creación de usuarios -Contraseñas en servidores
Cifrado de datos	Oracle Transparent Data Encryption	SSL (Secure Sockets Layer), en su versión de pago	No dispone	SSL (Secure Sockets Layer), a través de Apache

3.4 Lenguajes de consulta

EN este apartado se van a estudiar los lenguajes de consulta utilizados en cada uno de los gestores RDF. Los lenguajes que van a permitir realizar consultas sobre los gestores RDF y recuperación de la información que se solicite de estos. Sera el método de comunicación con el gestor.

Cada uno de los gestores que se utilizaran en este proyecto posee un lenguaje de consulta y son:

- Oracle→SQL + función SDO_RDF_Match
- Kowari→ITQL

- Virtuoso→ SPARQL

3.4.1 SQL (Oracle)

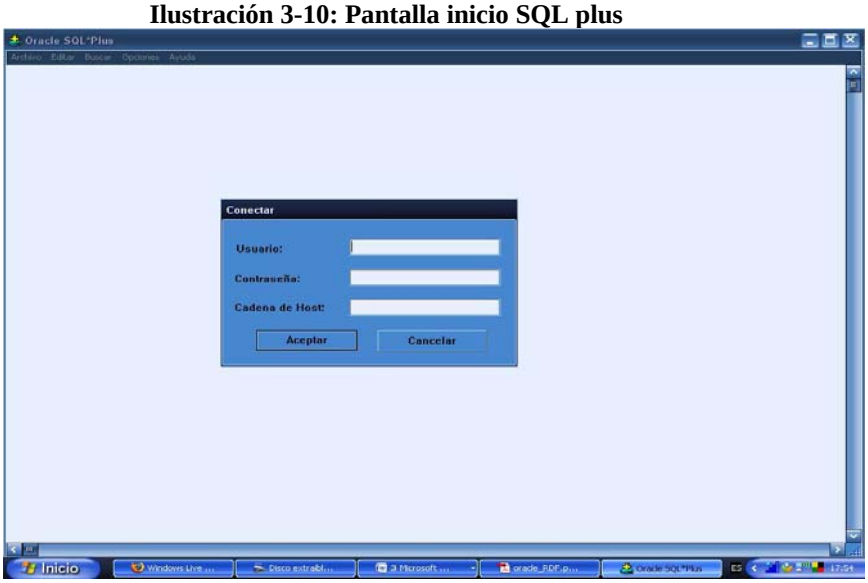
El lenguaje de consulta que se utiliza en Oracle es SQL Plus + la función SDO_RDF_Match, para así poder trabajar con las tripletas RDF dentro de Oracle.

Antes de nada, decir que SQL es Lenguaje de Consultas Estructurado, que permite la comunicación con el Sistema Gestor de Bases de Datos, que en nuestro caso es Oracle.

Las sentencias SQL pertenecen a dos categorías principales: Lenguaje de Definición de Datos, DDL y Lenguaje de Manipulación de Datos, DML. Estos dos lenguajes no son lenguajes en sí mismos, sino que es una forma de clasificar las sentencias de lenguaje SQL en función de su cometido. La diferencia principal reside en que el DDL crea objetos en la base de datos y sus efectos se pueden ver en el diccionario de la base de datos (a este grupo pertenecen sentencias como create table, create index…); mientras que el DML es el que permite consultar, insertar, modificar y eliminar la información almacenada en los objetos de la base de datos (a este grupo pertenecen sentencias como insert, delete, select...) [11].

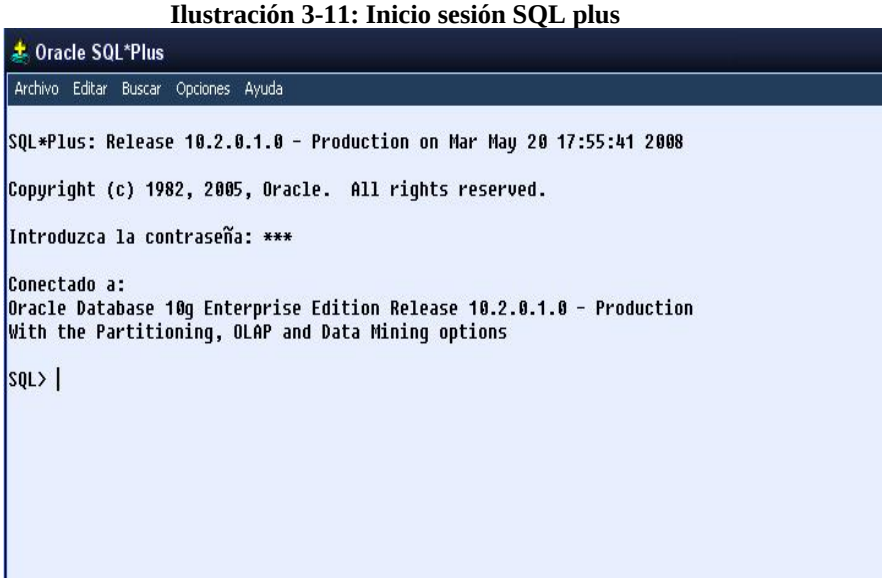
Oracle dispone de una herramienta que reconoce y envía sentencias SQL al servidor Oracle para su ejecución que es SQL Plus.

Ilustración 3-10: Pantalla inicio SQL plus



La ilustración 3-10 muestra la ventana de conexión, donde se introduce el usuario y la contraseña, y la base de datos a la que se quiere conectar.

Ilustración 3-11: Inicio sesión SQL plus



*Una vez iniciada la conexión podremos comunicarnos y operar con la base de datos mediante comandos SQL (ver ilustración 3-11)

Pero para el propósito del proyecto es necesario trabajar con tripletas RDF y para ello Oracle dispone de dos paquetes que son los que nos van a permitir trabajar con RDF y son los que vamos a usar:

1. SDO_RDF package: este paquete contiene funciones y procedimientos que nos permiten trabajar con RDF en Oracle. Los procedimientos y funciones más importantes son:

- SDO_RDF.CREATE_RDF_MODEL: crea un modelo RDF.
- SDO_RDF.DROP_RDF_MODEL: borra un modelo RDF.
- SDO_RDF.ADD_NAMESPACES: añade hasta 3 espacios de nombres.
- SDO_RDF.IS_REIFIED_QUAD: sirve para utilizar una sentencia (una tripleta ya añadida a la base de datos) como sujeto u objeto en otra sentencia.
- SDO_RDF.CREATE_RDF_NETWORK: crea el soporte RDF necesario a la base de datos.
- SDO_RDF.GET_MODEL_ID: devuelve el identificador del modelo.

2. SDO_RDF_INFERENCE package: contiene subprogramas (procedimientos y funciones) que nos permiten usar las capacidades de inferencia de RDF en una base de datos de Oracle, ya sean la creación de bases de reglas, índices….Dentro de este paquete tenemos:

- SDO_RDF_INFERENCE.CREATE_RULEBASE: crea una base de reglas. Estas reglase se usan para definir una ontología. Un ejemplo concreto de regla en Oracle sería este:

  EXECUTE _RDF_INFERENCE:CREATE_RULEBASE (‘family_rb’)
  INSERT INTO mdsys.rdfr_family_rb Values (‘granparent rule’ ‘(?x: parentOf ?y) (?y parent Of ?z), NULL, ‘(?x: grandParentOf ?z),SDO_RDF_Aliases(SDO_RDF_Alias(“,’http://www.example.org/family/’)))M

  Esta regla dice que si una persona es padre de un niño, el cual es padre de otro niño, esta persona es abuelo del hijo de este.

- SDO_RDF_INFERENCE.DROP_RULEBASE: borra una base de reglas.
- SDO_RDF_INFERENCE.CREATE_RULE_INDEX: crea un índice de reglas basado en uno o varios o modelos o en una o varias bases de reglas.
- SDO_RDF_INFERENCE.DROP_RULE_INDEX: borra un índice de reglas.
- SDO_RDF_INFERENCE.LOOKUP_RULE_INDEX: devuelve el nombre de un índice de reglas basado en un determinado modelo o en una determinada base de reglas.

  Un ejemplo concreto de regla en Oracle sería este:

  EXECUTE _RDF_INFERENCE:CREATE_RULEBASE (‘family_rb’)

  INSERT INTO mdsys.rdfr_family_rb Values (‘granparent rule’ ‘(?x: parentOf ?y) (?y parent Of ?z), NULL, ‘(?x: grandParentOf ?z),SDO_RDF_Aliases(SDO_RDF_Alias(“,’http://www.example.org/family/’)))M

Esta regla dice que si una persona es padre de un niño, el cual es padre de otro niño, esta persona es abuelo del hijo de este.

3.4.2 ITQL (Kowari y Mulgara)

Existen varios métodos de comunicación con nuestra base de datos (en Kowari o Mulgara) uno de ellos es RDQL que es un lenguaje de consulta RDF basado en *SquishQL*[21]. Con la idea de convertirse en un modelo de consulta orientado a datos por ser una aproximación más declarativa. Debido a esto, solo se pueden hacer consultas sobre la información que hay en el modelo, por lo que la inferencia o razonamiento no es posible.

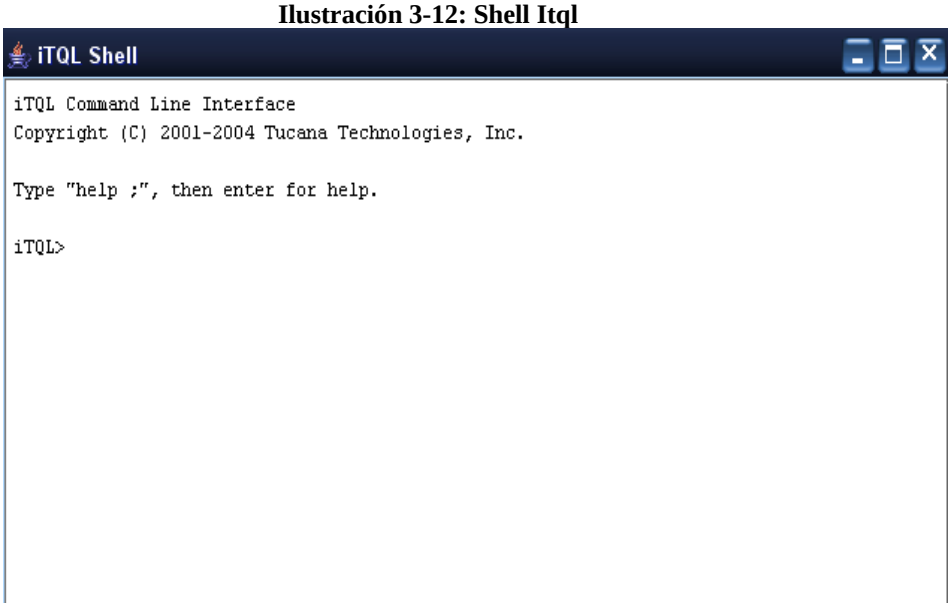
Como RDF provee una estructura de grafos, donde los nodos son recursos o Literales, RDQL permite especificar el patrón de la tripleta <sujeto, predicado, objeto> que queremos buscar en el grafo del modelo para poder recuperar cualquier parte de la tripleta, devolviendo todas las tripletas que cumplan el patrón que le pasamos.

Además de la desventaja de no permitir realizar ninguna inferencia, la utilización de filtros para obtener resultados es muy limitada. Las ventajas son la sencillez de manejo, ya que solo hace falta tener claro que tripleta <sujeto, predicado, objeto> se quiere preguntar, y otra de las ventajas son la facilidades de integración con Java.

Pero en nuestro caso vamos a utilizar el lenguaje ITQL, que nos permite algo más de flexibilidad que RDQL, como los límites en el tamaño de los resultados. ITQL es un lenguaje declarativo (son aquellos lenguajes de programación en los cuales se le indica a la computadora que es lo que se desea obtener o que es lo que se está buscando) basado en SQL y SquishSquish.

ITQL corre como un ejecutable aparte de la base de de datos y se puede usar tanto en remoto como en local. En la ilustración 3-12 se muestra la Shell de ITQL:

Ilustración 3-12: Shell Itql



ITQL permite crear modelos RDF, insertar y borrar datos de estos modelos y realizar consultas sobre ellos y permite cargar datos a nuestros modelos desde archivos externos. Además permite hacer back ups de los modelos o de la base de datos entera mientras este el servidor en funcionamiento, y ser usados los datos de los backups para restaurar la base de dato o nuestros modelo de ella si fuera necesario.

También dispone de otras características como:

- Nombrar a modelos RDF por URI
- Views
- Modelos de varios tipos de datos
- Soporte de alias
- Migración de datos

Una de las mayores diferencias de ITQl respecto a otros lenguajes de consulta de RDF (como SPARQL que el lenguaje de consulta RDF propuesto por la W3C), es el número de funciones y posibilidades que ofrece ITQL. Por ejemplo se puede hacer referencia a modelos por URIs y referirnos a estas en la cláusula FROM de la consulta (estos son los denominados alias). También existe diferencia en cuanto a las sintaxis y algunos operadores, pero el equipo de Kowari (y ahora Mulgara) tienen la intención de adaptarse al estándar que se siga y ser un subconjunto de él.

A continuación se va a ver un poco más sobre cómo trabajar con ITQL y algo de su sintaxis. Para poder ejecutar la Shell de ITQl Kowari (lo mismo Mulgara) dispone de un archivo ejecutable JAR que contiene todas las librerías y archivos de configuración necesarios para ejecutar la Shell. La forma de ejecutarlo desde la línea de comandos sería:

$ java -jar itql-1.1.0.jar

Una vez que se está en la línea de comandos de la Shell se puede ejecutar comandos ITQl para poder realizar consultas y actualizaciones de la base de datos, todos estos comandos terminaran por ";". Vamos a ver algunos de los comandos utilizados en ITQL:

- Alias: sirve para referirnos a un recurso de una forma más abreviada por ejemplo dentro de una consulta y su sintaxis es: **alias recurso as namespace**;
- Backup: realiza una copia de seguridad de un servidor (con todos los modelos que contiene) o de un modelo concreto, indicando donde queremos almacenar esa copia de seguridad. Decir que las copias de seguridad de un solo modelo están escritas en formato RDF, mientras que las copias de seguridad de un servidor entero están en el formato de compresión gzip. La sintaxis general es la siguiente:

  backup server[#model] to [local|remote] file

- Commit: confirmación en una transacción. Sintaxis:

  Commit;

- Create: creación de un modelo en un servidor existente de Kowari. Sintaxis:

  create model [type];

- Insert: inserta una sentencia (tripleta RDF) en un modelo ya creado. Sintaxis:

  insert <ns:subject> <ns:predicate> 'object' into
  <rmi://mysite.com/server1#model>;

- Delete: borra una sentencia de un modelo. Sintaxis:

  delete <ns:subject> <ns:predicate> 'object' from <rmi://mysite.com/server1#model1>;

- Drop: borrado de un modelo en un servidor existente de Kowari. Sintaxis:

  drop model;

- Help: muestra la sintaxis de comandos básicos.

  Help;

- Load: carga dentro de un modelo ya creado sentencias RDF (tripletas) desde un archivo que las contiene. Sintaxis:

  load [local | remote] file into model;

- Restore: recupera una copia de seguridad que previamente se realizó con un backup. Sintaxis:

  restore server from [local|remote] file;

- Rollback: descarta todas las modificaciones que se producen en una transacción. Sintaxis:

  Rollback;

- Select: comando que he utilizado para realizar las consultas sobre la base de datos. En la implementación hablaremos de él.

- Export: Solo en Mulgara. Exporta un graph a RDF/XML

3.4.3 SeRQL (Sesame)

En Sesame se pueden utilizar como lenguajes de consulta SPARQL o SeRQL. Para la realización de este proyecto se ha trabajado con SeRQL que es el lenguaje de

consulta propio de Sesame ("Sesame RDF Query Language"). SeRQL es muy similar a SPARQL pero con otra sintaxis. En la actualidad se pueden encontrar muchas características de SeRQL en SPARQL, como anteriormente SeRQL las tomo de SPARQL. A continuación se muestran las principales características de SeRQL:

- URIs, literales y variables

Como ya se vio los lenguajes de consultas las URIs y los literales son parte fundamental de RDF. Pero además ahora se añaden a la lista las variables. Las variables son identificadas con nombres y estos tienen que empezar por una letra o "_". Ejemplos validos serian Var1, _var2… A la hora de elegir nombres para variables solo se tienen que tener cuidado en no elegir nombres que formen parte de las palabras reservadas propias de SeRQL, como pueden ser select, Where, from…Las URIs como ya se vio son una cadena corta de caracteres que identifica inequívocamente un recurso. En SeRQL nos encontramos con dos formas de escribir las URIs. La primera forma sería escribiendo la URI entera, para lo que se escribiría esta entera y entre "<" y ">", unos ejemplos serían <http://www.ejemplo.es/index.html> o <file:///C:\rdffiles\test.rdf>. La otra forma consiste en crear una serie de prefijos para reducir el tamaño de las URIs, como están muchas veces tienen una parte común se pone esta parte común (namespace) como prefijo. Un ejemplo es jemplo:index.html.Por último tenemos los literales que en SeRQL van entre comillas que puede ir acompañado de una etiqueta de idioma (se denota con una "@" seguido del idioma) o de un tipo de dato (se denota con "^^" y la URI que identifica el tipo de dato). Ejemplos:

"book"
"book"@en,
"<book/>^^http://www.w3.org/1999/02/22-rdf-ayntaxis-ns#XMLLiteral

Por último también comentar que en SeRQL se tienen los nodos en blanco que son aquellos nodos que no están etiquetados con una URI o un literal.

- Path expressions

Las path expressions (representación de las tripletas RDF) ayudan a representar los gráficos RDF, para ser utilizados en las consultas. Si por ejemplo se tienen personas que trabajan en compañías que son compañías IT, las expresiones que se tendrían para esta representación serian:

Ilustración 3-13: Ejemplo path expresion

{Person} ex:worksFor {Company}
{Company} rdf:type {ex:ITCompany}

Los bordes y nodos en las path expressions pueden ser variables, URIs y literales. También se permite que los nodos puedan ser multi-valor ya que en alguna consulta pueden recibirse varios sujetos o objetos para un mismo predicado.

{sujeto1} {predicado1} {objeto1}
{sujeto1} {predicado1} {objeto2}
{sujeto1} {predicado1} {objeto3}

También en SeRQl si por ejemplo se quiere conocer los datos de un empleado en una empresa, y hay algún dato que no tiene porque existir para esa persona, SeRQL permite declararlo como opcional, para ello lo ponemos entre "[“y “]":

{Persona} Ex:nombre {nombre}
Ex:edad {edad}
[Ex: email {dirección email}]

Si no se hiciera así las personas que no tuvieran email no serian devueltas en la consulta.

- Consultas Select y Construct

  SeRQl maneja dos tipos de consultas, las select y las construct:

  - 3.4.3.1. Select

Estas consultas devuelven el valor o conjunto de valores sobre la variable o variables sobre los que se hace la consulta. En la clausula select se especifica la variable o variables de las cual se quiere obtener el valor. En la clausula from indicamos los path que son relevantes para la consulta. La clausula where es opcional y sirve para añadir alguna constricción (de tipo booleano) a la consulta.

Si en la consulta se quieren utilizar prefijos, estos se declaran mediante la clausula using namespace. Por último también está la clausula limit que delimita el número de respuestas a una consulta.

Un ejemplo de una consulta tipo select sería:

SELECT *FROM {<http://dbpedia.org/resource/Cindy_Sherman>}
http://dbpedia.org/property/training {A}

En esta consulta se pregunta donde estudió Cindy Sherman.

- 3.4.3.2. Construct

Este tipo de consultas nos devuelven un RDF graphs como un conjunto de tripletas.

Por ejemplo si se tiene una relación en la que un artista es un pintor que a su vez ha pintado una obra, se puede construir la propiedad por la cual un artista que tiene una obra pintada es un pintor. Un ejemplo sería:

CONSTRUCT DISTINCT {Artist} rdf:type {ex:Painter}
FROM {Artist} rdf:type {ex:Artist};
ex:hasCreated {} rdf:type {ex:Painting}
USING NAMESPACE ex = http://example.org/things#

- 3.4.3.3. ASK:

Devuelve un boolean indicando si los patrones de la consulta coinciden o no.

- 3.4.3.4. DESCRIBE:

Devuelve un grafo RDF que describe los recursos encontrados.

4. RESULTADO DE LOS EXPERIMENTOS

En este apartado del proyecto se va llevar a cabo la parte experimental consiste en la creación de repositorios en los 4 gestores RDF para la posterior inserción de tripletas RDF sobre ellos y la realización de diferentes consultas. Con este se pretende comparar el rendimiento durante estas operaciones en cada unos de los gestores para realizar una comparación entre ellos viendo las ventajas y desventajas que hay en el uso de uno otro.

Para ello es importante conocer las características del sistema sobre el que se va a montar los gestores RDF:

Sistema operativo: Microsoft Windows XP Profesional Version 2002, Sevice Pack 2.

Procesador: AMD Athlon 64 processor, 2.20 GHz.

Memoria: 1,00 GB de RAM, 80 GB de espacio en disco.

4.1 Implementación de gestores e inserción de datos

En este apartado se realiza una comparación entre las cuatro gestores que han sido llevadas a estudio analizando que tiempo se ha necesitado para insertar los datos con los que se trabajan en este proyecto, que uso de CPU se realiza durante esta operación y que medidas particulares a cada uno de los gestores RDF, se han tenido que tomar para poder realizar dichas inserciones.

Para la realización de este proyecto eran necesarias una serie de tripletas RDF con las que poder trabajar en los gestores RDF, estas tripletas han sido obtenidas de la página de Dbpedia [30]:

Donde se descarga el fichero Infoboxes.nt en el cual se encuentran unas 15 millones de tripletas RDF que serán insertadas en los cuatro gestores del proyecto.

Una vez bajado el fichero Infoboxes se ha procedido al análisis de este para poder ver el tipo de datos que hay en él y ver su estructura. Al ser Infoboxes un archivo

de más de 2 GB (exactamente 2.170.552.159 bytes) y ser imposible de abrir con cualquier editor de textos, se ha creado un programa que va dividir este fichero en otros más pequeños exactamente de un millón de tripletas cada uno, que ocuparan aproximadamente 130 megas y los cuales pueden ser abiertos y así analizarlos (que tipo de datos forman el fichero, su estructura…). Este programa se adjuntara con la memoria del proyecto.

