Licencia

UNIVERSIDAD OBERTA DE CATALUNYA

Ingeniería Informática

Desarrollo de un índice espacial para la extensión JASPA sobre H2

Alumno: José Antonio Calvillo Ardila

Dirigido por: Jesús de Diego Alarcón

CURSO 2010-11 (Septiembre/Febrero)

Índice de contenido

Índice de ilustraciones

Índice de tablas

Resumen

El presente trabajo tiene por objetivo ofrecer una solución para la creación de un índice espacial para la extensión *JASPA* (*Java SPAtial*) sobre la base de datos *H2*. El algoritmo de indexación elegido para la implementación del índice espacial ha sido el *Rtree*.

La implementación se ha realizado con el lenguaje de programación *Java* lo que ha facilitado su integración con la extensión *JASPA* y la base de datos *H2*, dado que ambos proyectos están codificados en el mismo lenguaje. El índice es persistente en memoria secundaria en una tabla de la propia base de datos H2.

La solución que se propone está limitada a operaciones espaciales en dos dimensiones y es lo suficientemente flexible como para que no se haya necesitado modificar ni el código fuente de *JASPA*, ni de *H2*. Además, se ha previsto que el algoritmo de indexación se pueda mejorar o sustituir fácilmente.

Por último, se ha tenido en cuenta que el proceso de creación y manipulación de los índices espaciales sea intuitivo y fácil de usar.

Abstract

This paper aims to provide a solution for creating a spatial index for the extension *JASPA* (*Java Spatial*) on the *H2* database. The indexing algorithm chosen to implement the spatial index has been the *RTree*.

The implementation was done with the Java programming language which has facilitated its integration with the extension *JASPA* database and *H2*, since both projects are coded in the same language. The index is persistent in secondary memory in a table in the database itself *H2*.

The proposed solution is limited to two-dimensional space operations and is flexible enough to not have needed to modify or *JASPA* source code, or *H2*. In addition, it is expected that the indexing algorithm can easily upgrade or replace.

Finally, we have taken into account that the process of creation and manipulation of spatial indexes is intuitive and easy to use.

1. Introducción.

Este trabajo representa la memoria final del Proyecto de Fin de Carrera (en adelante PFC) titulado *Desarrollo de un índice espacial para la extensión JASPA sobre H2*. *Ingeniería Informática (Universitat Oberta de Cataluña).*

En este proyecto se implementa un índice espacial en la extensión *JASPA*[1] (*JAva SPAtial*) para acelerar las operaciones sobre campos con información espacial en la base de datos *H2*[2].

Los objetivos del proyecto son:

- Profundizar en las características de la información geográfica y las necesidades derivadas de su almacenamiento y explotación en bases de datos, analizando el conjunto de soluciones de software libre de interés para tal fin.

- Estudiar la arquitectura común de las extensiones espaciales de las bases de datos más extendidas evaluando la importancia de los diferentes objetos que dan soporte a las funciones de análisis geográfico.

- Implementar una solución tecnológica que permita crear un índice espacial sobre la base de datos H2 a partir de las diferentes metodologías para la indexación de datos espaciales.

- Dar soporte a las funcionalidades espaciales implementadas en JASPA.

Este documento se ha dividido en diferentes capítulos, en el primer capítulo se realiza una introducción al PFC. Posteriormente en el segundo capítulo se exponen algunos conceptos teóricos básicos que servirán para comprender la solución final.

A continuación, en el tercer capítulo se describen las principales características de la extensión JASPA y H2.

Más adelante en el quinto capítulo se indica el análisis, diseño e implementación de la aplicación realizada. Por último, en el capítulo sexto se realiza un conjunto de pruebas y en el capítulo séptimo se recogen las conclusiones de este trabajo.

1 JASPA: http://jaspa.forge.osor.eu/
2 H2: http://www.h2database.com/

2. Conceptos básicos.

En esta primera parte se describen los conocimientos teóricos que servirán de base para realizar la implementación del proyecto. En concreto, este apartado se compone de una introducción a los Sistemas de Información Geográfica (en adelante SIG), una descripción básica del Estándar *OpenGIS* y de una introducción a las Bases de datos geográficas e Indexación de datos geográficos.