Antes de empezar con el análisis es necesario tener en cuenta que cuando se realice la inserción de estos ficheros en la base de datos no todos las tripletas serán insertadas ya que dependiendo de la base de datos con la que estemos trabajando alguna tripleta no se insertara. Esto se puede ver más claro a través de un ejemplo: Oracle por ejemplo no acepta tripletas en las que vaya una fecha y esta no esté en el formato que corresponda como se puede ver en este caso:

<http://dbpedia.org/resource/%C3%9Eorsteinn_P%C3%A1lsson><http://dbpedia.org/property/termStart>"1987-07-08"^^<http://www.w3.org/2001/XMLSchema#Date>

Esta es una tripleta cuyo objeto es una fecha, esta fecha es un tipo de dato fecha que tiene un formato fijo (año-mes-día) si este formato no se cumple la tripleta no sería válida por lo tanto no se produciría su inserción en la base de datos. Es muy frecuente según se ha podido observar que por ejemplo un dato de tipo fecha no cumple el formato que debería y por ejemplo puede estar incompleto (falta el año), con lo que esa tripleta no sería insertada. Esto mismo no ocurre en el resto de gestores y las fechas son almacenadas aunque no tenga ese formato. También puede ser que una de las partes que forma la tripleta (objeto, sujeto o predicado) tenga un tamaño más largo de lo permitido y esta no pueda ser insertada dentro del repositorio, en Oracle por ejemplo el tamaño es un varchar2 (30.000 caracteres), para el resto de gestores RDF se verá en sus correspondientes apartados. Por lo que se ha visto al analizar los ficheros resultantes de dividir el fichero infoboxes, son pocos los objetos que superan esta restricción pero los que la superan no serán insertados. En los otros gestores este problema que se ha detectado en Oracle no ocurre con lo que los datos con más de 30.000 caracteres si son insertados.

Dentro de cada uno de los gestores se tendrá que crear un modelo o repositorio con las características adecuadas para poder insertar y trabajar con las tripletas RDF, cada una de ellas tendrá un método de inserción y una herramienta para el análisis:

Tabla 4-1: modos de inserción y herramientas para el análisis

	Oracle	Kowari	Mulgara	Sesame
Tablas, modelos y repositorios creados	-Creación tablespace -Crea tabla para insertar tripletas RDF -Modelo en base a la tabla anteriormente creada -Creación de índices	-Crear modelo donde se insertan las tripletas RDF -Creación automática de índices al crearse el modelo	-Crear modelo donde se insertan las tripletas RDF -Creación automática de índices al crearse el modelo	-Crear repositorio -Creación de índices (manual o automática con la creación del repositorio).
Método de inserción	-SQL Loader	- Kowari Viewer o consola ITQL	-Mulgara Viewer o consola ITQL	-Consola Sesame o Worckbench
Herramienta análisis rendimiento	- Oracle Enterprise manager	- System Explorer	-System Explorer	-System Explorer

En la Tabla 4-1 se muestran los pasos que tenemos que realizar para preparar las distintas bases de datos antes de insertar los datos, que herramientas se utilizan tanto para insertar los datos y medir el rendimiento de cada base de datos a la hora de realizar la inserción de los datos en las 4 bases de datos.

A continuación se muestra como se ha realizado la inserción de datos en los cuatro gestores por separada mostrando que pasos se han seguido y que problemas han

aparecido en cada uno de ellos y al final de este apartado se mostraran los resultados obtenidos en cada uno de ellos, comparándolos y analizando los resultados.

4.1.1 Implementación en Oracle:

a) Preparación de la base de datos

Antes de hablar del rendimiento que se ha obtenido en las pruebas que se han realizado en la base de datos Oracle con datos RDF, es necesario explicar cómo se ha implementado la base de datos en Oracle.

Una vez instalado Oracle Database 10g Release 2 se tiene que crear un espacio de trabajo lo suficientemente grande como para insertar las tripletas RDF que tenemos en el fichero Infoboxes.nt. Además se tiene que crear un modelo RDF y la tabla dentro de este modelo donde insertar las tripletas:

```
Create tablespace rdf_tblspace

datafile 'C: /oracle/product/10.2.0/oradata/ora/rdf_tblspace.dat' size 2048m reuse

autoextend on next 256m maxsize unlimited

segment space management auto;
```

Procedimiento que crea las tablas y objetos requeridos en Oracle para dar soporto al trabajo con tripletas RDF

```
execute sdo_rdf.create_rdf_network ('rdf_tblspace');
```

Se crea la tabla donde irán las tripletas del modelo

```
create table tabla_rdf_data (id number,triple sdo_rdf_triple_s);
```

Se crea el modelo en base a la tabla anteriormente creada

```
execute sdo_rdf.create_rdf_model('modelo_rdf', 'tabla_rdf_data', 'triple');
```

Ya por último queda crear los dos índices:

```
CREATE INDEX tabla_sub_idx ON tabla_rdf_data (triple.GET_SUBJECT());
```

```
CREATE INDEX tabla_prop_idx ON tabla_rdf_data
(triple.GET_PROPERTY());
```

b) Inserción de datos

Una vez creado el modelo RDF el siguiente paso es ya insertar el fichero Infoboxes, para ello se insertaran los 16 ficheros en los que anteriormente se dividió el fichero Infoboxes.

Para realizar la inserción de los 16 ficheros se utiliza SQL Loader que es una potente utilidad de importación de datos que posibilita la carga automática de datos externos (residentes en ficheros del sistema operativo) en tablas de la base de dato. SQL Loader utiliza dos ficheros el de datos y el de control [12]. El fichero de datos es el fichero con los datos que queremos insertar pero con la extensión .dat y el de control es un fichero con extensión .ctl que indica al programa SQL Loader donde están los datos a insertar, en que fichero puede almacenar los datos que no se inserten y cómo interpretar los datos que hay en el fichero .dat. Para ver esto de forma más clara a continuación se puede ver un ejemplo de lo que sería un fichero control:

```
LOAD DATA

INFILE 'D:\FP1.dat' BADFILE 'D:\b1.bad'

APPEND

INTO table tabla_rdf_data

(id SEQUENCE(MAX,1),

subj boundfiller char terminated by whitespace,

pred boundfiller char terminated by whitespace,

obj boundfiller char terminated by whitespace,

triple expression "sdo_rdf_triple_s('modelo_rdf',:SUBJ, :PRED, :OBJ)")
```

Infile es el fichero que vamos a insertar y *Badfile* es el fichero donde se van a grabar los datos que no se han podido insertar.

Además de estos dos ficheros en la llamada al programa se ha incluido otro con la extensión .log donde el programa me guarda datos sobre la inserción como pueden ser el tiempo que ha tardado, los datos que ha leído, que datos ha insertado...

c) Problemas durante la inserción y soluciones

El primer fichero a insertar se inserto correctamente y se continuo con la carga del segundo millón de la misma manera, pero esta carga tuvo que ser interrumpida porque la inserción era demasiado lenta (en 3 días solo se habían insertado 100000 tripletas del segundo millón a insertar). En un primer momento se pensó que esto podía ser debido al uso de índices en la base de datos y claro cuánto más datos se insertaba más difícil y más lento era crear dichos índices para la base de datos, por lo que se borraron los índices y se intentó insertar de nuevo el segundo millón de tripletas, pero el proceso seguía siendo igual de lento, y se descartó que fuera por esta causa.

Después de la consulta en varios foros de internet como el propio foro de Oracle y no obtener ninguna solución al problema, se comprobó que en los ficheros de datos que se estaban intentando insertar el objeto podía ser una URL o un literal entre comillas que hace referencia a un tipo de datos del esquema de datos RDF y pensamos que tal y como había declarado el fichero de control, esto podía ser la causa de la ralentización a la hora de la inserción de los datos. La solución a este problema ha sido insertar por una parte las tripletas cuyo objeto es una URL y las tripletas cuyo objeto es un literal, con lo que los ficheros de un millón de datos han quedado divididos en dos ficheros uno con las tripletas que el objeto es una URL y otro con las que es un literal. Y con esta solución se ha conseguido insertar el fichero **Infoboxes** aunque el tiempo ha sido muy elevado. Esto ha podido ocurrir al estar los dos tipos de formatos de objeto en ficheros distintos las operaciones que realiza internas Oracle para la inserción de los datos y actualización de los índices son menos complejas y por eso ahora se pude realizar la inserción. De todas formas estos problemas pueden tener su origen en la arquitectura del sistema como ya se comento en el apartado correspondiente.

4.1.2 Implementación en Kowari

Para trabajar dentro de Kowari se dispone de dos opciones mediante la consola iTQL o mediante la aplicación Kowari Viewer.

Kowari Viewer es una web basada en el motor de inferencia de Kowari que nos permite ejecutar comandos ITQL y ver los resultados en formato HTML. Para poder ejecutar esta herramienta lo primero que tenemos que hacer es tener arrancado Kowari (un servidor de Kowari) y abriendo una ventana de nuestro explorador y introduciendo la Url **http://<host name>:<port>/webui/,** donde <host name> es el nombre del servidor y <port> es el puerto HTTP. EL puerto por defecto es el 8080. Si por ejemplo tenemos corriendo Kowari nuestro propio sistema en el puerto 8080, la URL sería http://localhost:8080/webui/. En la siguiente imagen se puede ver esta aplicación:

Ilustración 4-1: Kowari Viewer

Esta herramienta incluye los siguientes elementos [3]:

1 Model URI: nos permite especificar el nombre del modelo sobre el que vamos a trabajar.
2 Example Queries: es una lista despegable con 8 funciones ITQL.
3 Query text: muestra el resultado de cuál sería el código de una de las 8 funciones del despegable o podemos introducir nuestro propio código ITQl.
4 Botón Submit Query: ejecuta el comando ITQL.
5 Botón Clear Query: borra el campo Query text.

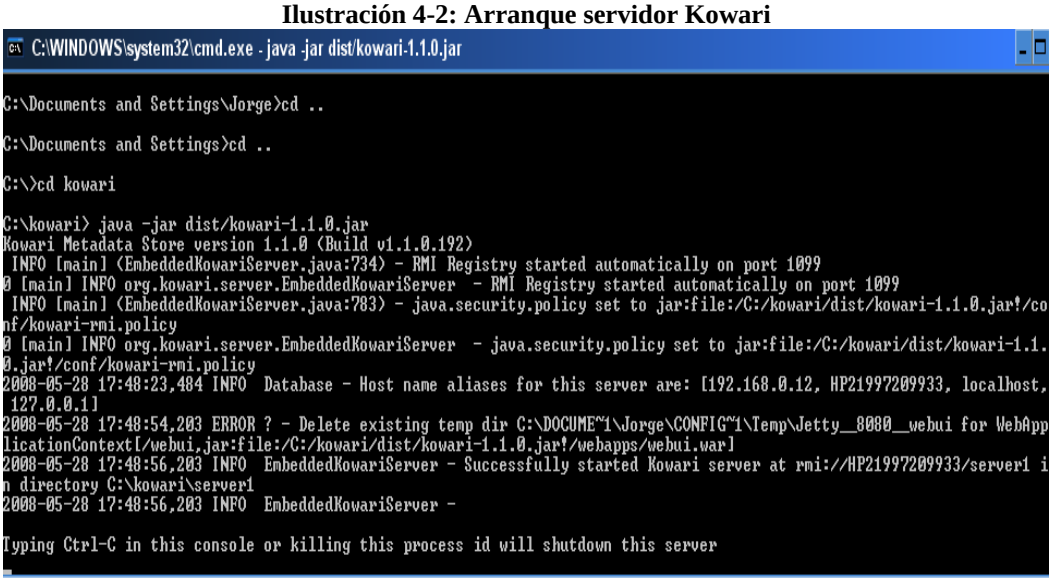
Si se quiere trabajar mediante la consola de Kowari se tendrá que arrancar el script de ITQL. Para ello se ejecuta el siguiente comando:

java -jar dist/itql-1.1.0.jar

Para trabajar con Kowari lo primero de todo es arrancar el servidor de Kowari. Para arrancar el servidor ejecutamos en una línea de comandos dentro de la carpeta donde hemos instalado Kowari:

java -jar dist/kowari-1.1.0.jar

Ilustración 4-2: Arranque servidor Kowari



a) Preparación de la base de datos

Al ser Kowari una base de datos específica para datos RDF en este caso solo se necesita crear un modelo dentro del servidor con el que se trabaja y al crear este modelo también se crean 6 índices como ya se comento en apartados anteriores. La sentencia de creación del modelo es:

create <rmi://HP21997209933/server1#modelo_rdf>;

b) Inserción

Una vez creado el modelo se inserta en dicho modelo los 16 ficheros en los que se divide Infoboxes a través de la consola ITQL de Kowari o en la aplicación Kowari Viewer. La sentencia de inserción seria:

Load <file:/d:/kowari/FP2.nt> into <rmi://HP21997209933/server1#modelo_rdf>;

c) Problemas durante la inserción y soluciones

Los dos primeros millones se insertaron sin ningún problema pero con la inserción del 3 fichero aparece el siguiente problema:

Ilustración 4-3: error carga

```
iTQL> load <file:/d:/kowari/FP5.nt> into <rmi://HP21997209933/server1#modelo_rdf>;
Could not load file:/d:/kowari/FP5.nt into rmi://HP21997209933/server1#modelo_rdf
Unable to load file:/d:/kowari/FP5.nt into rmi://HP21997209933/server1#modelo_rdf
Caused by: (QueryException) javax.transaction.RollbackException: null
Command execution time - 4684.875 seconds
```

Este problema lleva a la base de datos a realizar un rollback hasta el punto anterior junto antes de lanzar la última transacción (en este caso la consulta), por lo que no se realiza ninguna nueva inserción en la base de datos. Esta excepción no contiene la causa de porqué se produce el fracaso en la transacción, se ha llegado a la conclusión de que este problema puede ser debido a que al tener que hacer un gran número de operaciones para realizar la inserción de cada tripleta, actualizar los índices… llega un momento que el programa no puede con más (se queda sin menoría interna) y aborta la transacción.

Para solucionar este problema, se ha realizado la misma operación que en Oracle, hemos reducido el número de inserciones por transacción a la mitad y además se ha realizado la inserción de las tripletas cuyo objeto es una URL por lado y las tripletas cuyos objetos son literales por otra parte. De esta manera las inserciones se han producido sin problemas.

4.1.3Implementación en Mulgara

Los pasos a seguir en esta base de datos son los mismos que en Kowari (se tiene hasta el mismo problema), la única diferencia que se ha encontrado es que para realizar las inserciones de los ficheros se ha utilizado Mulgara Viewer, ya que aunque también podríamos haber utilizado la consola ITQL como se podía hacer en Kowari, esta no nos permite saber el tiempo que tarda la inserción en realizarse ya que la opción Set time on (nos indica el tiempo que tarda una operación en realizarse) no funciona y por lo tanto utilizando la consola es imposible conocer el tiempo que tarda en realizar la inserción, para conocer el tiempo que tarda en realizar la inserción esta se realiza mediante Mulgara Viewer que si lo muestra.

4.1.4 Implementación en Sesame

a) Preparación de la base de datos

Como ya se dijo en anteriores apartados Sesame trabaja con repositorios que son los contenedores para almacenar las tripletas RDF. Para poder crear un repositorio e insertar el fichero Infoboxes es utilizar la consola de Sesame [7].

Lo primero es conectarse a través de la consola del servidor de Sesame con el que se va a trabajar, en este caso será http://localhost:8080/openrdf-sesame.

El siguiente paso es crear el repositorio, para ello se utiliza el comando ***create*** acompañado del tipo de repositorio que vamos a crear, estos tipos se vieron en anteriores apartados. Para insertar los datos utilizados en este proyecto se ha creado un repositorio *Native-rdfs* con los índices spoc y posc que se crean por defecto al crear el repositorio.

create native-rdfs.
Please specify values for the following variables:

Repository ID [native]: rdf2

Repository title [Native store]: repositorio proyecto

Triple indexes [spoc,posc]:

Repository created

Una vez creado el repositorio antes de insertar datos es necesario abrir el repositorio para ello se utiliza el comando ***open*** seguido del nombre del repositorio: Open rdf2.

b) Inserción

El siguiente paso es comprobar si el fichero con los datos a insertar es correcto, que tipo de datos contiene y cuantos datos tiene ese fichero, para ello utilizamos el comando ***verify***:

Verify c:\sesame\fichero-insercion.nt

Una vez verificado el fichero ya se puede proceder a la inserción de los datos. Se puede ver esto en la siguiente captura:

Ilustración 4-4: Inserción datos Sesame

Viendo los resultados obtenidos con las anteriores bases de datos, se opto directamente por no insertar los 16 ficheros en que se divido Infoboxes si no utilizar los ficheros en que a su vez se dividió a estos 16 donde unos tienen las tripletas cuyo objeto es una URL por lado y los que tienen las tripletas cuyos objetos son literales.

También la carga de datos se puede hacer desde OpenRDF Workbench, para ello se abre la aplicación en el navegador y seleccionamos el servidor y a continuación el repositorio con el que queremos trabajar. Dentro del repositorio accediendo al apartado modify se podrán cargar los ficheros o datos que se desee.

c) Problemas y soluciones durante la inserción

El problema que aparece en Sesame es que el simple levantamiento del servidor consume casi la memoria entera del equipo (más de 800 Mb sobre el 1 GB de memoria que tiene el equipo) y una vez levantado el servidor la memoria utilizada es prácticamente la mitad de la que disponemos. Debido a esto en el momento de la inserción el uso de memoria se vuelve a elevar usándose gran parte de la memoria del equipo con lo que la memoria virtual de java se llena y se produce un error al proceder con la inserción.

Para encontrar el problema se miró en los archivos .log (Los archivos log son simples archivos de texto en los que Apache escribirá, en general, los accesos y los errores detectados) que almacena Apache Tomcat y se encontró el siguiente error java.lang.OutOfMemoryError: Java Heap Space. Este problema es debido a que Sesame utiliza mucha memoria a la hora de hacer las inserciones y actualizar los índices, además cuanto más datos tengamos insertados en el repositorio estas operaciones serán más complejas.

La solución este problemas es dar más memoria a Sesame, esto se realiza mediante los parámetros –Xms, –Xmx y –Xss cuando realizamos la llamada java para arrancar el servidor.-Xms es el tamaño de inicio de la memoria, -Xmx el tamaño máximo que podrá tener y –Xss fija el tamaño máximo de la pila para cualquier hilo [18]. Por tanto cuando se arranque el servidor se hará de la siguiente manera:

Ilustración 4-5: Arranque servidor Sesame

```
C:\Archivos de programa\Apache Software Foundation\Tomcat 5.5>java -Xms128m -Xmx
1024m -Xss768k -jar bootstrap.jar
```

Después de que este problema quedara solventado el resto de los ficheros fueron añadidos al repositorio (rdf2).

4.1.5 Conclusión:

Viendo los puntos anteriores se aprecia que para la operación de inserción en los cuatro gestores RDF cuando ya tienen una cantidad elevada de datos almacenados, necesitan de una cantidad importante de memoria para poder realizar dichas operaciones

de inserción (que además implican otras acciones como es la actualización de índices), siendo además la cantidad de memoria del equipo donde se han realizado los experimentos algo limitada, la inserción según aumentan los datos insertados dentro de los repositorios cada vez requiere más tiempo.

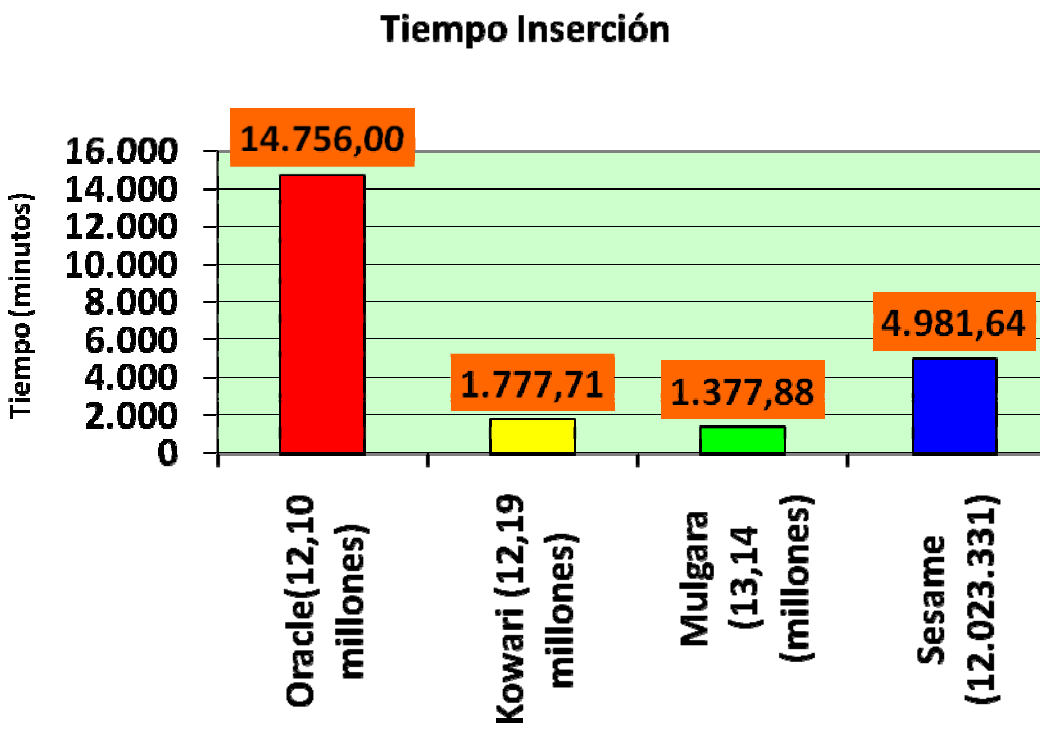
4.2 El coste de Inserción de datos (tiempo, espacio y memoria)

En este apartado se analizan los resultados obtenidos en el apartado anterior y se realizan las comparaciones entre los cuatro gestores RDF llevados a estudio.

4.2.1 Coste tiempo

En la primera gráfica se muestra los datos que se han conseguido insertar en cada una de las bases de datos y cuanto tiempo ha sido requerido para insertar estos datos:

Ilustración 4-6: Tiempo inserción datos



Como se aprecia en la gráfica Mulgara a pesar de ser la base de datos con más millones de tripletas insertadas es la que menos tiempo ha requerido para su inserción, sin embargo el tiempo que ha sido requerido para insertar los datos en Oracle es unas 11 veces mayor que Mulgara y Kowari.

Esta diferencia de más o menos un millón de datos de Mulgara respecto al resto de bases de datos se produce debido a que al principio de experimentar con Mulgara se

inserto un millón de datos, para hacer unas primeras pruebas con él y ver su funcionamiento. Después se insertaron el resto de datos de el fichero infoboxes como en las demás bases de datos, en este momento se podrían haber eliminado las tripletas anteriores o haber creado un nuevo repositorio para insertar el fichero infoboxes, pero se consideró interesante probar con un millón de datos más insertados para probar que sucedía respecto a rendimiento comparándolo principalmente con Kowari, pero también con el resto de repositorios.

Además comprobando el resultado de los experimentos este millón de datos más insertados no ha supuesto un problema a la hora de trabajar y no interfiere en los resultados obtenidos en el experimento a la hora de comparar los gestores RDF llevados a estudio.

Por otra parte se ha observado que mientras que en Mulgara y Kowari el tiempo de inserción aumentaba según aumentaban el número de datos que había en la base de datos, que es algo totalmente normal debido a que las operaciones para actualizar los índices cada vez son más complejas, en Oracle el tiempo ha permanecido constante y no es que haya ido incrementando según había más datos en la base de datos. Esto se puede ver de un modo más claro si se ven las gráficas de las inserciones de cada una de las bases de datos por separado:

Ilustración 4-7: Tiempo inserción Mulgara

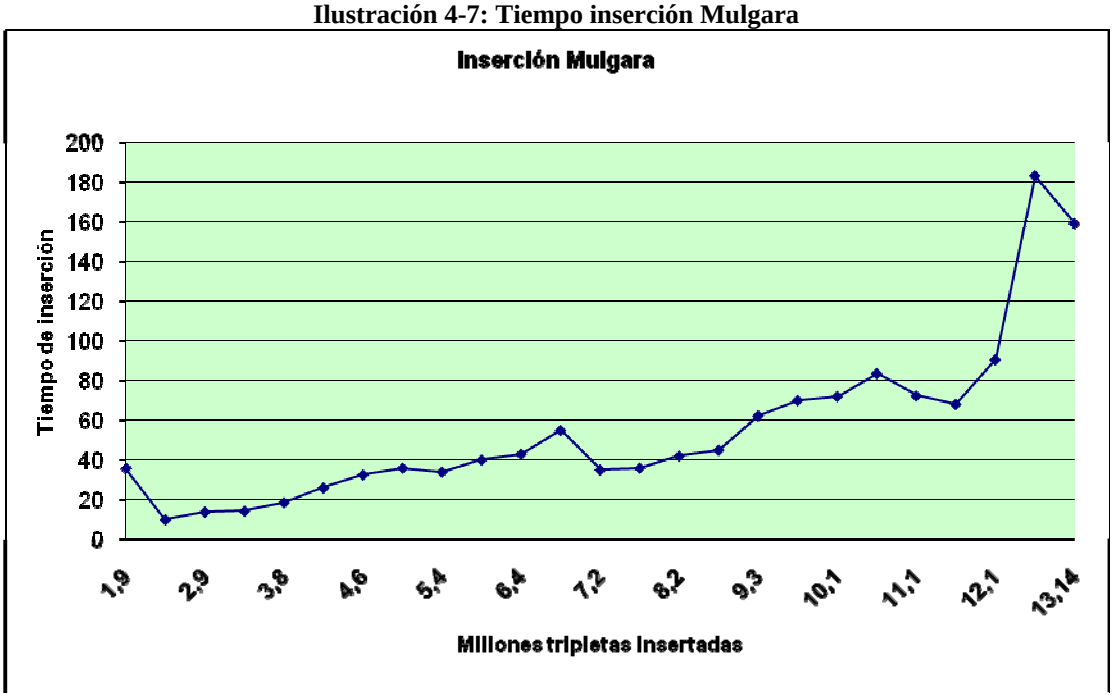
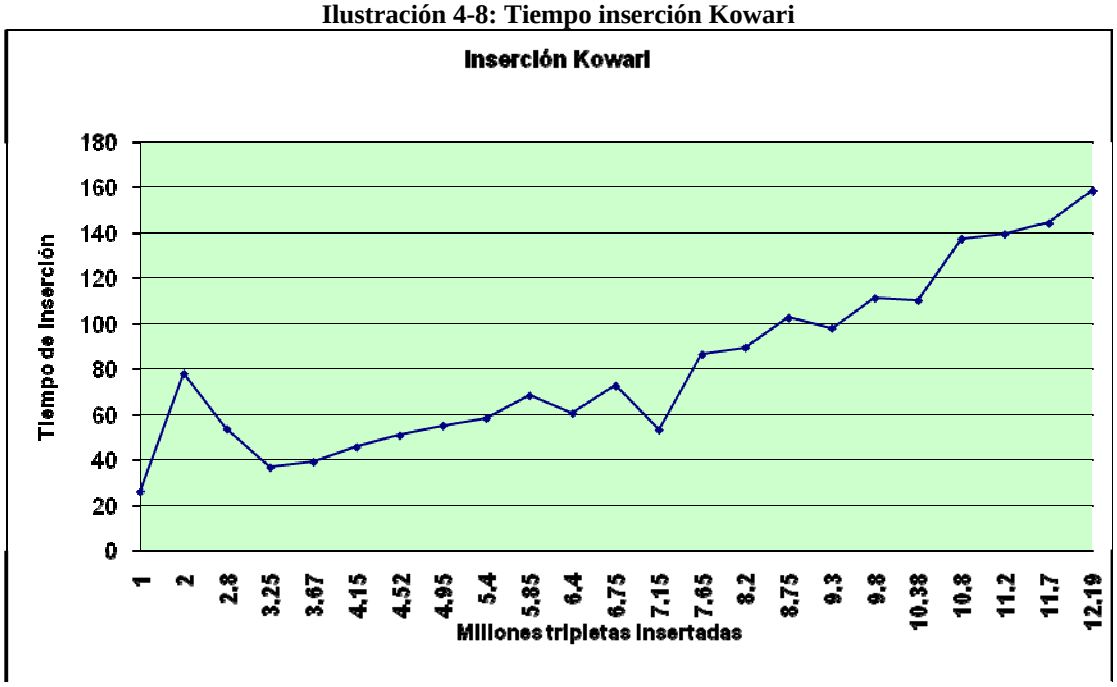


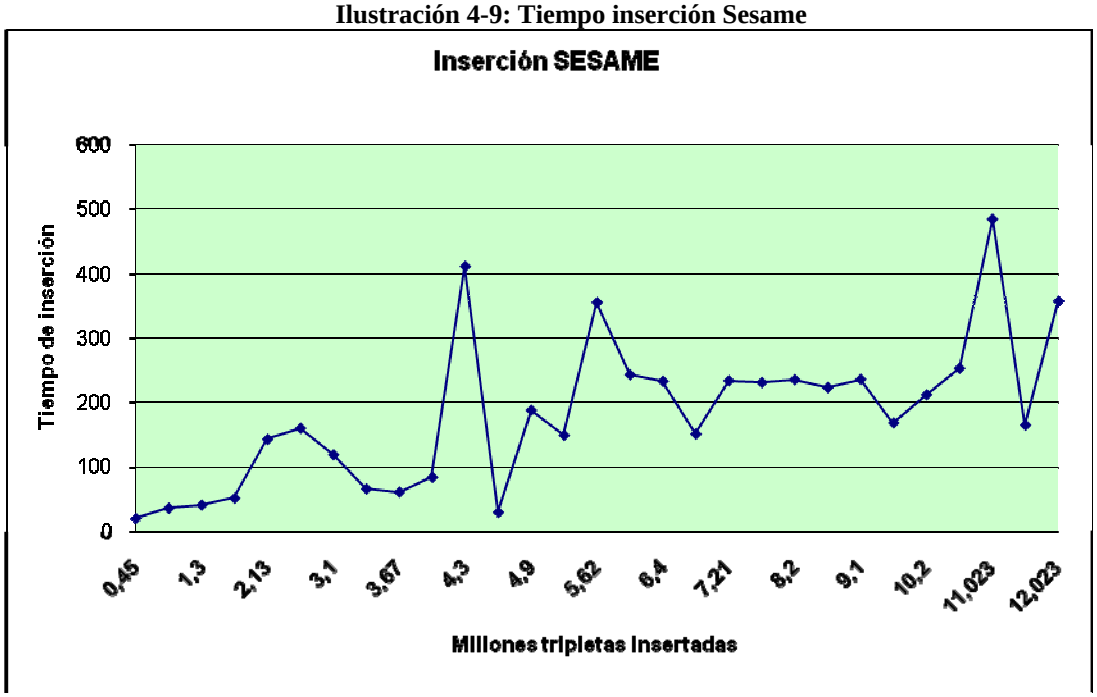
Ilustración 4-8: Tiempo inserción Kowari



Como se ve en estas dos gráficas tanto en Kowari como en Mulgara el tiempo se ha ido incrementando según aumentaba el número de tripletas insertadas en la base de datos. Esto tiene la fácil explicación de que las operaciones de inserción y actualización de índices, cada vez que hay más datos insertados, son más complejas y más costosas en cuanto a tiempo se refiere.

En Sesame sin embargo no se puede apreciar este incremento en el tiempo de forma tan claro y lineal como ha sucedido en Kowari y Mulgara. Lo que si se ha observado es que el tiempo medio en las inserciones si ha ido estabilizando en las últimas inserciones manteniéndose un tiempo de inserción más o menos constante, como se puede observar en el siguiente gráfico.

Ilustración 4-9: Tiempo inserción Sesame

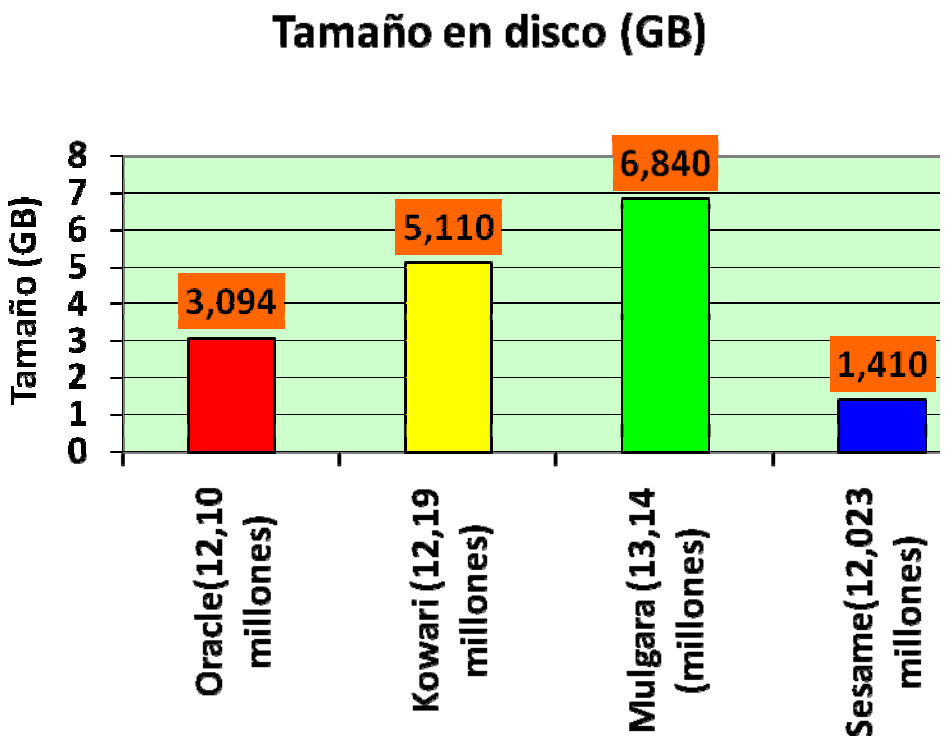


Por último Oracle ha tenido un tiempo de inserción prácticamente constante durante la inserción de los datos aunque estos han sido muy elevados (al ser constantes y no variar se ha decido no añadir ninguna gráfica para el tiempo de inserción en Oracle). Además por mucho que se ha investigado y preguntado en propios foros y ayudas de Oracle no se ha recibido contestación al porque de este elevado tiempo en la inserción. Una de las posibilidades que se barajó era la de la complejidad en la actualización de los índices pero se ha descartado por completo al crearse otro modelo para insertar las mismas tripletas pero sin índices y el tiempo siguió siendo muy elevado comparándolo con las otras bases de datos.

4.2.2 Coste espacio

También es importante conocer cuánto ocupan en disco cada una de las bases de datos una vez insertadas las tripletas:

Ilustración 4-10: Gráfica comparativa tamaño en disco



Para evaluar correctamente esta gráfica tenemos que considerar ciertos puntos:

- Oracle y Sesame utilizan dos índices (sujeto-predicado-objeto y predicado-objeto-sujeto) mientras Kowari y Mulgara por defecto vienen con 6 índices como ya se indico en apartados anteriores.

- Se consideró haber creado también 6 índices en Oracle y Sesame, pero a la vista en los primeros experimentos que las inserciones eran más lentas que en las otras dos bases de datos se decidió probar con dos índices (en este caso los dos que se crean por defecto en Sesame) y de esta manera se podrá también observar a la hora de analizar el rendimiento en las consultas que efecto tiene en estas el que existan 4 índices menos.

A la vista del gráfico anterior (Ilustración 4-10) Mulgara requiere bastante más espacio que en este caso la que requiere menos que es Sesame. Esto no es solo achacable a que Mulgara tiene 4 índices más que Sesame ya que el tamaño que ocupan los índices de Sesame en disco es de poco menos de 1GB (cada uno ocupa 500 Mb) si se crearan 4 índices más el tamaño seria poco mas de 2 GB con lo que aun se estaría lejos del tamaño requerido por Mulgara. La gran ventaja de Sesame sobre las otras

bases de datos en cuanto almacenamiento en disco se requiere es la compresión de los datos que en disco ocupan más o menos 400 MB.

Tabla 4-2: Espacio en disco

	Oracle	**Kowari**	**Mulgara**	**Sesame**
Tripletas insertadas	12,10 millones	12,19 millones	13,14 millones	12.023.331 millones
Espacio en disco	3,094 GB	5,11 GB	6,84 GB	1,41 GB
Espacio en disco detallado		XaStatementStore= 4, 31 GB XaNodePool=64 Kb XaStringPool=817 MB	XaStatementStore= 5,89 GB XaNodePool=24 MB XaStringPool=950 MB	triples-posc.dat= 569MB triples-spoc.dat= 540MB Resto espacio para almacenar las tripletas

La Tabla 4-2 muestra los datos insertados en cada una de las bases de datos, del espacio que ocupa cada una de las bases de datos en memoria y cuanto ocupan por separado los datos y los índices en cada una de ellas.

4.2.2 Coste memoria

Por último queda por hablar del uso de la CPU y de la memoria (RAM). En este apartado las peores colocadas son Sesame y sobre todo Oracle. Oracle durante las operaciones de inserción consume por valor medio 79,19 de memoria y un uso de la CPU que alcanza el 100%, para verlo a continuación se muestran unas capturas del Oracle Enterprise manager durante el periodo en el que se realizaron inserciones:

Ilustración 4-11: Uso medio de memoria durante inserción en Oracle

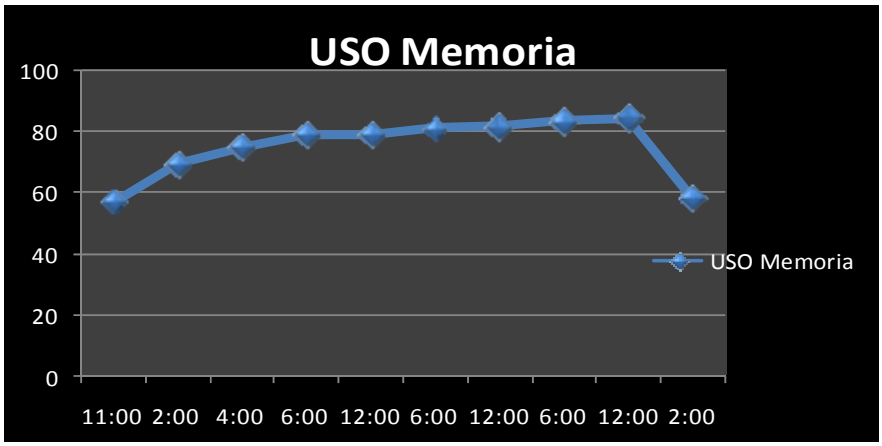
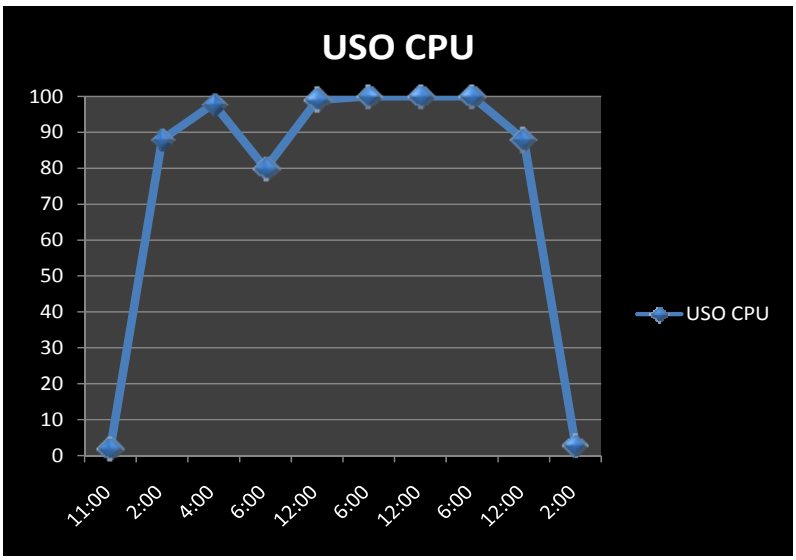


Ilustración 4-12: Uso CPU Oracle



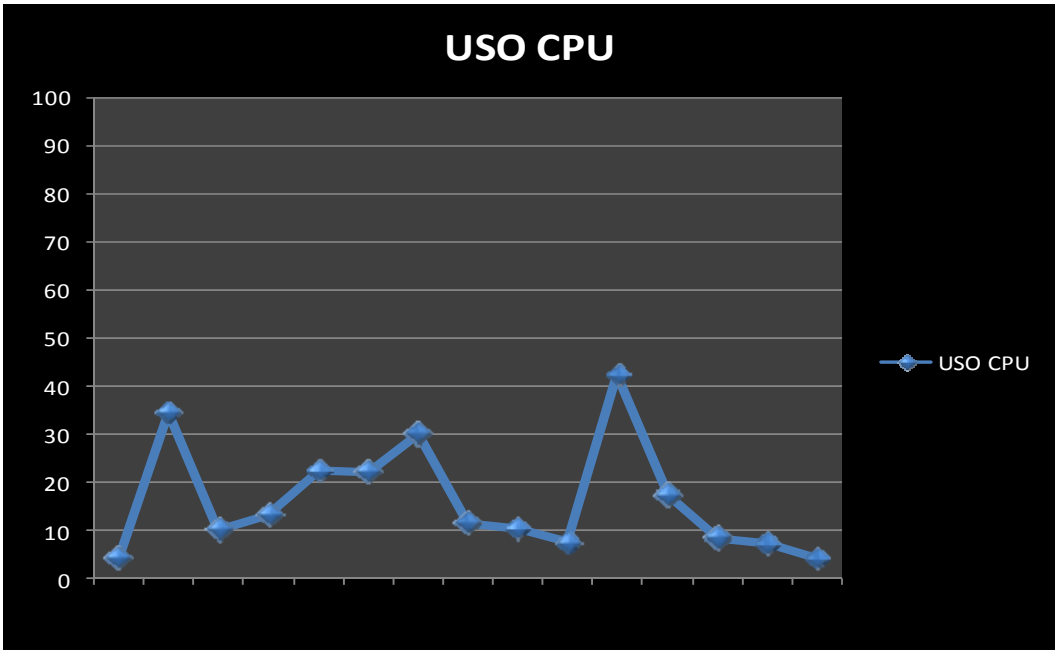
Por otro lado el uso de CPU y memoria en Sesame es también muy elevado. Para conseguir arrancar y poder hacer las inserciones como se dijo antes es necesario

reservar una gran cantidad de memoria para este proceso java. En la siguiente captura se puede ver el uso de memoria de Sesame durante el proceso de inserción de uno de los ficheros, en este caso es de 655 MB:

Ilustración 4-13: Uso memoria Sesame

En la siguiente gráfica también podemos ver el uso de CPU que realiza Sesame en las inserciones que en este caso no es muy elevado (durante este proceso el servidor está utilizando casi toda memoria), notar que la línea de color negro es el uso de CPU y la línea rosa son las operaciones de escritura y lectura por segundo:

Ilustración 4-14: Uso CPU Sesame



En la siguiente tabla se muestra el uso de CPU y de memoria de las 4 bases de datos:

Tabla 4-3: Uso memoria y CPU de los gestores RDF

	Oracle	**Kowari**	**Mulgara**	**Sesame**
Uso medio CPU	87,51 %	38 %	31 %	20 %
Uso medio de memoria	79,17 %	30 -35 %	Con el servidor arrancado =31 % (un uso de 310-320 MB) Realizando inserciones= 45%	Con el servidor arrancado =85 % (un uso de 820-850 MB) Realizando inserciones= 65% - 70%

Viendo estos resultados la ventaja en cuanto al uso de CPU de Kowari y Mulgara es clara ya que tanto el uso de memoria como de CPU no es muy elevado permitiendo un buen rendimiento aunque el equipo como es en este caso no sea muy potente.

4.2.4 *Conclusiones*

En este apartado se deben destacar a Sesame y Mulgara. Mulgara nos aporta frente al resto de bases de datos su mayor rapidez en la hora de las inserciones añadido el poco coste de memoria (RAM) requerida para llevar a cabo las inserciones, cosa que no sucede en Sesame o en Oracle donde la memoria (RAM) y uso de la CPU es muy elevado. El único punto flojo de Mulgara es que su ocupación en disco es más grande que las otras bases de datos.

Por otro lado está Sesame que como mayor ventaja nos encontramos con la compresión del tamaño de discos de los datos insertados y aunque tiempo de inserción es mayor que en Mulgara este problema es debido a que Sesame necesita usar más memoria (RAM) y el servidor consume mucha memoria, y al ser este un equipo cuya memoria está muy limitada (1GB) y el mero hecho de levantar el servidor para realizar las inserciones ya consume casi 800 MB (una cantidad muy elevada comparado con los 54 MB que utiliza Mulgara), esto dejaría de ser problema con un equipo algo más potente cosa totalmente asumible en la actualidad.

4.3 Pruebas de rendimiento en consultas

En este apartado se realizará la comparativa de rendimiento y funcionamiento de las consultas en las cuatro bases de datos. Para ello se han realizado 18 consultas en los respectivos lenguajes de consulta correspondientes a cada una de las bases de datos y se han ejecutado en las 4 bases de datos, tomando los resultados obtenidos en cada una de ellas y tomando el rendimiento durante la realización de dichas consultas (midiendo el tiempo de respuesta en milisegundos, el uso de CPU y de memoria necesarios para las consultas). Las consultas serán escritas en los 3 lenguajes de consultas con los que se trabajan en este proyecto.

Para las consultas realizadas se ha comprobado que los datos sobre los que se preguntaba estaban en la base de datos para que esta comparativa entre consultas tuvieran sentido (ya que podrían pertenecer esos datos al conjunto de datos que no se insertan correctamente, porque no fueran admitidos por alguna de las bases de datos).

4.3.1 Planificación de experimentos

Estas 18 consultas que se han creado se pueden agrupar en 6 diferentes grupos, constando cada uno de ellos de 3 consultas cada uno, cada uno de ellos con una característica u objetivo distinto. Estos grupos se definen a continuación:

- Grupo A ("consultas único objeto"): aquellas consultas en las que se pregunta por un único objeto de una tripleta RDF. Estas consultas son las más simples que se pueden realizar, además al tener Sesame, Oralce y Mulgara índice para este tipo de consultas (índice Sujeto-Predicado-Objeto) ayudara en la busqueda del resultado. A este grupo pertenecen las consultas CF1, CF2 y CF3.

- Grupo B ("consultas varios objetos"): son iguales a las del grupo A pero en este caso se devuelve varios resultados, es decir existen varios objetos para el sujeto y el predicado por el que se pregunta en la consulta. A este grupo pertenecen CF4, CF5 y CF6.

- Grupo C ("consultas muchos objetos"): en este grupo nos encontramos con las consultas CF7, CF8 y CF9 que son del mismo tipo que las del grupo B pero en ellas se devuelven entre 171 y 191 resultados dependiendo de la consulta, una cantidad muy elevada de datos a devolver, comparándolas con las del grupo B.