2.1 Introducción a los Sistemas de Información Geográfica.

Una definición de Sistema de Información Geográfica bastante aceptada es la redactada por el *NCGIA* (*National Centre of Geographic Information and Analysis*):

> *Un SIG es un sistema de hardware, software y procedimientos elaborados para facilitar la obtención, gestión, manipulación, análisis, modelado, representación y salida de datos espacialmente referenciados, para resolver problemas complejos de planificación y gestión [1].*
> *(http://gis.washington.edu/phurvitz/professional/SSI/index.html)*

En definitiva, un SIG es una aplicación informática basada en un gestor de base de datos con herramientas especializadas en el manejo de información alfanumérica e información geográfica capaces de facilitar su análisis en el espacio.

Se define información geográfica como aquellos datos que están localizados en el espacio, respecto de algún sistema de referencia.

Un SIG puede gestionar dos tipos de datos geográficos [2] (véase la figura 1) para representar la realidad:

- *Datos vectoriales*: utilizan tres tipos de elementos geométricos básicos (puntos, líneas y polígonos, también llamados primitivas geográficas) para modelizar la realidad. Se usan vectores definidos por pares de coordenadas relativas a algún sistema cartográfico para representar los objetos geográficos. Por ejemplo fincas, carreteras, etc.

- *Datos Raster*: utilizan redes regulares (mallas o grids) para modelizar la realidad.

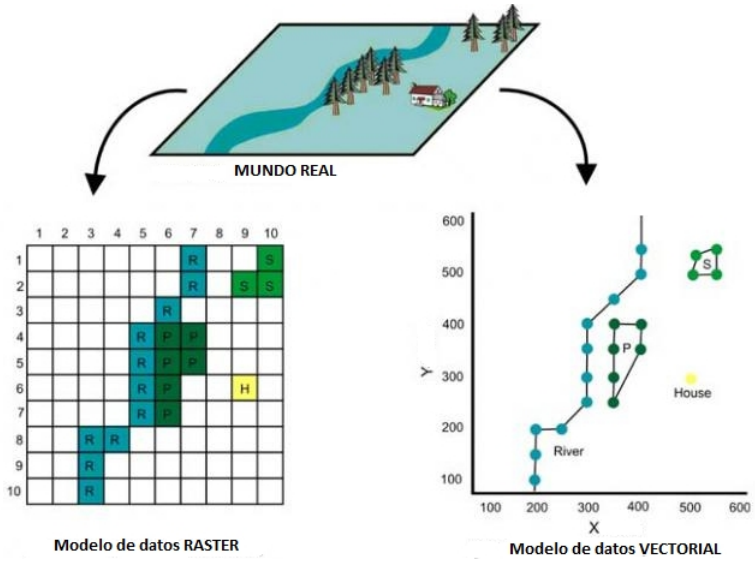

Figura 1: Imagen ilustrativa del Modelo Raster y Vectorial

Una característica natural de un SIG es la capacidad de realizar análisis espacial [3]. A continuación, se describen las principales herramientas que le permiten realizar este tipo de análisis:

- **Superposición**: es la función más básica de los SIG y está basada en la anteposición de planos. Permite realizar el solapamiento de capas de información derivadas de una amplia variedad de operaciones matemáticas.

- **Determinación de áreas de influencia**: se conocen como áreas de influencia aquellas que a partir de una entidad espacial y de acuerdo a una variable o conjunto de variables define una nueva entidad en el espacio.

- **Análisis de distancia, proximidad y vecindades**: Permite conocer cómo se relaciona un objeto geográfico con su entorno y viceversa,

- **Análisis de redes**: se caracteriza el terreno en redes de nodos y líneas. Se emplean para calcular recorridos óptimos, cambios en el tráfico, variaciones de tensión eléctrica, ...

- **Análisis Raster**: permite obtener perfiles o cortes y realizar análisis de visibilidad.