- Grupo D ("consultas sobre sujeto y objeto"): en este grupo están las consultas CF10, CF11 y CF12 en las que se pregunta por el sujeto y por el objeto de una tripleta RDF. Estas consultas Sesame se ayudara del índice POSC, Oracle del índice del predicado y Kowari y Mulgara de los índices predicado-sujeto-objeto y predicado-objeto-sujeto.

- Grupo: E ("consultas sobre sujeto"): en este grupo están las consultas CF13, CF14 y CF15 en las cuales se pregunta por el sujeto de una tripleta RDF y se contara con el índice POS en el caso de Sesame, el del predicado en caso de Oracle y en el índice predicado-objeto-sujeto en el caso de Mulgara y Kowari.

- Grupo F ("consultas sobre predicado"): en este último grupo existen tres consultas CF16, CF17 y CF18 en las que se pregunta por el predicado de alguna tripleta RDF, es decir, qué relación existe entre un sujeto y un objeto. Estas tres consultas lo que tienen de especial es que ahora ni en Oracle ni en Sesame se dispone de un índice que ayude a realizar estas consultas, y con ello podemos ver qué diferencias hay de rendimiento respecto a consultas que si tienen algún índice de ayuda. También se comprobara si el rendimiento de Mulgara y Kowari es mejor ya que estas sí que tienen índices de ayuda para estas consultas (recordar los 6 índices de que disponían visto en el apartado modelo de datos).

Con estos 6 grupos se prueban diferentes tipos de consultas en los que se preguntan por las distintas partes que forman la tripleta, realizando en cada uno de los grupos 3 consultas para poder evaluar de forma correcta el funcionamiento y el rendimiento de las consultas en los gestores RDF.

Antes de comenzar a ver los resultados de los experimentos aclarar que las consultas del grupo A y grupo B son aquellas como: ¿qué libros escribió el escritor x? , ¿En qué equipos jugo Romario? ... También destacar las consultas del grupo F ya que comparándolas con las anteriores permitirá ver si es necesario o merece la pena tener más índices en la base de datos o con los dos que se tienen en Oracle y Sesame frente a los seis de Mulgara y Kowari.

Como ya se ha dicho antes el objetivo de este apartado es comparar el rendimiento de las 4 bases de datos a la hora de realizar capturas, para ello se han tenido que tomar una serie de datos como uso CPU, uso de memoria y tiempo de la consulta. A continuación se muestra como se han obtenido dichas medidas para cada unas de las bases de datos:

a) Planificación para Oracle

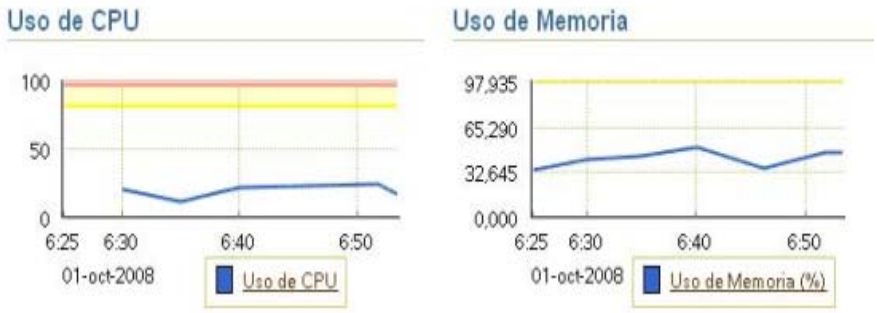
Para calcular el tiempo de una consulta, se ha optado por poner delante y detrás de la consulta en SQL, la siguiente sentencia:

```
SELECT TO_CHAR (systimestamp, 'DD/MM/YYYY HH24:MI:SS.FF3') from
dual;
```

Esta sentencia mostrará la hora con horas, minutes, segundos y milisegundos. Al escribirla delante y detrás de la consulta nos mostrara el tiempo que transcurrió mientras que se hizo la consulta.

Para el uso de CPU y de memoria utilizaremos Oracle Enterprise manager el cual dispone de una sección de rendimiento en la cual podemos ver estos valores tanto numéricamente como en gráficas como se puede ver en la siguiente imagen:

Ilustración 4-15: Uso memoria y CPU de Oracle



Como se puede ver en la Ilustración 4-15 hay dos gráficas. En la primera se muestra el porcentaje de uso de la CPU (eje vertical) durante un periodo de tiempo (eje horizontal). La segunda grafica muestra que porcentaje de memoria es utilizada mientras que Oracle realiza sus operaciones durante un cierto periodo de tiempo.

b) Planificación para Kowari y Mulgara

Para ver el tiempo que requiere una consulta para llevarse a cabo disponemos de dos opciones. Si realizamos la consulta mediante la Shell de ITQL tendremos que activar la función set time y ponerla a on para que después de realizar una consulta se nos muestre el tiempo que esta operación ha requerido (esta opción no es válida en Mulgara ya que este parámetro aunque existe aunque se active no funciona), como se puede ver en la siguiente imagen:

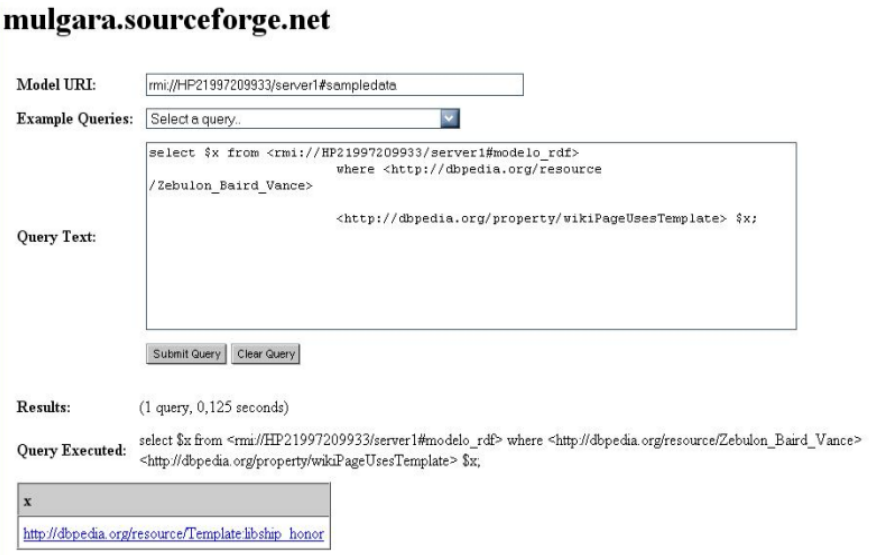
Ilustración 4-16: Consulta Shell de Kowari

```
iTQL> select $x from <rmi://HP21997209933/server1#modelo_rdf>
where <http://dbpedia.org/resource/Philadelphia_mayoral_election%2C_1995> <http://dbpedia.org/property/wikiPageUsesTemplate>
$x;
[ http://dbpedia.org/resource/Template:sequence ]
[ http://dbpedia.org/resource/Template:election_box_candidate_with_party_link ]
[ http://dbpedia.org/resource/Template:election_box_hold_with_party_link ]
3 rows returned.
Command execution time - 0.828 seconds
```

La Ilustración 4-16 muestra una consulta realizada en ITQL en la como se pude ver al final de su ejecución muestra el tiempo que tardo en realizar esta operación.

El otro método seria directamente realizar las consultas mediante Kowari Viewer que sin ser necesario activar ninguna opción te muestra el tiempo que requirió la consulta:

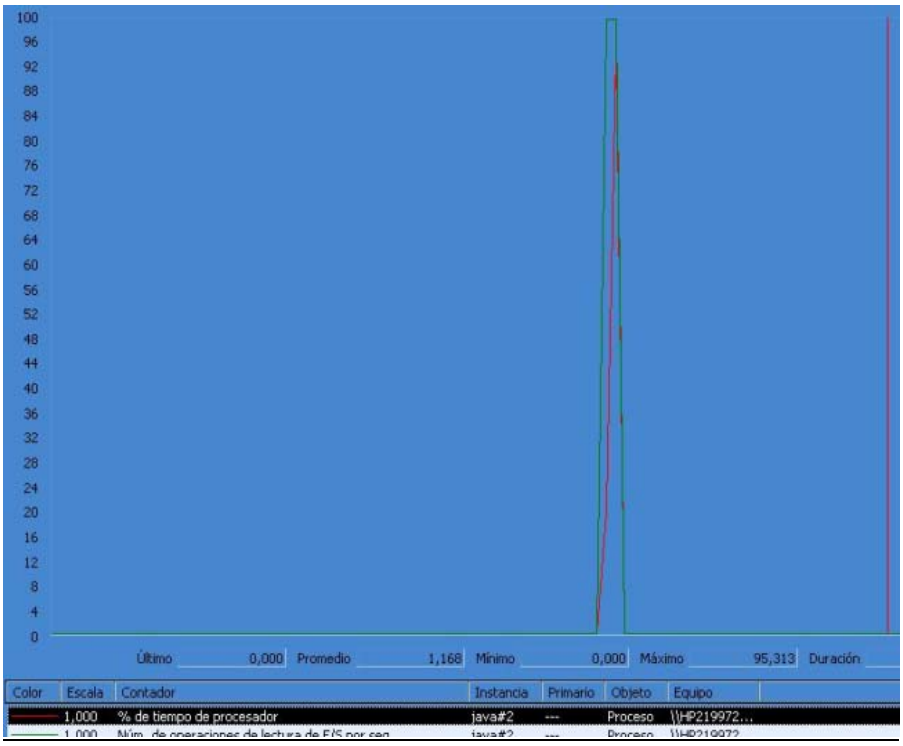
Ilustración 4-17: Consulta Mulgara Viewer



En la Ilustración 4-17 se muestra el resultado obtenido al realizar una consulta en Mulgara Viewer y el apartado Results nos muestra el tiempo de ejecución de la consulta.

Para conocer el uso de la memoria y el uso de CPU durante las consultas se utiliza System Explorer del cual se hablo en apartados anteriores. Solo es necesario saber a qué proceso hace referencia nuestra base de datos y una vez que se tiene esta información el propio programa nos proporciona mediante gráficas y datos numéricos esta información, como podemos ver en la siguiente imagen:

Ilustración 4-18: Gráfica System Explorer



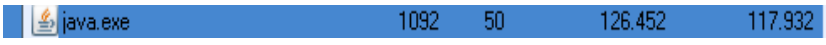
En la Ilustración 4-18 se muestra una gráfica que se ha obtenido mediante la herramienta System Explorer. En esta gráfica existen dos parámetros que son los siguientes:

- Línea roja = **tiempo de procesador**, que es el tiempo durante el cual todos los subprocesos del proceso han utilizado el procesador para la ejecución de instrucciones.

- Línea Azul = número de operaciones de lectura de E/S por segundo , que La frecuencia a la que el proceso está realizando operaciones de E/S de lectura. Este contador reúne toda la actividad de E/S generada por el proceso para incluir E/S de dispositivo, red y archivos.

Siendo en la gráfica el eje horizontal la línea de tiempo y el eje vertical el porcentaje.

Ilustración 4-19: Proceso java



En la Ilustración 4-19 el primer número que se muestra es el PID del proceso (es decir el identificador del proceso), el segundo seria el uso de la CPU en ese momento, el

tercero el uso de memoria que se está produciendo (en kilobytes) y por último el tamaño de VM (el tamaño de memoria virtual que usa el proceso).

Nota: en el anexo del proyecto se explica cómo utilizar y crear las gráficas de rendimiento que nos ofrece System Explorer.

c) Planificación para Sesame

El método para conocer el uso de memoria y uso de CPU durante las consultas en Sesame es igual al de Kowari y Mulgara.

Mientras que para conocer el tiempo que necesita una consulta para ser llevado a cabo, lo obtenemos directamente cuando realizamos las consultas al repositorio mediante la consola de Sesame que justo después del resultado de la consulta nos muestra el tiempo que tardo expresado en milisegundos:

Ilustración 4-20: Consulta consola Sesame

```
rdf2> serql SELECT A FROM {<http://dbpedia.org/resource/Philadelphia_mayoral_ele
ction%2C_1999>} <http://dbpedia.org/property/votes> {A}.
Evaluating query...
+-------------------------------------------------------------------------------+
| A                                                                             |
+-------------------------------------------------------------------------------+
| "5376"^^<http://www.w3.org/2001/XMLSchema#Integer>                            |
| "211136"^^<http://www.w3.org/2001/XMLSchema#Integer>                          |
| "203908"^^<http://www.w3.org/2001/XMLSchema#Integer>                          |
+-------------------------------------------------------------------------------+
3 result(s) (174 ms)
```

La Ilustración 4-20 muestra una consulta realizada sobre un repositorio y como se pude ver después de los resultados obtenidos se puede ver cuánto tiempo a requerido esta consulta y cuanto resultados a obtenido.

4.3.2 Resultados rendimiento consultas

Una vez explicado el porqué de las consultas y como se obtienen los datos para poder medir y comparar el rendimiento de las consultas realizadas en los gestores RDF, se muestran los resultados obtenidos agrupados por grupos de consultas:

4.3.2.1 **GRUPO A** (consultas único objeto)***:***

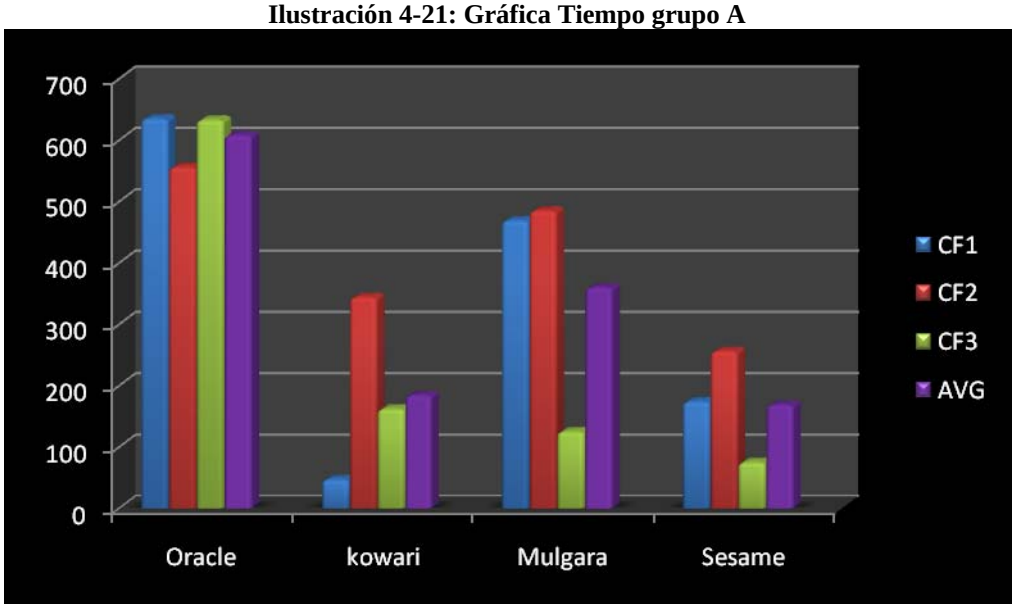
CF1: En esta consulta se pregunta por el número de votos para cada uno de los participantes de las elecciones de Philadelphia del año 1999.

CF2: Con esta consulta se pretende saber el número de páginas de que consta el libro The city of the dead.

CF3: Con esta consulta se quiere conocer cuál es el autor del libro The city of the dead, libro por el cual se pregunto en la consulta CF2.

a) TIEMPO: a continuación se muestra la gráfica de los resultados obtenido por estas tres consultas en cuanto a tiempo se refiere en cada uno de los gestores RDF. El tiempo esta expresado en milisegundos. Además en la gráfica se ha añadido la media (AVG) de las 3 consultas en cada unos de los gestores.

Ilustración 4-21: Gráfica Tiempo grupo A



Observando estos resultados se puede ver que los mejores resultados los ha obtenido Sesame que ha sido el gestor RDF con menor tiempo de respuesta, aunque Kowari ha obtenido tambien tiempos muy similares.

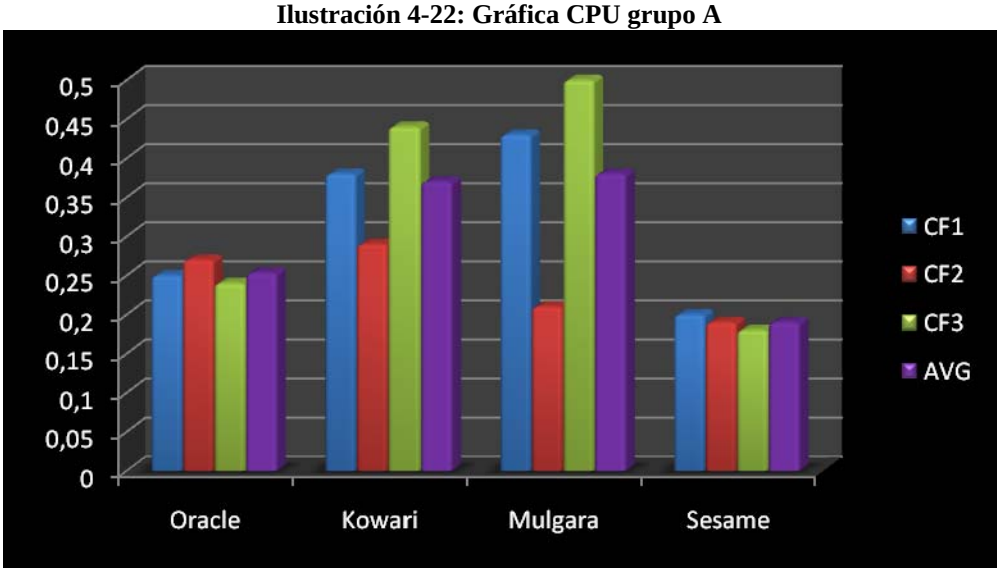
Respecto a Mulgara indicar que en un principio cabia suponer obtener resultados muy parecido a los de Kowari, cosa que no ha sido a si para esto tipo de consultas. Esto puede ser debido a que Mulgara tiene un millon más de tripletas insertadas (recordar que se comento en puntos anteirores que este millon más era para realizar pruebas de escalabilidad) y por eso el tiempo de respuesta puede ser algo mayor.

Notar tambien que la disminución sustancial de tiempo que se ha notado en la consulta CF3 en Mulgara, kowari y Sesame se debe principalmente a que en la consulta que se ejecuto anteriormente se preguntaba por el mismo sujeto lo único que ha cambiado ha sido el predicado y por ello los punteros del índice que se utiliza en esta consulta están más cerca del resultado y por eso ahora los tiempos de resolución de la consulta son menores que en los casos anteriores.

Por último con bastante diferencia respecto a los demas gestores Oracle obtiene unos resultados peores, siendo la media de tiempo casi 3 veces mayor que en Sesame (una media de 608 ms frente a los 168,33 de Sesame). Eso si estos resultados obtenidos en Oracle han experimentado poca diferencia en sus resultados.

b) CPU: En la siguiente gráfica se muestra el uso de CPU de las 3 consultas en los gestores RDF durante su ejecución. Como en el caso del tiempo tambien se añade la media de las 3 consultas (AVG). Los mostrados en esta gráfica muestran exactemente el porcentaje de uso de CPU que requiren las consultas durante su ejecución.

Ilustración 4-22: Gráfica CPU grupo A



Los resultados de uso de CPU tienen valores similares dentro de cada uno de los gestores y las diferencias entre los resultados de las diferentes consultas son mínimas.

Los mejores resultados al igual que en el tiempo son para Oracle con un media del 19% de uso CPU, la mitad que en otros gestores como Kowari y Mulgara.

4.3.2.2 **GRUPO B** (consultas varios objetos)**:**

Son consultas similares a las del grupo A con la diferencia que se van a devolver más de un resultado.

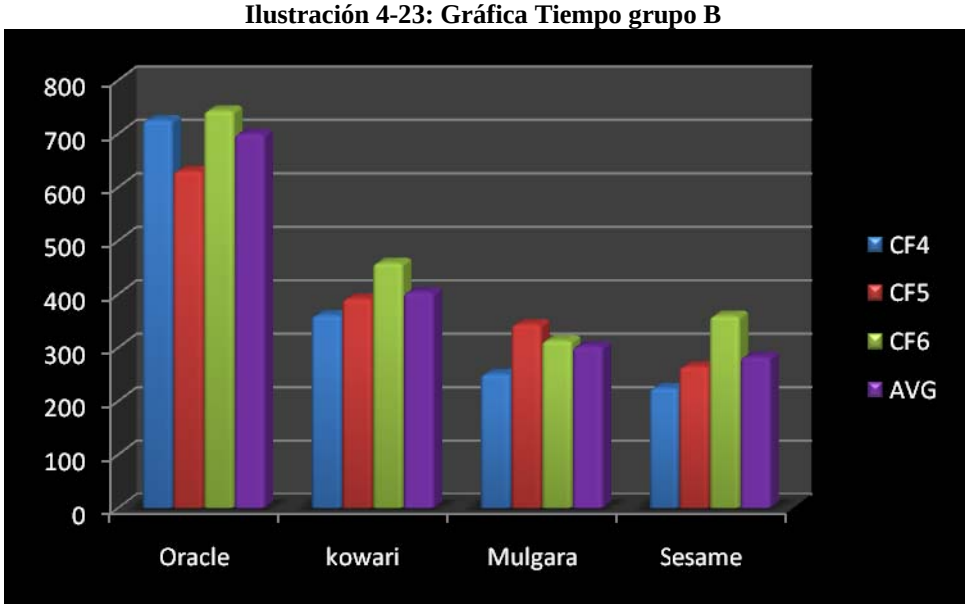
CF4: En esta consulta se pregunta por las divisiones de un organismo llamado Trichuris.

CF5: Con esta consulta se consigue conocer donde jugó el jugador Hristo Arangelov. EL resultado son los 4 clubs en los que jugó este jugador.

CF6: Consulta utilizada para conocer donde se aplico la ley AMNESTY LAW:

a) **TIEMPO:** Se muestra la siguiente gráfica donde se puede ver el tiempo que a llevado realizar cada una de las consultas del grupo B en los 4 gestores RDF. Recordar que el tiempo esta expresado en milisegundos y AVG es la media de las 3 consultas en cada uno de los gestores RDF.

Ilustración 4-23: Gráfica Tiempo grupo B

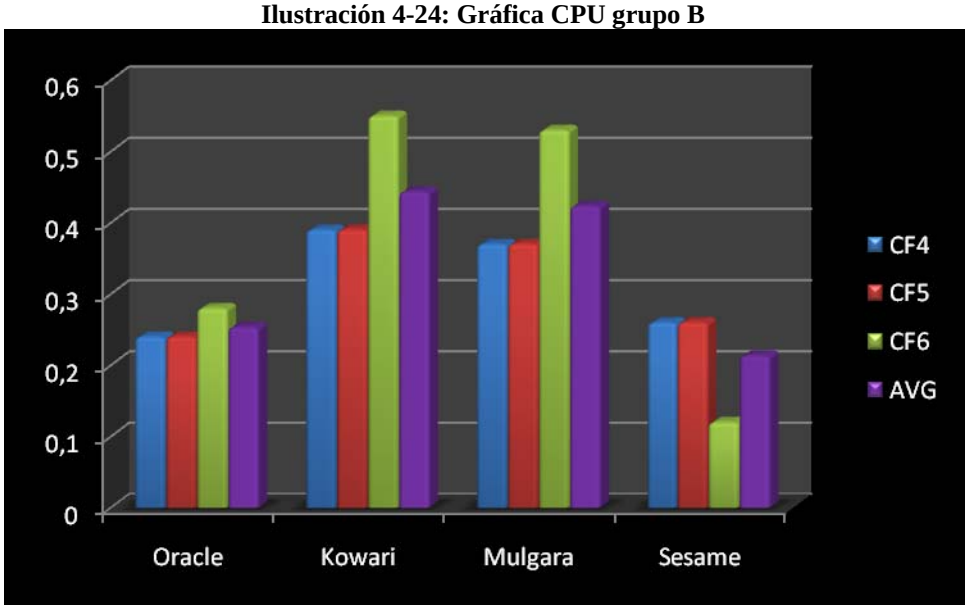


Ahora los resultados para este tipo de consultas son algo más elevados, con la aproximación del tiempo de respuesta de Mulgara y Kowari a Sesame que otra vez es el que obtiene los mejores resultados en tiempo con una media de 282,33 ms.

Por otra parte los resultados obtenidos en Oracle siguen siendo algo elevados si los comparamos con los otros gestores.

b) **CPU:** A continuacion se muestran los resultados de porcentaje de uso de la CPU para las consultas realizadas dentro del grupo B.

Ilustración 4-24: Gráfica CPU grupo B



Los resultados que se han obtendio han sido practicamente los mismos que los obtendos en las consultas del grupo A. Tanto Sesama como Oracle siguen siendo los gestores que mejores resultados nos proporcioanan con un porcentaje de uso de casi la mitad que en Kowari y Mulgara.

Notar que en Kowari y Mulgara las medias de rendimiento son hasta ahora muy parecidas al tratarse de sistemas muy similares (La media de Kowari es de 0,44 frente al 0,42 de Mulgara).

4.3.2.3 **GRUPO C** (consultas muchos objetos)**:**

al igual que las del grupo B estas consultas devuelven varios resultaos pero en una cantidad más elevada, más concretamente entre 171 y 191.

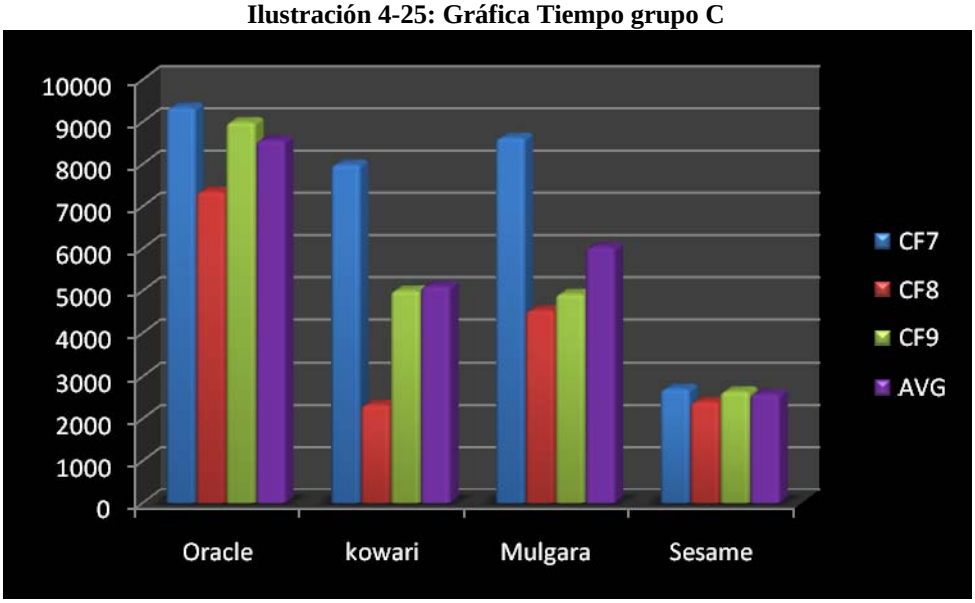
CF7: Con esta consulta se quieren conocer los juegos que se tienen almacenados en la base de datos para la categoría de Personal Computer. A diferencia de las consultas anteriores esta consulta nos devuelve un gran número de valores en este caso 171.

CF8: jugadores que han jugado de delantero (Striker)

CF9: Consulta que nos indica que programas están disponibles para Linux. Esta consulta devuelve 191 resultados.

a) **TIEMPO:** En la siguiente gráfica se muestra el tiempo de respuesta de las consultas del grupo C. El en el eje vertical se indica el tiempo en milisegundos y en eje horizontal cada uno de los gestores RDF.

Ilustración 4-25: Gráfica Tiempo grupo C



En este grupo como es logico el tiempo de las consultas a aumentado respecto a las consultas del grupo B, que eran similares pero con menos resultados, pero es lógico debido al gran número de resultados que ha respecto a las del grupo B. Pero si se observa la gráfica los resultados son similares en cuanto al comportamiento de los gestores RDF, ya que Sesame sigue opteniendo tiempos muy similares entre las consultas y sigue siendo el gestor que tarda menos en procesar y mostrar los resultados de las consultas.

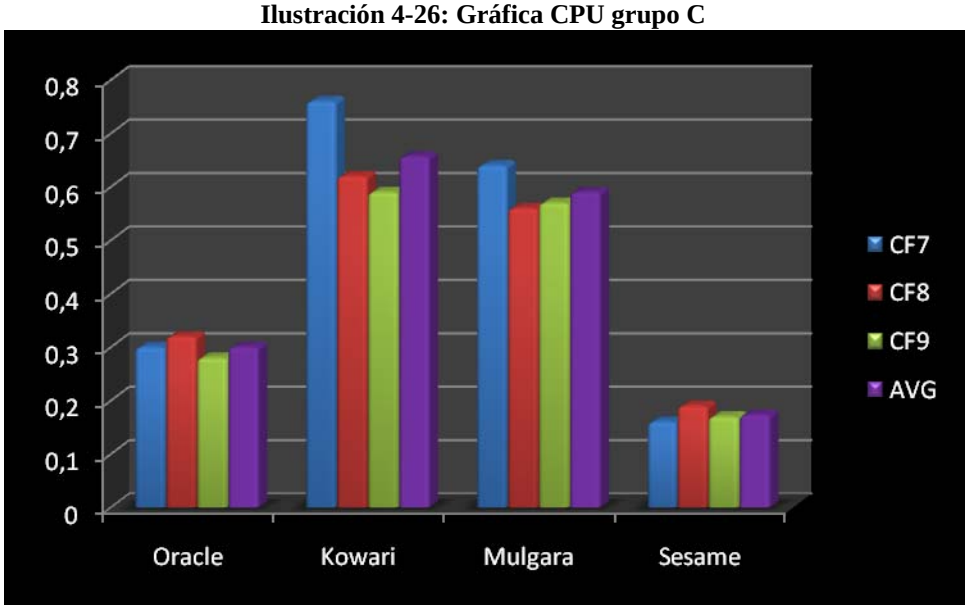
Por otra parte el tiempo medio de las consultas en Kowari y Mulgara sigue siendo muy similar, 5101 ms en Kowari mientras que el tiempo mediao en Mulgara es de 6024 ms.

Oracle sigue obteniendo unos resultados de rendimiento en tiempo que en este grupo multiplica por mas de 3 el tiempo obtenido en Sesame (En Oracle una media de 8555,33 ms frente a los 2577,33 ms de media que tienen Sesame).

Tambien se observa que los tiempos en Oracle y en Sesame son mas constantes entre las diferentes consultas, mientras que en Kowari y Mulgara se encuantran mas diferencias entre los resultados obtenidos en una y otras consultas.

b) CPU: En la siguiente gráfica se muestra el uso de CPU de las consultas del grupo C. El en el eje vertical se indica el porcentaje de memoria que se utiliza y en eje horizontal cada uno de los gestores RDF.

Ilustración 4-26: Gráfica CPU grupo C



Como se puede apreciar en este gráfico el número de resultados, que ahora es mayor que en los anteriores grupos, si ha afectado a Mulgara y a Kowari en cuanto a Uso de CPU incrementandose considerablemente, aumnetado en un 20% en los dos gestores.

Sin embargo en cuanto a Oracle y Sesame esto no ha sido un problema ya que siguen manteniendo una media muy parecida a la obtenida en los otros grupos de consultas, incluso la de Sesame es inferior situandose en uso medio de CPU del 17%.

4.3.2.4 **GRUPO D** (consultas sobre sujeto y objeto)**:**

Formado por 3 consultas en las que se pregunta por el sujeto y por el objeto de una tripleta RDF. Estas consultas a diferencia que la del resto de grupos vistos hasta ahora preguntan por 2 partes de la tripleta y no por una. . Ahora el índice que se utilizara será POSC en Sesame, mientras que Kowari y Mulgara dispondrá de 2 índices para la

realización de esta consulta (predicado-sujeto-objeto, predicado-objeto-sujeto) y Oracle el índice al predicado.

CF10: En esta consulta se quiere conocer qué juegos están almacenados en nuestra base de datos y en que plataformas están dichos juegos

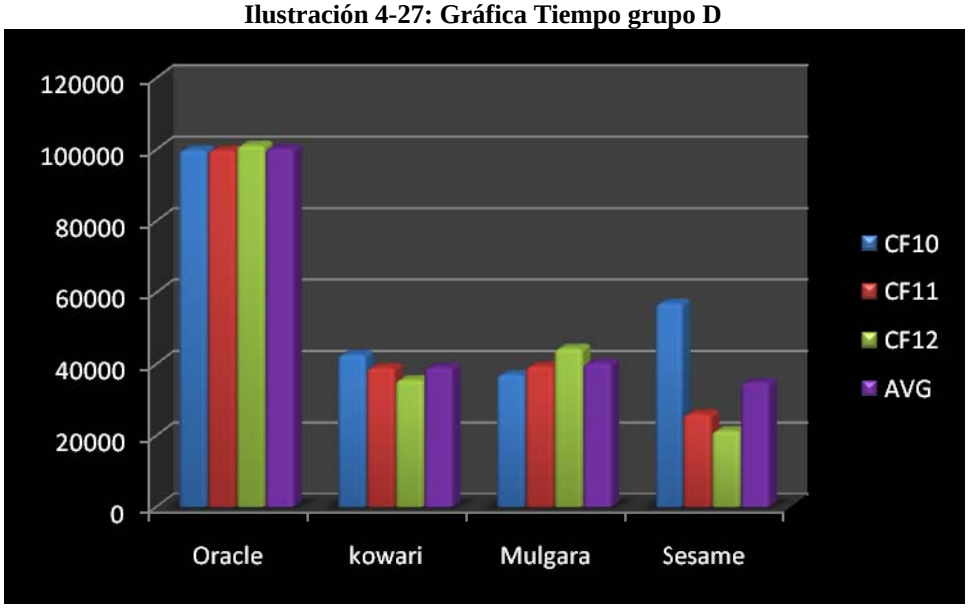
El resultado de esta consulta es de 9142 juegos y sus correspondientes plataformas.

CF11: Se quiere conocer las personas que han sido Managers de futbol y en que equipos han estado contratados.

CF12: Se quieren conocer los programas de ordenador que están almacenados en los repositorios y también conocer para qué plataformas están disponibles.

a) **TIEMPO:** En la siguiente gráfica se muestra el tiempo de respuesta de las consultas del grupo D. El en el eje vertical se indica el tiempo en milisegundos y en eje horizontal cada uno de los gestores RDF.

Ilustración 4-27: Gráfica Tiempo grupo D

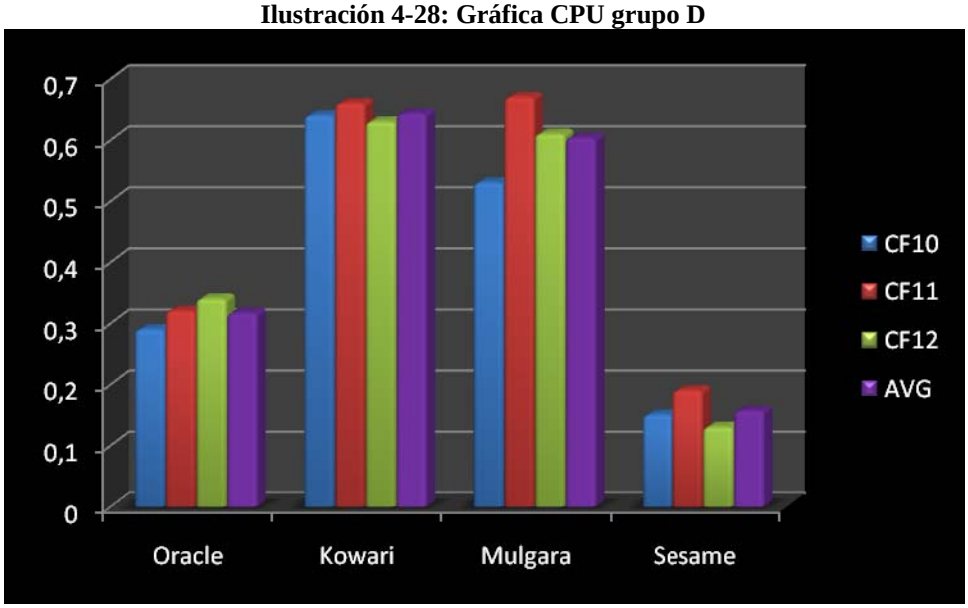


El menor tiempo medio de las consultas sigue siendo para Sesame, pero en este caso la difrencia con respecto a Mulgara y a Kowari a disminuido, siendo este tiempo medio casi igual. Esto es debido principalmente a que ahora para realizar este tipo de busquedas en estas consultas Kowari y Mulgara disponen de dos indices, pero a persar de ello no han sido capaces de mejor los tiempos de Sesame, salvo en la consulta CF10.

Por lo visto en este grupo parece que con los dos indices que tiene Sesame parece suficiente para conseguir buenos resultados en cuanto a tiempo se refiere en cualquier tipo de consulta, ya se pregunto por el objeto, el sujeto o el predicado de una tripleta RDF.

b) CPU: En la siguiente gráfica se muestra el uso de CPU de las consultas del grupo D. El en el eje vertical se indica el porcentaje de memoria que se utiliza y en eje horizontal cada uno de los gestores RDF.

Ilustración 4-28: Gráfica CPU grupo D



En cuanto a uso CPU se refiere esta grafica nos muestra que todo sigue igual y los rendmientos en los 4 gestores muestran resultados similares a los de los anteriores grupos de consultas. Con lo que se puede decir que por lo visto hasta ahora el relaizar un tipo de consulta u otra sobre los gestores RDF no influeye en el uso de CPU ya que los valores medios obtenidos son similares.

4.3.2.5 **GRUPO E** (consultas sobre sujeto)**:**

Consultas en las que se pregunta por el sujeto de una tripleta RDF y se contara con el índice POS en el caso de Sesame, el del predicado en caso de Oracle y el de predicado-objeto-sujeto en el caso de Mulgara y Kowari.

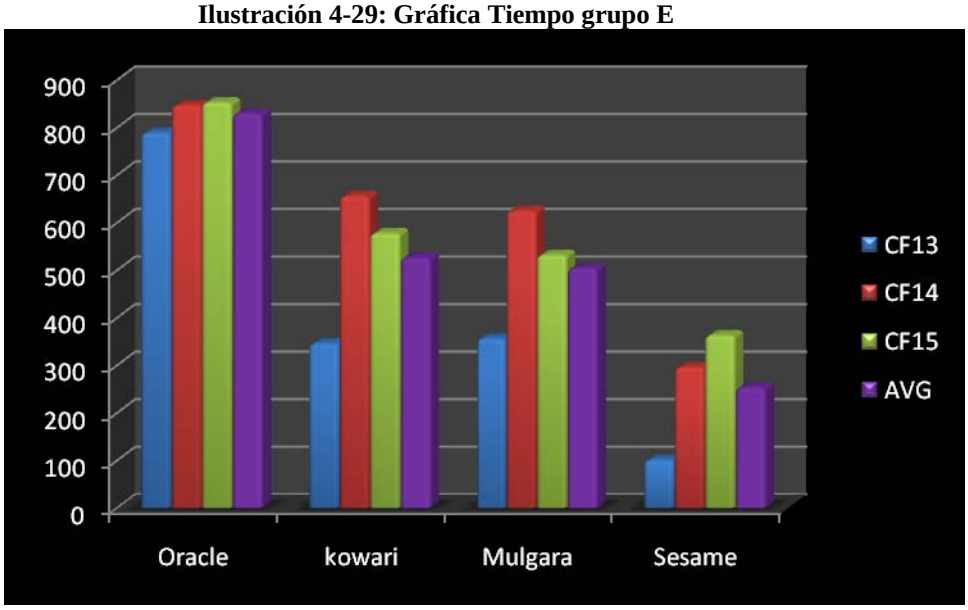
CF13: Ahora se pregunta por el sujeto de la tripleta RDF conocido uno de los objetos, es la consulta contraria a la número CF7.

CF14: Con esta consulta se consigue saber que gente ha muerto en el estado de MASSACHUSETTS.

CF15: Con esta consulta se quiere conocer a partir de el nombre de un capitulo de una serie de televisión, ver cuál es la serie a la que pertenece dicho capitulo.

a) TIEMPO: En la siguiente gráfica se muestra el tiempo de respuesta de las consultas del grupo D. El en el eje vertical se indica el tiempo en milisegundos y en eje horizontal cada uno de los gestores RDF.

Ilustración 4-29: Gráfica Tiempo grupo E

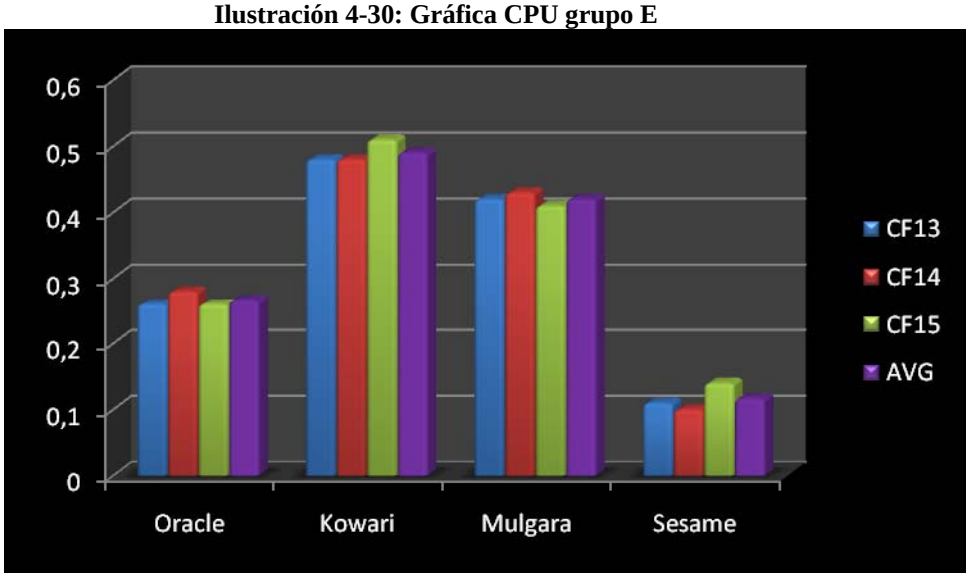


Respecto a lo que tiempos se refiere la utilización de un indice distinto para la ayuda en la busqueda de los objetos por los que se pregunta en la consulta respecto a los indices usados en otras consultas (como las del grupo A y B), no se producen grandes cambios en los tiempo obtendios mateniendose una media muy similar.

Sesame vuelve a obtener los mejores resultados con una media de 253,33 milisegundos frente a los 829,67 milisegundos que tiene de media Oracle (casi 4 veces el tiempo de Sesame).

b) CPU: En la siguiente gráfica se muestra el uso de CPU de las consultas del grupo D. El en el eje vertical se indica el porcentaje de memoria que se utiliza y en eje horizontal cada uno de los gestores RDF.

Ilustración 4-30: Gráfica CPU grupo E



Como ahora el número de resultados ya no estan elevado como en los grupos C y D, el uso de CPU en las Kowari y Mulgara baja entre un 15 y 20 %. Pero sigue siendo un porcentaje elevado respecto a Oracle que tiene casi la mitad de uso que en ellas, y sobre la gran diferencia sigue siendo con Sesama donde el el uso de CPU es tan solo del 10% (3 veces menos que en Mulgara y Kowari).

4.3.2.6 **GRUPO F** (consultas sobre predicado)**:**

Estas tres consultas lo que tienen de especial es que ahora ni en Oracle ni en Sesame disponen de un índice que ayude a realizar estas consultas, y con ello podemos ver qué diferencias hay de rendimiento respecto a consultas que si tienen algún índice de ayuda.

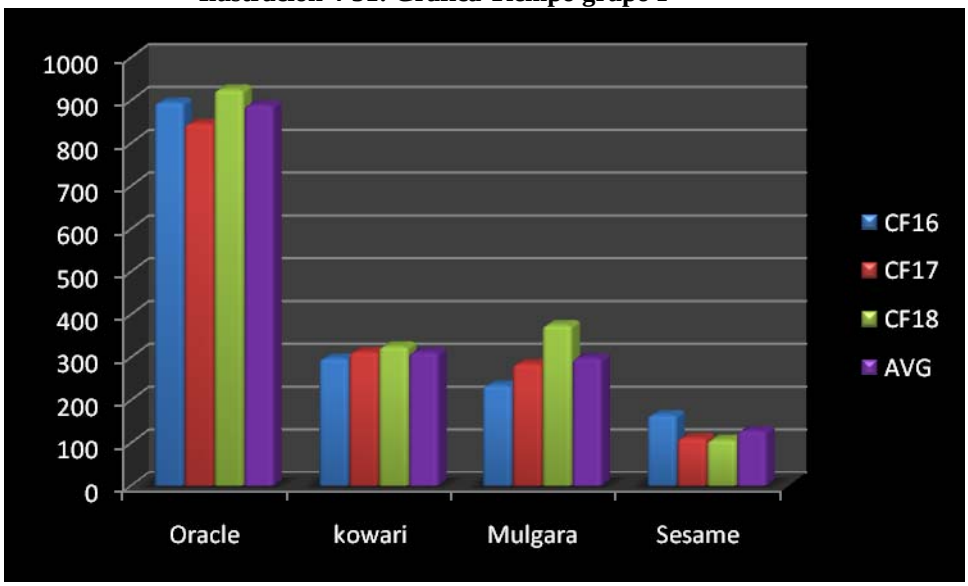
CF16: En la siguiente consulta se quiere conocer el predicado o posibles predicados para una tripleta RDF conocidos los sujetos y objetos de está. Para ser más exacto se preguntara por la relación entre el partido demócrata y las elecciones de Philadelphia 1999, el resultado es winner, ya que en ese año el ganador de las elecciones fue el partido demócrata.

CF17: En la siguiente consulta se pregunta por la relación que existe entre el libro The city of the dead y la persona Lloyd Rose.

CF18: En esta consulta se quiere conocer si existe algún tipo de relación entre Philibert Francois Rouxel de Blanchelande y la ciudad de Dijon. Los resultados de la consulta nos muestran que Dijon es la ciudad donde nació este personaje.

a) TIEMPO: En la siguiente gráfica se muestra el tiempo de respuesta de las consultas del grupo D. El en el eje vertical se indica el tiempo en milisegundos y en eje horizontal cada uno de los gestores RDF.

Ilustración 4-31: Gráfica Tiempo grupo F



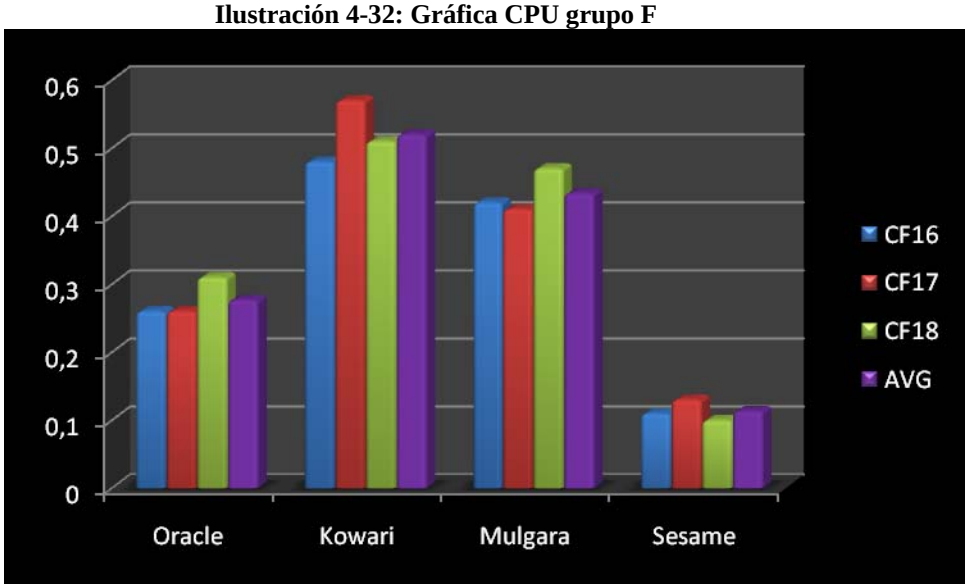
A pesar de que Sesame no dispone de incide esto no es problema en cuanto al tiempo de respuesta pra este tipo de consultas, ya que la media de tiempo obtendia para este grupo de consultas ha sido la menor (incluso que en las del grupo A que eran consultas simples en las que además se disponia de un índice), exactamente 127,67 milisegundos de media.

Kowari y Mulgara tambien han obtenido buenos resultados, pero siguen estando algo lejos de los tiempos de Sesame (Kowari 311,33 y 297,67 Sesame).

Oracle muestra una media de 888,33 milisegundos que ahora es mas de 8 ves mayor que el tiempo medio de Sesame para este grupo de consultas.

b) CPU: En la siguiente gráfica se muestra el uso de CPU de las consultas del grupo D. El en el eje vertical se indica el porcentaje de memoria que se utiliza y en eje horizontal cada uno de los gestores RDF.

Ilustración 4-32: Gráfica CPU grupo F



4.4 Conclusiones rendimiento en las consultas

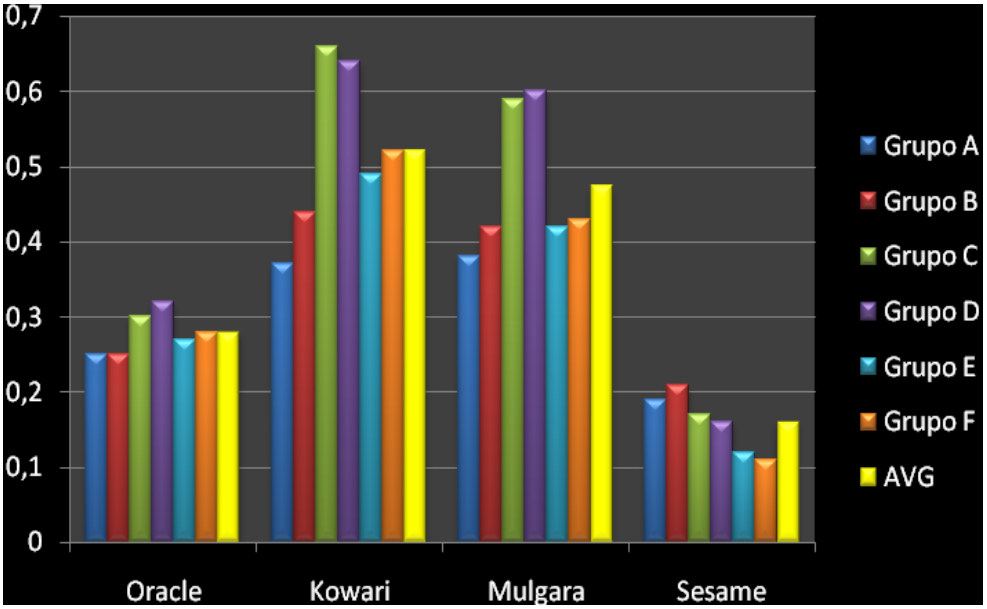
A continuación se muestra una tabla en la que se quiere resumir los datos de Uso de CPU y uso de la memoria durante las consultas realizadas:

Tabla 4-4: Uso medio de CPY y memoria de los gestores RDF

	Oracle	Kowari	Mulgara	Sesame
Uso medio de CPU (%)	0,278	0,52	0,473	0,16
Uso medio de memoria(KB)	442400	43274,66	34014,4	86228,66

Por último se añaden las graficas con las medias de uso de CPU de cada uno de los grupos de consultas que se tienen por cada gestor RDF. También se añade la gráfica con las medias de los tiempos, pero en este caso se ponen aparte las consultas del grupo C y D ya que al ser consultas con tiempos muy altos (debido a la cantidad de datos devueltos) no es muy lógico mostrarlas con las de los otros grupos de consultas:

Ilustración 4-33: Total media CPU



En la gráfica 4.33 de aprecia como en Sesame y en Oracle las medias de uso de CPU en los diferentes grupos son mas constantes y difieren pco entre si, cosa que en Mulgara y Kowari no ocurre, donde las diferencias son más notables.

Además en este gráfica se puede apreciar mucho más claro lo ya dicho anteriormente, que es el buen rendimiento en cuanto de uso de CPU que tiene Sesame frente al resto de los gestores, con una media que no supera en ningun caso el 20 % de uso de CPU.

Ilustración 4-34: Total media Tiempo

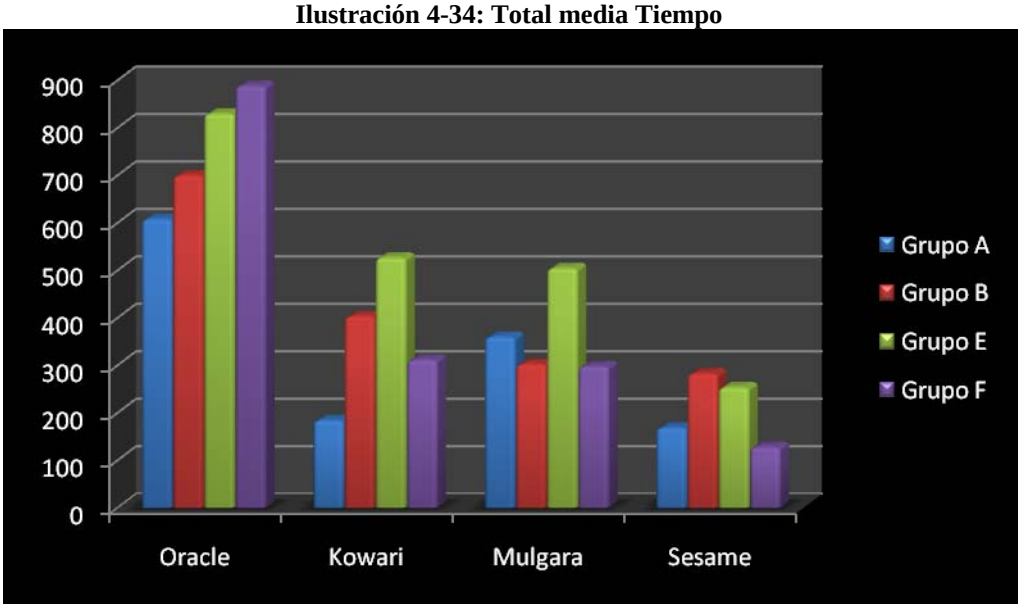
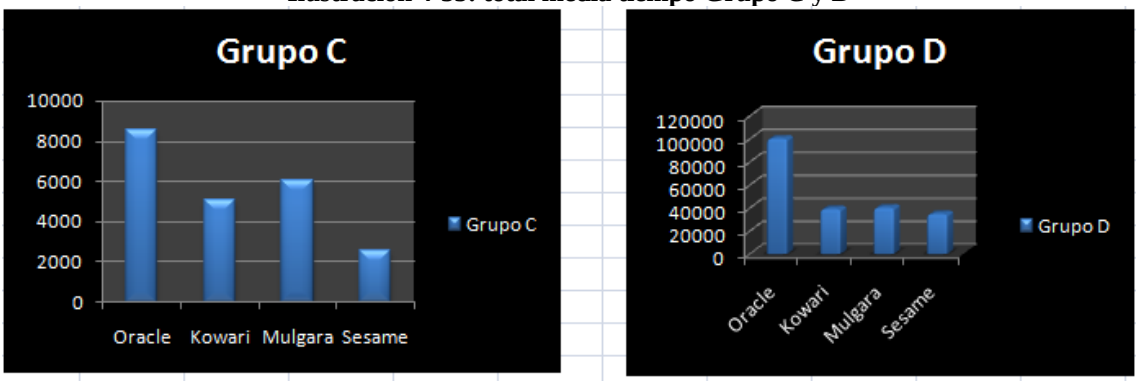


Ilustración 4-35: total media tiempo Grupo C y D



Viendo las gráficas 4.34 y 4.35 se puede apreciar también de forma muy clara el mejor rendimiento en cuanto a tiempo se refiere de Sesame frente a los otros gestores. También queda claro las escasa diferencia que existe entre Mulgara y Kowari también se colaboran observando los resultados obtenidos en cuanto a tiempo de respuesta en las consultas, al igual que ya sucedería en otras mediadas de rendimiento. Po último en estas dos gráficas se aprecia de forma clara que Oracle tiene unos tiempo de media muchos más altos que los otros gestores.

Queda claro que después de ver las gráficas y tablas anteriores en cuanto a tiempo de respuesta y uso de CPU durante las consultas realizadas, Sesame ha obtenido los mejores resultados. Su único punto débil si lo comparamos con los otros gestores RDF es el uso de memoria que requiere que es prácticamente el doble que en los otros gestores RDF, algo que puede ser un problema si los equipos con los que se trabaja no tienen la suficiente memoria.

Viendo los resultados obtenidos en las consultas realizadas se puede decir que Sesame es sin duda la que mejor rendimiento ha proporcionado para los 3 campos que se han medido, sobre todo destacando en el poco uso de procesador que realiza a la hora de realizar las consultas y el poco tiempo que tarda en realizar las consultas a pesar de que por ejemplo en comparación con Kowari y Mulgara que disponen de 6 índices en lugar de los dos con los que cuanto Sesame.

También estas pruebas han servido para seguir corroborando el mal funcionamiento de Oracle (si lo comparamos con los otros gestores RDF) que al igual que ya paso en la inserción ha sido la peor de las 4 bases de datos en este apartado, sobre todo en el tiempo de respuesta, siendo más del doble que en las otras bases de datos además de un uso de memoria muy elevado.

Por último Kowari y Mulgara como era de esperar han tenido un rendimiento muy parecido y las pequeñas diferencias que hay entre ellas sobre todo en el tiempo podrían deberse a que Mulgara tiene casi un millón más de tripletas insertadas. Tanto en el tiempo de respuesta como en el uso de memoria se han obtenido unos resultados muy satisfactorios quizás la única pega seria el uso de la CPU que con respecto a Sesame es bastante más elevado.

5. Conclusiones y Líneas Futuras

5.1 Conclusiones

En este aparatado se van a analizar los resultados obtenidos en el estudio realizado en este proyecto. Para ello se van analizar por separado los distintos elementos que han sido estudiados:

- *Arquitectura:* como se ha visto en este punto la gran diferencia en este apartado es que Oracle es una base de datos relacional la cual mediante una serie de paquetes de funciones se ha adaptado a las características que implica trabajar con tripletas RDF, mientras que Kowari, Mulgara y Sesame han creado una capa de almacenamiento propia para poder almacenar este tipo datos (tripletas RDF).

  Este aspecto es uno de los que nos llevan a pensar que el mal rendimiento obtenido en la parte de implementación sea provocado a esta arquitectura usada por Oracle para almacenar tripletas RDF. Oracle no ha creado ninguna capa sino que lo ha hecho a través de la capa Oracle's Network Database (que está a su vez construido en la capa de aplicaciones), este va a provocar que la escalabilidad en Oracle según tengamos una cierta cantidad de datos en nuestra base de datos va ser mucho peor que en las otras. Quizás para una cantidad pequeña de datos, pero cuando la base de datos contiene un gran número de datos como sucede este proyecto la escalabilidad y el rendimiento de Oracle no ha sido nada bueno comparado con el de los otros Gestores.

  Respecto a los otros gestores RDF (Kowari, Mulgara y Sesame) utilizan una arquitectura estructurada por capas. Los tres disponen de una capa de almacenamiento con su propio modelo de datos el cual se analizara en el siguiente punto, una capa pensada para la comunicación y otro para la interacción y manejo de las tripletas RDF.

  Quizás la mayor diferencia entre ellas este en la parte del tratamiento que se hace a las consultas. Sesame dispone de un modulo para consultas y Mulgara y Kowari tienen el Query engine (motor de consultas). Sesame en este apartado realiza un mejor tratamiento de las consultas realizadas.

- *Modelo de datos*: los cuatro gestores RDF a la hora de almacenar las tripletas RDF lo hacen de forma similar introduciendo el sujeto, el predicado y el sujeto, y un cuarto elemento que en Oracle es n número que identifica a la tripleta (que se genera al insertar esta tripleta en la base de datos tomando un valor que será el del número de tripletas que hasta en ese momento de la inserción haya almacenadas), en Mulgara y Kowari es un meta-nodo (indica modelo de sentencia) y en Sesame este cuarto elemento se denomina contexto.

  En este apartado vuelve a tomar ventaja Sesame, Kowari y Mulgara sobre Oracle, debido a que este último gestor al ser en realidad una base de datos relacional con una serie de paquetes que permiten trabajar con tripletas RDF (y no gestores con modelos propios para almacenar y trabajar con estas tripletas) necesita crear una serie de tablas para poder trabajar con este tipo de datos que aumentan el coste de trabajar con ellos. Oracle tiene que mantener las tablas MDSYS.RDF_MODEL$ (que contiene la información de los modelos almacenados en la base de datos) MDSYS.RDFM_model-name (contiene información de las tripletas almacenadas en un modelo especifico), RDF_NAMESPACE$ (información sobre Namespces que pueden ser usados en documentos RDF) y tabla MDSYS.RDF_VALUE$ (información sobre los sujetos, predicados y objetos utilizados para forma sentencias).

  También dentro de este aparato se han analizado los índices que utilizan cada uno de los gestores. Por una parte Sesame utilizan arboles B, Oracle arboles B* y Kowari y Mulgara arboles AVL. La utilización de un árbol AVL, implica un uso del disco muy deficiente: cada acceso implica leer un bloque de memoria y de este bloque únicamente se utiliza un registro, cuyos índices de izquierda y derecha apuntan, con mucha probabilidad, a un registro de otro bloque. Además este problema se agrava al manejar en este proyecto un gran número de datos (implica gran volumen de información en memoria).

  La última diferencia que existe en este apartado es el número de índices que se utiliza en cada uno de los gestores. Oracle y Sesame permite elegir el número de índices que se desean crear y sobre qué elementos se quieren crear. Mientras que Kowari y Mulgara cran directamente 6 índices como se comento en el apartado de modelo de datos. Esto a priori podría pensarse que es un

problema ya que al tener más índices el coste de actualización de cada uno de ellos en las inserciones haría que el rendimiento en estos gestores fuera peor, pero los resultados obtenidos para nada muestran este tipo de problemas como se pudo ver en ese apartado. Pero también es cierto que a la hora de las consultas tampoco han conseguido que Mulgara y Kowari obtengan unos mejores resultados por tener más índices que ayuden en la búsqueda.

Como se ha podido ver en el blog de uno de los desarrolladores de Mulgara [13], uno de los proyectos que tienen entre manos es que quieren hacer pasar de 6 a 3 índices porque hay muchos de ellos que ni se llegan a usar. Así que sería una buena idea como mejora para Mulgara y Kowari el poder seleccionar por uno mismo el número y que índices se quieren imprentar dentro del repositorio con el que se trabaje.

- *Lenguajes de consulta:* en este apartado el gestor más valorado es Sesame debido a que permite utilizar SPARQL que es el lenguaje de consulta que el 15 de Enero de 2008 el W3C propuso como lenguaje de consulta dentro de la Web semántica. Pero para este proyecto se trabajo con SeRQL dentro de Sesame, que es el lenguaje específico de Sesame, que varia simplemente en algunos aspectos de sintaxis frente a SPARQL. Se ha utilizado el lenguaje especifico de Sesame en lugar de SPARQL porque en los otros gestores también se han utilizado los gestores específicos suyo y además en Internet se han encontrado ya pruebas con SPARQL mientras que con SeRQL no tanto, y se ha considerado esto como un punto interesante para utilizar este lenguaje y ver más sobre su funcionamiento. Este lenguaje tiene algunas características como el de las sentencias Construct, Ask y Describe (que se vieron en los puntos 3.4.3.2, 3.4.3.3 y 3.4.3.4) que no poseen los lenguajes utilizados por Oracle (SQL), Kowari y Mulgara (iTQL).

  Respecto a Mulgara notar que para próximas versiones del producto se está trabajando para poder trabajar con SPARQL como lenguaje de consulta, para ser más concreto en la versión 2.0.6 del producto esta será una de las mejoras respecto a anteriores versiones.

  Un aspecto importante a tener en cuanta de Oracle frente a los otros gestores es que su lenguaje de consulta (SQL+ paquetes SDO_RDF) dan a

Oracle ciertas capacidades de inferencia (El desarrollo del poder interpretar lo que no está explicito en un texto). Esto se consigue definiendo reglas.

Para ver lo de forma más clara se va a suponer que se está trabajando en un repositorio en el que se están almacenando publicaciones de una biblioteca. En este caso se podría tener una clase base "Publicación" y varias subclases como "Libro", "Revista", "Artículo", etc. Ahora se alimenta el almacén RDF con tripletas que representen que "El Quijote" es un Libro. Si se hiciera una consulta solicitando los libros existentes aparecería, pero sin embargo si se pidiesen todas las publicaciones no se obtendría como resultado, pese a ser Libro una subclase de Publicación. Para ello se tendría que haber definido explícitamente que "El Quijote" es también una Publicación. Esto es lo que sucedería en un gestor como Mulgara que no tiene capacidades de inferencia, mientras que en Oracle se podría definir una regla (estas reglase se usan para definir una ontología, se pudo ver un ejemplo en el apartado 3.4.1 de SQL de Oracle) que indicara que si un elemento es un libro este a su vez también es una publicación.

Por último a su vez un aspecto importante de Sesame es que es capaz de generar tripletas inferidas en tiempo de carga a partir de una Ontología definida.

- *Seguridad:* en este apartado existe una gran diferencia entre Oracle y el resto de gestores RDF analizados debido principalmente a que Oracle en si es un sistema gestor de bases de datos relacional que junto a una serie de procedimientos puede almacenar y trabajar con tripletas RDF, mientras que el resto son almacenes específicos para tripletas RDF que a su vez soportan distintos gestores de bases de datos. Con esto lo que se quiere decir es que Oracle es un gestor de bases de datos con todo tipo de funcionalidades, el cual posee unos paquetes que nos permitan trabajar con RDF y aprovechar Oracle como repositorio de las tripletas RDF, mientras que los otros gestores RDF son eso simplemente gestores que nos permiten trabajar con tripletas RDF, pero no dan las funcionalidades que puede dar un gestor de bases de datos como puede ser Oracle. Y claro esta una de las funcionalidades que pude proporcionar Oracle frente a los otros gestores es la seguridad.

Así que la gran ventaja de trabajar con Oracle es que los métodos de seguridad ya viene integrados en él al ser un gestor de bases de datos, sin embargo, si utilizamos Mulgara o Sesame se tendrá que estudiar qué sistemas gestores bases de datos soportan (como PostgreSQL, MySql,…) y ver qué medios de seguridad pueden proporcionar estos sistemas, ya que Mulgara y Sesame por si solos poco aparte de la realización de backups o exportación de datos nos pueden aportar en cuanto a métodos y medidas de seguridad se refiere. Además se ha publicado una beta en al que la SAIL de Sesame esta soportada por Oracle [31] y tal u como se ha podido saber también se está trabajando para poder integrar Mulgara con Oracle.

- *Rendimiento durante inserción:* en este punto destacan sobre manera Kowari y Mulgara los cuales tienen un tiempo de inserción bastante inferior respecto a Sesame (3 veces más lento que estos) y Oracle (entre 7 y 8 veces más lento). El tiempo de inserción en Mulgara y en Kowari ha seguido una progresión lógica como se vio en las gráficas y mayor número de datos insertados en la base de datos más tiempo tardaban en insertar nuevos datos, algo lógico porque el coste de la inserción y el mantenimiento de los índices cuanto más datos se tengan insertados mayor será coste.

  Respecto al uso de CPU y uso de memoria durante las inserciones Mulgara y Kowari también obtienen los mejores resultados no teniendo un uso muy elevado (una media del 30%) frente al excesivo uso que se produce en los otros, en Oracle 87% y en Sesame aunque sea de un 20 % tiene en contra el gran uso de memoria que realiza de un 70% frente al 35% de media que aproximadamente utilizan Kowari y Mulgara.

  Respecto al espacio utilizado en disco aunque Mulgara y Kowari son los que necesitan una mayor ocupación en disco, esto es debido a que Kowari y Mulgara tienen 4 índices más y hace que su ocupación en disco es menor. Cierto es que viendo la espacio requerido por Sesame para sus dos índices (algo más de 500 MB para cada uno) si en Sesame se hubiesen implementado 4 índices más para de este modo tener los mismos índices de Mulgara o Kowari, y teniendo en cuanta la diferencia de datos insertados en cada ellos (Mulgara tiene un millón más de datos insertados), Sesame hubiese requerido de algo más de 2GB más

para mantener estos índices, y aun así tendría una menor ocupación en disco que Kowari y Sesame.

Lo que también se ha podido comprobar en estas pruebas realizadas en las inserciones, es el mal funcionamiento de Oracle. El tiempo de carga de datos ha sido elevadísimo, tardando 10 veces más que Kowari y Mulgara. Esta gran diferencia puede ser debido principalmente a la arquitectura y al modelo de datos usado por Oracle, como se ha dicho a lo largo de este proyecto Oracle es una base de datos relacional que no ha sido concebida como Kowari, Mulgara y Sesame que sean creado específicamente para trabajar con tripletas RDF. Oracle se puede decir que en este primer intento para trabajar con datos RDF ha hecho más bien un parche el cual permite trabajar con datos RDF pero los resultados obtenidos en este punto hacen ver que trabajar con Oracle no es una buena solución (en cuanto a rendimiento se refiere) especialmente cuando hay que manejar grandes cantidades de datos porque su rendimiento es muy inferior respecto a sus competidores y además como se podrá ver en el siguiente punto tampoco ha tenido un rendimiento bueno en el apartado de la consultas.

Pero claro hay que tener en cuanto que en cuanto lo que es rendimiento como se ha dicho no es la mejor solución, pero claro el rendimiento no es lo único que nos puede interesar buscar, y Oracle proporciona otros servicios que los otros gestores de datos que se han estudiado no proporcionan. Oracle es un importante gestor en el mundo de las bases de datos y nos proporcionan grandes ventajas en cuanto al manejo, seguridad, compatibilidades con otros gestores… Pero si es cierto que en cuanto a rendimiento se refiere tendría que mejorar.

- *Rendimiento durante las consultas realizadas:* en este último apartado Kowari y Mulgara han seguido obteniendo unos buenos resultados de rendimiento con resultados muy parecidos y cuyas diferencias de tiempos simplemente pueden deberse a la diferencia de un millón de datos insertados más en Mulgara que en Kowari. Respecto al uso de CPU de ambos gestores, es donde quizás donde los resultados obtenidos han sido peores que en los otros gestores.

  Sesame es el gestor que sin duda en este apartado ha conseguido los mejores resultados tanto en tiempo como en uso de CPU. Con esta pruebas se ha

demostrado el alto rendimiento de su modulo de consultas. Además destacar que Sesame contaba con 2 índices respecto a los 6 de Kowari y Mulgara y esto no ha hecho que obtuviera peores resultados que ellos. Dicho esto se puede decir que sería interesante que en un futuro Mulgara diera la posibilidad de elegir el numero de índices con los que se quiere trabajar para poder decidir uno mismo si van a ser necesario más o menos índices según el tipo de consultas que se quieren realizar, y de esta manera conseguir ahorrar esfuerzo en el mantenimiento de los índices, en espacio en disco…

También estas pruebas han servido para seguir corroborando que Oracle, al igual que ya paso en la inserción, ha sido la peor de las 4 bases de datos en este apartado, sobre todo en el tiempo de respuesta, siendo más del doble que en los otros gestores RDF, además de un uso de memoria muy elevado.

- *Conclusiones generales:* la primera conclusión de este proyecto es que se desaconseja utilizar Oracle para trabajar con tripletas RDF, siempre y cuando lo que se busque sea un buen rendimiento, ya que viendo los resultados obtenidos en la implantación de este gestor RDF, se ha visto que en lo referido a las experimentos realizados ha tenido un peor rendimiento que los otros gestores RDF.

  El gran problema de Oracle es que no es un gestor RDF pensado desde su diseño para trabajar con este tipo de datos, si no que es una base de datos relacional la cual ha añadido la funcionalidad de trabajar con tripletas RDF sobre la arquitectura y diseño de Oracle, y esto no ha tenido un buen resultado en cuanto a rendimiento se refiere., algo a priori inesperado al tratarse de una de las empresas más importantes en cuanto bases de datos se refiere. Oralce ha utilizado la capa de aplicaciones para implementar las funcionalidades para trabajar con RDF y ha utilizado una serie de tablas para almacenar las tripletas, mientras que el resto de gestores estudiados tienen su propia capa de almacenamiento especialmente diseñada para trabajar con este tipo de datos.

Investigando en foros de Internet se ha descubierto que la creadora de la capa de RDF para Oracle fue Susie Stephens, una bióloga que construyo esto como una herramienta para otro proyecto en el que estaba trabajando. Ella en principio no pensaba que este código iría destinado a una amplia distribución, solo pensaba que era algo puntual que funciono para un proyecto concreto. Así que ahora Oracle si desea mejorar en este apartado tendrá que plantearse el crean una capa de almacenamiento especifica si quiere estar a la altura de los otros gestores RDF [32].

Pero al mismo tiempo Oracle es un sistema completo y tiene todas las funciones necesarias como bases de datos, esto es quizás el punto en el que a lo mejor se tendría que considerar el uso de Oracle para trabajar con RDF. No tiene un gran rendimiento, pero cumple con todas las funciones de bases de datos, no es solo un repositorio para tripletas RDF, y esto según lo que se busque en el trabajo con RDF puede ser una ventaja en otros aspectos que no sea solo el rendimiento. Así que ahora solo cabe esperar que en futuras versiones mejore el rendimiento y de esta manera aprovechar el resto de buenas funcionalidades que tiene un gestor de bases de datos tan importante como Oracle.

Respecto a Kowari y Mulgara, se ha comprobado como poco a poco las nuevas funcionalidades que se están incorporando en Mulgara respecto a Kowari están surgiendo efecto y aportan nuevas funcionalidades interesantes. Como aspectos a mejorar respecto a los otros gestores RDF, sería útil el poder crear los índices que se consideraran necesarios y que estos no fueran impuestos a la hora de crear el modelo RDF donde se insertaran las tripletas. También sería interesante que se pudiera trabajar con lenguaje de consulta estándar como ocurre con Sesame donde se puede trabajar con SPARQL, como se ha podido ver en la página oficial de Mulgara [5] es algo en lo que ya se está trabajando y en la versión beta 2.0.6 de Mulgara ya se están haciendo pruebas.

Se podría decir que si se busca un buen rendimiento general el gestor más indicado sería Mulgara pero tiene una desventaja frente a Sesame, y que carece de ninguna capacidad de inferencia. Por lo que ante cualquier consulta SPARQL por ejemplo, únicamente se responderá con el conocimiento explícito de las

tripletas almacenada. Sesame sí permite generar tripletas inferidas en tiempo de carga a partir de la ontología definida.

Otro apartado importante es la posibilidad que ofrece Sesame para trabajar con algunos sistemas de gestión de bases de datos, como pueden ser MySQL y PostgreSQL. También como ya se comento antes se está trabajando en una beta para poder trabajar con la SAIL de Sesame en Oracle [31].

Se puede concluir con la idea de que actualmente Sesame es un gestor más maduro y ofrece más posibilidades que Mulgara, aunque no se descarta el uso de Mulgara por los grandes resultados de rendimiento obtenidos y el gran trabajo que están realizando sus desarrolladores para seguir mejorando este sistema.

En la siguiente tabla se muestra un breve de lo comentado anteriormente:

Tabla 5-1: Tabla Resumen

	Conclusión
Arquitectura	- Kowari, Mulgara y Sesame tienen capa de almacenamiento específica para trabajar con RDF, lo que es algo positivo frente a Oracle que no la tiene
Modelo de datos	- Kowari, Mulgara y Sesame menos coste trabajar con los datos (son gestores específicos para tratar con datos RDF) - Respecto a los índices mejor Oracle y Sesame por uso de arboles B, y por elegir el número de índices con los que se desea trabajar
Lenguaje de consulta	- Sesame es el único gestor de los estudiados que usa el lenguaje estándar recomendado por W3C (SPARQL)
Seguridad	- Oracle gestor al ser gestor de bases de datos cubre más aspectos en cuanto a seguridad se refiere.
Rendimiento inserción	- Mulgara menor tiempo en la inserción y buen uso de los recursos del sistema (memoria y CPU)
Rendimiento consultas	- Sesame mejor rendimiento en cuanto a tiempos y uso de recursos del sistema durante las consultas realizadas en los experimentos.

5.2 Conclusiones generales del proyecto

Después de meses de investigación, de aprendizaje, de implementación, de pruebas…, se han conseguido llevar a cabo los objetivos marcados en este proyecto. Para la realización de estos objetivos se ha realizado un aprendizaje desde cero sobre esta matería, ya que hasta antes de la realización de este proyecto mis ideas sobre la Web Semántica y sobre cuáles eran sus objetivos eran muy escasas, y desconocía por completo el significado y las utilidades de RDF.

Pero sobre todo donde he encontrado más problemas ha sido a la hora de la implementación de cada uno de los gestores RDF. Ya que cada uno de ellos era muy diferente del resto, cada uno con un lenguaje de consulta diferente que he tenido que aprender a manejar y además cada uno de ellos han presentado dificultades a la hora de la inserción de las tripletas RDF.

Aunque puedo decir que al final de este largo camino se han conseguido cumplir los objetivos que fueron marcados desde un principio:

- Estudiar el modelo de datos RDF.
- Análisis de la arquitectura, modelo de datos y seguridad de los gestores RDF llevado a estudio en este proyecto.
- Estudio de los lenguajes de consultas usados en los gestores RDF.
- Implementación de cada uno de los gestores e inserción en cada uno de ellos del fichero Infoboxes.
- Comparación del rendimiento de los Gestores RDF tanto en la inserción de datos como a la hora de realizar consultas sobre estos repositorios.

Además este proyecto pretende servir a aquellas personas que pretendan utilizar un Gestor RDF , como una guía la cual les permita saber que Gestor RDF de los estudiados puede servirles más para los objetivos de la actividad que quieran desarrollar con ellos, y ver qué ventajas y desventajas les puede aportar cada uno de ellos. Y de esta manera ahorrar y tiempo y esfuerzo a la hora de buscar un Gestor RDF con el cual trabajar en el campo de la Web semántica.

5.3 Líneas futuras

Una vez finalizado este proyecto es necesario realizar una pausa y evaluar los posibles desarrollos que se podrían realizar más adelante, teniendo como base este proyecto, ya que aunque se tratase de un estudio amplio sobre varios gestores RDF se pueden hacer ampliaciones de este estudio, pruebas conjuntas con otros sistemas, estudios de otros gestores… A continuación serán comentadas las reflexiones sobre los trabajos que se podrían realizar en un futuro:

- Una vez se tienen los datos almacenados en los gestores RDF de este proyecto sería interesante crear una aplicación (un programa en Java por ejemplo) para que trabajara con los repositorios creados en cada uno de los gestores, para ver como es la conectividad entre los gestores y la aplicación, que posibilidades existen a la hora de trabajar …

- Mulgara posee descriptores los cuales permiten usar las APIs de Mulgara de una forma alternativa que permiten incluir una serie de opciones muy interesantes, sobre todo a la hora de trabajar con otras aplicaciones (en la búsqueda de integrar Mulgara con otras aplicaciones):

- Sería interesante probar el rendimiento de algún otro gestor RDF existente en el mercado como puede ser Virtuoso. En un principio fue uno de los gestores pensados para este proyecto pero viendo que la versión para Windows, que es el sistema con el que se trabaja en este proyecto, era una versión de prueba para 15 días se decidió descartar de estudiar este gestor, ya que el tiempo era algo corto para poder realizar las pruebas requeridas. Además su uso esta mas bien pensado para trabajar con repositorios ya creados y utilizarlo como un sistema gestor, no tanto como una base de datos RDF propiamente dicho.

- También existen otra serie de herramientas como puede ser Jena, que permiten manipular metadatos desde una aplicación Java, incluyendo APIs para el soporte de RDF y el manejo de ontologías (soporte OWL como lenguaje de ontologías). Este tipo de sistemas están pensados para trabajar con modelo RDF bien definidos, desde una aplicación Java.

- Durante este proyecto se ha visto como algunos gestores RDF como puede ser Sesame pueden trabajar con gestores de bases de datos como pueden ser PostgreSQL y MySql. En un futuro sería interesante ver como se integra algún gestor RDF como Mulgara y Sesame con dichos gestores de bases de datos, y estudiar cómo se integran con ellos, ver que funcionalidades ganan y ver si el rendimiento sigue siendo el mismo.

- Como se comento anteriormente Kowari Y Mulgara utilizan arboles AVL que no son los más eficientes en cuanto al uso de disco se refiere, mientras que Sesame y Oralce utilizan arboles B que si lo son. Sería interesante que Kowari y Mulgara probaran o implementaran otro tipo de índices. (Kowari y Mulgara son los que gestores que tienen mayor uso de disco)

6. REFERENCIAS

- Tesis Especificación de Mecanismos para la Administración de un Mercado de Objetos de Aprendizaje. María Erika Olivos Contreras , UNIVERSIDAD AUTONOMA DE PUEBLA 2007
- Creating the semantic Web with RDF. Johan Hjelm, publicado en 2001
- Oracle database 10g: guia de aprendizaje. Ian Abramson, 2004
- Oracle 10g: administración y análisis de bases de datos, 2005
- Proyecto fin de carrera Diseño y gestión de modelos RDF. Almudena Abanda Hernández, Universidad Carlos III de Madrid 2006.
- Proyecto de fin de carrera Estudio y análisis del metalenguaje RDF: construcción de un repositorio para su almacenamiento. María Teresa Carcelén Fernández, Universidad Carlos III de Madrid.
- Apuntes de la asignatura *Ficheros y Bases de Datos.* Ingeniería Técnica en Informática de Gestión, Universidad Carlos III de Madrid. 2004.
- Apuntes de la asignatura *Diseño de Bases de Datos.* Ingeniería Técnica en Informática de Gestión, Universidad Carlos III de Madrid. 2005.

Enlaces

- 1. RDF: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/
- 2. Conceptos sobre RDF: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/
- 3. Página oficial Kowari: http://www.kowari.org/
- 4. Documentación sobre Kowari: http://kowari.org/oldsite/index.htm
- 5. Página oficial Mulgara: http://www.mulgara.org/
- 6. Página oficial Sesame: http://www.openrdf.org/

- 7. Consola de Sesame: http://confuseddevelopment.blogspot.com
- 8. Página oficial Oracle: http://www.oracle.com/global/es/index.html
- 9. Información general sobre Oracle: http://www.monografias.com/trabajos35/comparativa-bases-datos/comparativa-bases-datos3.shtml
- 10. Modelo RDF de Oracle y paquetes SDO_RDF: http://download.oracle.com/docs/cd/B19306_01/appdev.102/b19307/sdo_rdf_concepts.htm#CHDCEFIE
- 11. SQL: http://www.infor.uva.es/~jvegas/cursos/bd/sqlplus/sqlplus.html#sql*plus
- 12. SQL Loader: http://members.fortunecity.com/2horasdejazz/soporte/curso_loader.html

  http://www.orape.net/article19.html

  http://ugweb.cs.ualberta.ca/~c391/manual/chapt5.html
- 13. Blog (uno de los desarrolladores de Mulgara, Paul Gearon, (entrada 2 de diciembre, Dropping Indexes)): http://gearon.blogspot.com/
- 14. Documentación sobre Mulgara: http://docs.mulgara.org/
- 15. Wiki de Mulgara: http://www.mulgara.org/trac
- 16. Apache Tomcat 5.5:

  http://www.monografias.com/trabajos56/instalacion-tomcat/instalacion-tomcat.shtml.
- 17. Guía para desplegar aplicaciones Web en Apache Tomcat: http://www.programacion.com/java/articulo/desp_servlets/
- 18. Ayuda para maquina virtual de Java: http://www.consultoriajava.com/publico/JavaHeapSpace.shtml

  http://blogs.sun.com/watt/resource/jvm-options-list.html

- 19. SSL: http://es.wikipedia.org/wiki/Transport_Layer_Security
- 20. Manual usuario System Explorer: http://systemexplorer.wetpaint.com/page/System+Explorer+User+Reference+Guide
- 21. Buscadores semánticos:

  http://www.javi.it/semantic.html

  http://lcriadof.blogspot.com/2008/06/situacin-de-los-buscadores-semnticos.html
- 22. Web semántica: http://www.w3.org/2001/sw/
- 23. Dbpedia: http://wiki.dbpedia.org/OnlineAccess
- 24.Arquitectura Oracle: http://download.oracle.com/docs/cd/B19306_01/server.102/b14220/intro.htm#i6235
- 25.Arquitectura Oracle: http://alejandrotrujillo.wordpress.com/2008/10/07/arquitectura-de-oracle-10g/
- 26. Arbole AVL: http://www.tech-faq.com/lang/es/flat-file.shtml
- 27. Checklist Oracle: http://www.oracle.com/technology/deploy/security/pdf/twp_security_checklist_db_database.pdf
- 28.Cifrado Oracle http://www.oracle.com/technology/obe/10gr2_db_vmware/security/tde/tde.htm
- 29. Formatos exportación Sesame:

  http://www.cs.uns.edu.ar/~prf/teaching/FSW08/rdf1.pdf
- 30. Fichero tripletas RDF para experimentación:

  http://wiki.dbpedia.org/Downloads31
- 31. Beta Release of Sesame/openrdf SAIL for Oracle 10g http://www.oracle.com/technology/tech/semantic_technologies/sample_code/index.html

- 32. Entrada Oracle: http://gearon.blogspot.com/search?updated-min=2007-01-01T00%3A00%3A00-06%3A00&updated-max=2008-01-01T00%3A00%3A00-06%3A00&max-results=50
- 33. Formatos exportación Sesame:

  http://wiki.aduna-software.org/confluence/display/SESDOC/GettingStarted

7. Glosario

1. ***World Wide Web Consortium (W3C)*** es un consorcio que se dedica a producir estándares para la Web semántica.

2. ***XML*** es un lenguaje de marcas desarrollado por el W3C y permite definir la gramática de lenguajes específicos.

3. ***Framework*** es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Para entenderlo fácilmente diremos que es el esqueleto sobre el cual varios objetos son integrados para una solución dada.

4. ***Lucene*** es un API para recuperación de información de código abierto (Interfaz de Programación de Aplicaciones).

5. ***Metadatos*** literalmente «sobre datos», son datos que describen otros datos. En general, un grupo de metadatos se refiere a un grupo de datos, llamado recurso.

6. ***HTML*** lenguaje de Marcas de Hipertexto, es el lenguaje de marcado predominante para la construcción de páginas web.

7. ***DBF*** formato para almacenar datos estructurados.

8. ***URIs*** son cadenas que identifican algún recurso -como imágenes, documentos, archivos,…, para hacerlo disponible bajo una gran cantidad de esquemas como HTTP o FTP.

9. ***URL*** significa *Uniform Resource Locator*, es decir, localizador uniforme de recurso.

10. ***SQL*** Structured Q.uery Language, es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas.

11. ***Transnacional*** es una propiedad de aquellas bases de datos que en caso de que una operación de escritura no se completara con existo por algún motivo, podrían volver atrás.

12. ***API*** es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

13. ***SOFA*** API de Java para construir ontologías y bases de conocimiento en aplicaciones de Web semántica.

14. ***Java RDF (JRDF)*** implementación RDF basada en Java.

15. ***iTQL*** lenguaje de consulta interactivo.

16. ***RDQL*** lenguaje de consulta RDF.

17. ***SOAP*** que es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

18. ***SPARQL*** es un acrónimo recursivo del inglés SPARQL Protocol and RDF Query Language. Se trata de una recomendación para crear un lenguaje de consulta dentro de la Web semántica.

19. ***SERQL*** es un lenguaje de recuperación para RDF/RDFS desarrollado por Aduna como parte del software Sesame.

20. ***Rollback*** es una operación que devuelve a la base de datos a algún estado previo.

21. **SquishQL** lenguaje que permite realizar consultas con una sintaxis muy parecida a SQL.

22. **Ontología**: es el término ontología en informática hace referencia a la formulación de un exhaustivo y riguroso esquema conceptual dentro de un dominio dado, con la finalidad de facilitar la comunicación y la compartición de la información entre diferentes sistema

8. ANEXOS

8.1 ANEXO A: SINTAXIS DE CONSULTAS

En este anexo esta la sintaxis de las consultas realizadas para el rendimiento de las consultas en los gestores RDF de este proyecto. Se mostrara la sintaxis de las consultas para los tres lenguajes de consulta necesarios en este proyecto: SQL (Oralce), ITQL (Kowari y Mulgara) y SERQL (Sesame).

CF1:

SERQL: SELECT A FROM {<http://dbpedia.org/resource/Philadelphia_mayoral_election%2C_1999>} <http://dbpedia.org/property/votes> {A}

SQL: select m from table(sdo_rdf_match('(b:Philadelphia_mayoral_election%2C_1999 a:votes?m)',sdo_rdf_models('modelo_rdf'),null,SDO_RDF_Aliases(SDO_RDF_Alias('b','http://dbpedia.org/resource/'),SDO_RDF_Alias('a','http://dbpedia.org/property/')),null));

ITQL: select $x from <rmi://HP21997209933/server1#modelo_rdf>where <http://dbpedia.org/resource/Philadelphia_mayoral_election%2C_1999> <http://dbpedia.org/property/votes> $x;

CF2:

SERQL: SELECT A FROM {<http://dbpedia.org/resource/The_City_of_the_Dead_%28book%29>} <http://dbpedia.org/property/pages> {A}

SQL: select m from table(sdo_rdf_match('(b:The_City_of_the_Dead_%28book%29 a:pages?m)',sdo_rdf_models('modelo_rdf'),null,SDO_RDF_Aliases(SDO_RDF_Alias('b','http://dbpedia.org/resource/'),SDO_RDF_Alias('a','http://dbpedia.org/property/')),null));

ITQL: select $x from <rmi://HP21997209933/server1#modelo_rdf> where <http://dbpedia.org/resource/The_City_of_the_Dead_%28book%29> <http://dbpedia.org/property/pages> $x;

CF3:

SERQL: SELECT A FROM {<http://dbpedia.org/resource/The_City_of_the_Dead_%28book%29>} <http://dbpedia.org/property/writer> {A}

SQL: select m from table(sdo_rdf_match('(b: The_City_of_the_Dead_%28book%29 a:writer?m)',sdo_rdf_models('modelo_rdf'),null,SDO_RDF_Aliases(SDO_RDF_Alias('b','http://dbpedia.org/resource/'),SDO_RDF_Alias('a','http://dbpedia.org/property/')),null));

ITQL: select $x from <rmi://HP21997209933/server1#modelo_rdf>where <http://dbpedia.org/resource/The_City_of_the_Dead_%28book%29> <http://dbpedia.org/property/writer> $x;

CF4:

SERQL: SELECT A FROM {<http://dbpedia.org/resource/Trichuris>} <http://dbpedia.org/property/subdivision> {A}

SQL: select m from table(sdo_rdf_match('(b:Trichuris a:subdivision ?m)',sdo_rdf_models('modelo_rdf'),null,SDO_RDF_Aliases(SDO_RDF_Alias('b','http://dbpedia.org/resource/'),SDO_RDF_Alias('a','http://dbpedia.org/property/')),null));

ITQL: select $x from <rmi://HP21997209933/server1#modelo_rdf> where <http://dbpedia.org/resource/Trichuris> <http://dbpedia.org/property/subdivision> $x;

CF5:

SERQL: SELECT A FROM {<http://dbpedia.org/resource/Hristo_Arangelov>} <http://dbpedia.org/property/clubs> {A}

SQL: select m from table(sdo_rdf_match('(b:Hristo_Arangelov a:clubs ?m)',sdo_rdf_models('modelo_rdf'),null,SDO_RDF_Aliases(SDO_RDF_Alias('b','http://dbpedia.org/resource/'),SDO_RDF_Alias('a','http://dbpedia.org/property/')),null));

ITQL: select $x from <rmi://HP21997209933/server1#modelo_rdf>where <http://dbpedia.org/resource/Hristo_Arangelov> <http://dbpedia.org/property/clubs> $x;

CF6:

SERQL: SELECT A FROM {<http://dbpedia.org/resource/Amnesty_law>} <http://dbpedia.org/property/seeProperty> {A}

SQL: select m from table(sdo_rdf_match('(b:Amnesty_law a:seeProperty?m)',sdo_rdf_models('modelo_rdf'),null,SDO_RDF_Aliases(SDO_RDF_Alias('b','http://dbpedia.org/resource/'),SDO_RDF_Alias('a','http://dbpedia.org/property/')),

ITQL: select $x from <rmi://HP21997209933/server1#modelo_rdf> where <http://dbpedia.org/resource/Amnesty_law> <http://dbpedia.org/property/seeProperty> $x;

CF7:

SERQL: SELECT A FROM {A} <http://dbpedia.org/property/platforms> {<http://dbpedia.org/resource/Personal_computer>}

SQL: select m from table(sdo_rdf_match('(?m b:platforms a:Personal_computer)',sdo_rdf_models('modelo_rdf'),null,SDO_RDF_Aliases(SDO_RDF_Alias('b','http://dbpedia.org/property/'),SDO_RDF_Alias('a','http://dbpedia.org/resource/')),null));

ITQL: select $x from <rmi://HP21997209933/server1#modelo_rdf> where $x> <http://dbpedia.org/property/platforms> <http://dbpedia.org/resource/Personal_computer>;

CF8:

SERQL: SELECT A FROM {A} <http://dbpedia.org/property/position> {<http://dbpedia.org/resource/Striker_%28football%29> }

SQL: select m from table(sdo_rdf_match('(?m b:position a: Striker_%28football%29)',sdo_rdf_models('modelo_rdf'),null,SDO_RDF_Aliases(SDO_RDF_Alias('b','http://dbpedia.org/property/'),SDO_RDF_Alias('a','http://dbpedia.org/resource/')),null));

ITQL: select $x from <rmi://HP21997209933/server1#modelo_rdf> where $x> <http://dbpedia.org/property/position> <http://dbpedia.org/resource/Striker_%28football%29>;

CF9:

SERQL: SELECT A FROM {A} < http://dbpedia.org/property/operatingSystem> {< http://dbpedia.org/resource/Linux>}

SQL: select m from table(sdo_rdf_match('(?m b: operatingSystem a:Linux)',sdo_rdf_models('modelo_rdf'),null,SDO_RDF_Aliases(SDO_RDF_Alias('b','http://dbpedia.org/property/'),SDO_RDF_Alias('a','http://dbpedia.org/resource/')),null));

ITQL: select $x from <rmi://HP21997209933/server1#modelo_rdf> where $x> < http://dbpedia.org/property/operatingSystem> < http://dbpedia.org/resource/Linux>;

CF10:

SERQL: SELECT A, C FROM {A} <http://dbpedia.org/property/platforms> {C}

SQL: select m from table(sdo_rdf_match('(?n a:platforms ?m)',sdo_rdf_models('modelo_rdf'),null,SDO_RDF_Aliases(SDO_RDF_Alias('a','http://dbpedia.org/property/')),null));

ITQL: select $x $y from <rmi://HP21997209933/server1#modelo_rdf> where $x <http://dbpedia.org/property/platforms> $y;

CF11:

SERQL: SELECT A, C FROM {A} < http://dbpedia.org/property/managerclubs > {C}

SQL: select m from table(sdo_rdf_match('(?n a:managerclubs ?m)',sdo_rdf_models('modelo_rdf'),null,SDO_RDF_Aliases(SDO_RDF_Alias('a','http://dbpedia.org/property/')),null));

ITQL: select $x $y from <rmi://HP21997209933/server1#modelo_rdf> where $x < http://dbpedia.org/property/managerclubs > $y;

CF12:

SERQL: SELECT A, C FROM {A} < http://dbpedia.org/property/operatingSystem > {C}

SQL: select m from table(sdo_rdf_match('(?n a:operatingSystem ?m)',sdo_rdf_models('modelo_rdf'),null,SDO_RDF_Aliases(SDO_RDF_Alias('a','http://dbpedia.org/property/')),null));

ITQL: select $x $y from <rmi://HP21997209933/server1#modelo_rdf> where $x < http://dbpedia.org/property/operatingSystem > $y;

CF13:

SERQL: SELECT A FROM {A} <http://dbpedia.org/property/seeProperty> {"Ley de Punto Final"@en.}

SQL: select m from table(sdo_rdf_match('(b: 5605_%28Hebrew_year%29 a: monthHebrewCalendarProperty?m)',sdo_rdf_models('modelo_rdf'),null,SDO_RDF_Aliases(SDO_RDF_Alias('b','http://dbpedia.org/resource/'),SDO_RDF_Alias('a','http://dbpedia.org/property/')),null));

ITQL: Select $x from <rmi://HP21997209933/server1#modelo_rdf> where $x <http://dbpedia.org/property/seeProperty> ‘Ley de Punto Final’@en;

Nota: En Kowari se tendría que quitar el @en del objeto para que la consulta fuera.

CF14:

SERQL: SELECT A FROM {A} http://dbpedia.org/property/deathPlace {<http://dbpedia.org/resource/Massachusetts>}

SQL: select m from table(sdo_rdf_match('(b: Massachusetts a: deathPlace ?m)',sdo_rdf_models('modelo_rdf'),null,SDO_RDF_Aliases(SDO_RDF_Alias('b','http://dbpedia.org/resource/'),SDO_RDF_Alias('a','http://dbpedia.org/property/')),null));

ITQL: Select $x from <rmi://HP21997209933/server1#modelo_rdf> where $x <http://dbpedia.org/property/deathPlace> <http://dbpedia.org/resource/Massachusetts>;

CF15:

SERQL: SELECT A FROM {A} <http://dbpedia.org/property/series> {"Heroes"@en.}

SQL: select m from table(sdo_rdf_match('(b: 5605_%28Hebrew_year%29 a: monthHebrewCalendarProperty?m)',sdo_rdf_models('modelo_rdf'),null,SDO_RDF_Aliases(SDO_RDF_Alias('b','http://dbpedia.org/resource/'),SDO_RDF_Alias('a','http://dbpedia.org/property/')),null));

ITQL: select $x from <rmi://HP21997209933/server1#modelo_rdf> where $x <http://dbpedia.org/property/series> ‘"Heroes"@en;

CF16:

SERQL**:** SELECT A FROM {<http://dbpedia.org/resource/Philadelphia_mayoral_election%2C_1999>} A {"Democratic Party (US)"@en }

SQL:

ITQL: select $p from <rmi://HP21997209933/server1#modelo_rdf> where <http://dbpedia.org/resource/Philadelphia_mayoral_election%2C_1999> $p 'Democratic Party (US)'@en;

CF17:

SERQL: SELECT A FROM
{<http://dbpedia.org/resource/The_City_of_the_Dead_%28book%29>} A
{<http://dbpedia.org/resource/Lloyd_Rose>}

SQL:

ITQL: select $x from <rmi://HP21997209933/server1#modelo_rdf>where
<http://dbpedia.org/resource/The_City_of_the_Dead_%28book%29> $x
<http://dbpedia.org/resource/Lloyd_Rose>;

CF18:

SERQL**:** SELECT A FROM {<
http://dbpedia.org/resource/Philibert_Francois_Rouxel_de_Blanchelande >} A
{<http://dbpedia.org/resource/Dijon> }

SQL:

ITQL: select $p from <rmi://HP21997209933/server1#modelo_rdf> where <
http://dbpedia.org/resource/Philibert_Francois_Rouxel_de_Blanchelande > $p
<http://dbpedia.org/resource/Dijon>;

8.2 ANEXO B: HERRAMIENTAS TOMA DE DATOS DE RENDIMIENTOS

- Oracle Enterprise Manager:

Es una herramienta Web que nos proporciona Oracle, mediante la cual se puede administrar toda la base de datos y tener control sobre ella. Entre sus utilidades están poder administrar las bases de datos de las que se disponga, ver información de la base de datos (espacio, tablas,…), gestión de backups y la parte interesante para el proyecto que es la parte relacionada con el rendimiento e la base de datos. En la ventana principal de esta herramienta existe una pestaña que pone rendimiento que es lugar donde se encuentra toda la información relacionada con el rendimiento de la base de datos. En este apartado se pueden ver mediante gráficas y datos numéricos el uso de CPU, el uso de memoria, entradas y salidas, procesos, usuarios conectados,… , además se pondrá indicar el período de tiempo del cual se quiere ver estos datos.

Ilustración 0-1: Rendimiento de Oracle

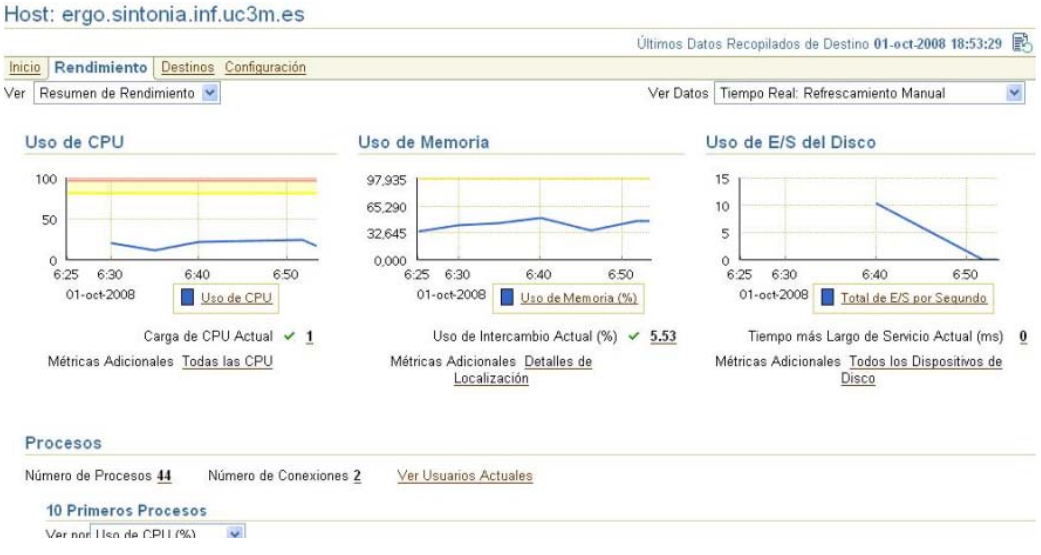
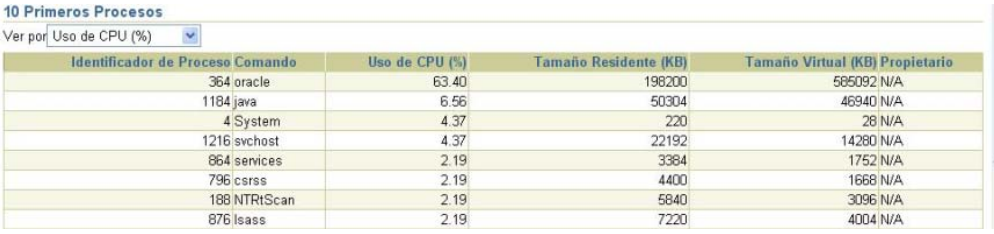


Ilustración 0-2: datos uso CPU Oracle



10 Primeros Procesos

Ver por Uso de CPU (%)

Identificador de Proceso	Comando	Uso de CPU (%)	Tamaño Residente (KB)	Tamaño Virtual (KB)	Propietario
364	oracle	63.40	198200	585092	N/A
1184	java	6.56	50304	46940	N/A
4	System	4.37	220	28	N/A
1216	svchost	4.37	22192	14280	N/A
864	services	2.19	3384	1752	N/A
796	csrss	2.19	4400	1668	N/A
188	NTRtScan	2.19	5840	3096	N/A
876	lsass	2.19	7220	4004	N/A

Si se quiere información más concreta de cada uno de ellos bastara con pulsar en ellos y se ampliara esta información como se puede ver en la siguiente ilustración:

Ilustración 0-3: gráfica Uso CPU

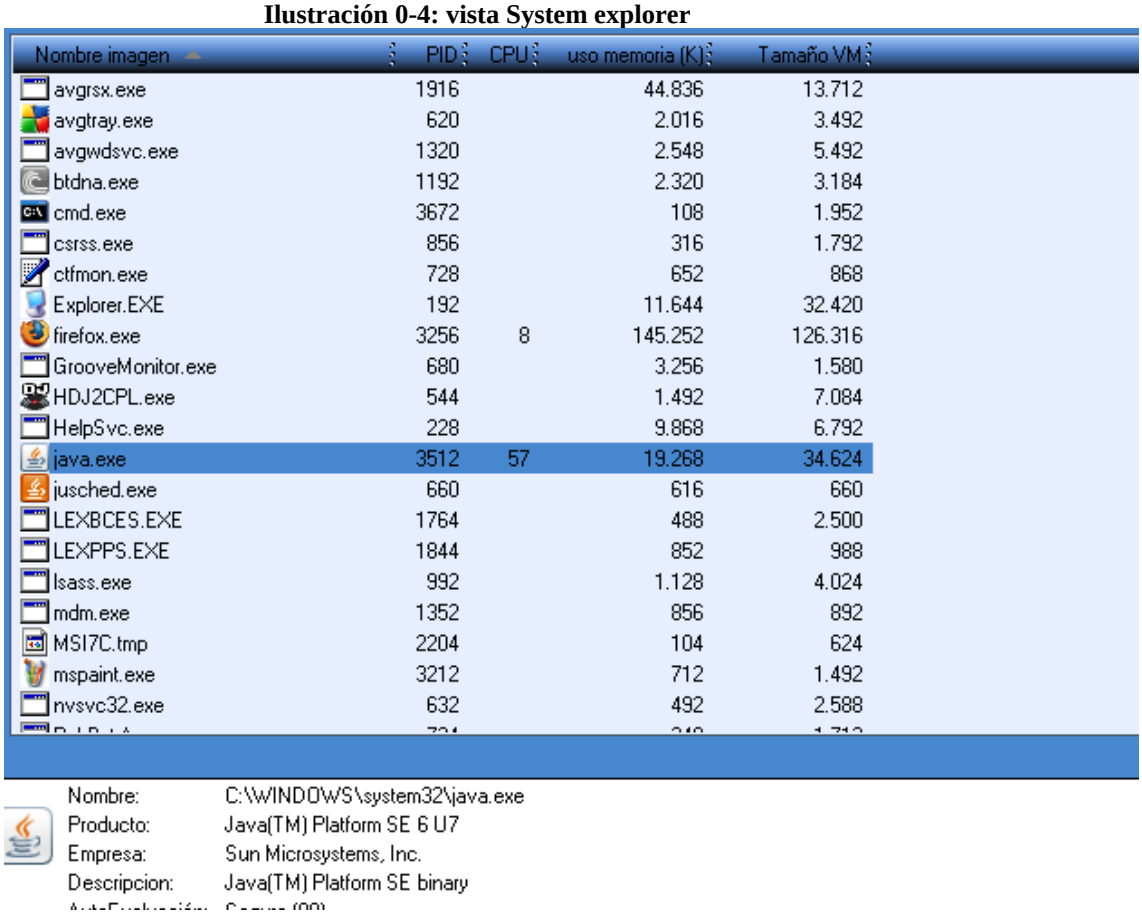
Además existe la posibilidad de generar informes más detallas de la actividad de la base de datos durante un periodo de tiempo en el que se pueden ver con más detalle aspectos relevantes al rendimiento, como pueden ser las operaciones que se han realizado, que tiempo ha requerido cada una y uso de CPU… Estos informes son generados en HTML.

- System Explorer

Esta es la herramienta que ha permitido tomar la captura de datos de rendimiento para Kowari, Mulgara y Sesame. System Explorer permite ver información detallada de cada uno del los procesos que existen en una computadora proporcionando información detallada de cada uno de ellos además de gráficas en tiempo real que muestran multitud de información tanto del procesador como de un determinado proyecto.

Para iniciar la toma de rendimientos de datos lo primero que hay que hacer es conocer cuál es el proceso que queremos medir, por ejemplo en el caso del Mulgara es un proceso java, al cual se le asigna un identificador de proceso (PID) y del cual se dispone una cierta información como se muestra en la siguiente captura:

Ilustración 0-4: vista System explorer

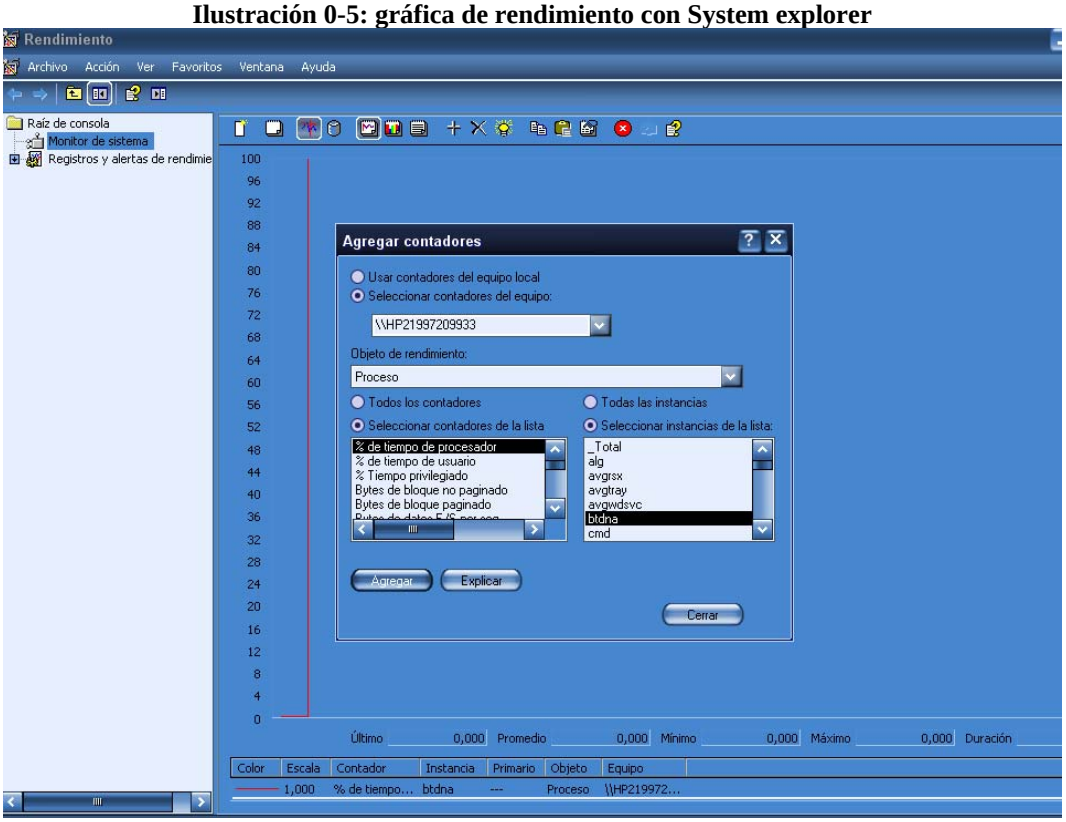


El proceso marcado con azul sería el correspondiente al gestor RDF que en ese momento se está utilizando (en este caso Mulgara), de este proceso se muestra el uso de CPU que tiene y el uso de memoria que necesita.

Además dispone de una utilidad llamada “monitor de rendimiento” el cual permite mostrar mediante una gráfica las propiedades de rendimiento del procesador o determinadas propiedades de un proceso que se está ejecutando. Los pasos que se han utilizada para crear la gráfica de rendimiento usada en este proyecto son los siguientes:

- Abrir monitor de rendimiento
- Crear un nuevo conjunto de contadores
- Agregar contadores

Ilustración 0-5: gráfica de rendimiento con System explorer



- Seleccionar el proceso que se quiere medir e ir agregando los contadores que se deseen, como por ejemplo tiempo de procesador y número de operaciones de lectura de E/S por segundo, que fueron los que se utilizaron en el apartado de las consultas.

8.3 ANEXO C: SQL Loader

SQL Loader es una importante utilidad que permite cargar datos externos (en ficheros de texto por general) en tablas de una base de datos en Oracle. La carga de datos en las tablas implica que estas tendrán que estar creadas con antelación en la base de datos y podrán tener o no datos insertados con anterioridad. También disponemos de la opción de que los datos que vamos a insertan sustituyan a los datos ya existentes en la tabla o bien añadirlos a los ya insertados.

Es posible la carga de datos almacenados en forma de ficheros de texto (lo más corriente) o binarios. La entrada del programa consiste en:

- Uno o varios ficheros de datos, con nombres por defecto terminados en .DAT, conteniendo los datos a importar.
- Un fichero (texto) de control, con nombre por defecto terminado en .CTL, que contiene órdenes que permiten guiar y particularizar el proceso de carga de datos. En este fichero se especifican los atributos de las tablas de la base de datos en los que se van a insertar los valores contenidos en el fichero de datos.

Como salida se generan hasta tres ficheros de texto:

- Un fichero de resultados, con nombre por defecto terminado en .LOG, que contiene diversos informes sobre la realización del proceso de carga de datos (tiempo d inserción, datos insertados,…).
- Un fichero de errores, con nombre por defecto terminado en .BAD, que contiene aquellos datos del fichero de entrada que no han podido ser cargados en la base de datos por diversos errores. Si la carga se ha realizado sin errores entonces no se genera este fichero.
- Un fichero de descartados, con nombre por defecto terminado en .DSC, que contiene aquellos datos del fichero de entrada que no han sido cargados en la base de datos porque así se había especificado en el fichero de control. Si no se ha descartado ningún dato durante el proceso de carga no se genera este

La diferencia entre los datos erróneos y los descartados es que los primeros son aquellos datos que no han sido insertados en las tablas de la base de datos porque no han

podido ser leídos correctamente desde el fichero de datos de entrada o bien su inserción causa errores de incumplimiento de restricciones definidas en las tablas, mientras que los datos descartados no se insertan en la base de datos porque no verifican una determinada condición que puede imponerse en el fichero de control, de manera que tan solo se inserten los datos que satisfacen dicha condición

Para ejecutar SQL*Loader se necesita especificar el nombre y palabra clave en el sistema Oracle de un usuario (usualmente el propietario) que disponga de permiso de inserción (INSERT) sobre las tablas en las que se van a cargar los datos. La invocación sería algo como:

Ilustración 0-6: SQL Loader

El parámetro “errors” que vemos en la línea de arriba nos indica el número de fallos que están permitidos en la inserción. Una vez ejecutemos esta línea, la aplicación nos pedirá el nombre y la contraseña de un usuario que como se ha dicho antes tendrá que tener permisos en la base de datos para hacer esta operación.

Ya en el apartado de implementación se mostraran los ficheros de control utilizados para la inserción de los datos utilizados en el proyecto.