- **Geocodificación**: proporciona un procedimiento para la localización geográfica de un objeto de interés a través de los valores de sus atributos.

- **Análisis estadístico**: provee una serie de técnicas matemáticas para el estudio e interpretación de datos por áreas, variables, etc

Antes de analizar cómo se almacena la información geográfica en el apartado siguiente, se introducirá el estándar fijado por el *Open Geospatial Consortium Inc.* (*OGC*), donde se define la estructura de datos que debe tener un Sistema gestor de base de datos (*SGBD*) espacial.

El análisis de este estándar contribuirá a comprender en apartados posteriores cómo ampliar este modelo de datos para incluir un índice espacial.

2.2 Bases de datos geográficas

Las bases de datos que han de almacenar información geográfica, han de almacenar dos tipos generales de datos:

- Geográficos
- Alfanuméricos

Una base de datos relacional presenta deficiencias para manejar objetos de tipo geométrico [4], por ejemplo no puede efectuar una indexación adecuada de datos espaciales. Para cubrir estas necesidades se definen nuevas estructuras de almacenamiento para datos espaciales, de forma que sea posible almacenar y utilizar datos espaciales en bases de datos relacionales, como por ejemplo las estructuras del estándar OpenGIS.

Además del almacenamiento, la base de datos geográficos debe permitir realizar consultas sobre los datos espaciales, modificaciones y la exportación e importación de los datos.

2.3 Estándar OpenGIS

Los estándares son convenios eficazmente formalizados y documentados que contienen especificaciones técnicas o criterios precisos para que sean usados como reglas o guías y asegurar que los productos y sus servicios cumplen con su propósito.

Dentro del área de las tecnologías de información espacial, esta estandarización de la geoinformación es un asunto muy importante ya que pretende crear un mecanismo que permita ahorrar tiempo y dinero, al facilitar complejas migraciones evitando la incompatibilidad entre datos.

Las especificaciones *OpenGIS®* son creadas y difundidas por el *Open Geospatial Consortium Inc.* (*OGC*) [5], un conjunto de más de 200 organizaciones públicas y privadas, dedicadas a normalizar la geoinformación y unificar criterios para su almacenamiento, publicación y transferencia.

OpenGIS establece dos formatos de almacenamiento de objetos geométricos, en ambos casos se guarda la misma información: el tipo de objeto y sus coordenadas:

- *Well-Known Text Representation for Geometry (WKT)*: Representa o codifica objetos espaciales en formato vectorial, es decir, en texto plano.

- *Well-Known Binary Representation for Geometry* (*WKB*): Representa o codifica objetos espaciales en formato binario, es decir, en una secuencia de bytes en formato hexadecimal.

Los estándares *OpenGIS*, describen multitud de servicios y formatos de datos geoespaciales, a continuación se indican los más conocidos:

- *GML* (*Geography Markup Language*): Lenguaje basado en XML mediante el cual se establece el estándar para modelar, transferir y almacenar información de contenido Geográfico.

- *WMS* (*Web Map Service*): Especificaciones que estandarizan el envío y recepción de información Geoespacial, a través de imágenes (PNG, JPG, GIF entre otros).
- *WFS* (*Web Feature Service*): Especificaciones de implementación que permiten obtener y actualizar datos geoespaciales desde una base de datos a través de Internet.

- *OpenLS* (*Open Location Service*): Especificaciones para Servicios de Localización.

Entre las especificaciones OpenGIS están las dedicadas a la gestión de datos espaciales en bases de datos relacionales bajo el lenguaje SQL. El estándar se subdivide en dos partes: una que define la arquitectura de los datos y otra específica para SQL.

En definitiva, *OpenGIS Simple Features Access – Part 2: SQL Option*[3] describe un subconjunto de tipos de geometría SQL junto con funciones SQL sobre estos tipos. Las especificación para SQL de características simples de OpenGIS define tipos de objetos GIS estándar, los cuales son manipulados por funciones, y un conjunto de tablas de metadatos basadas en los tipos de datos predefinidos.

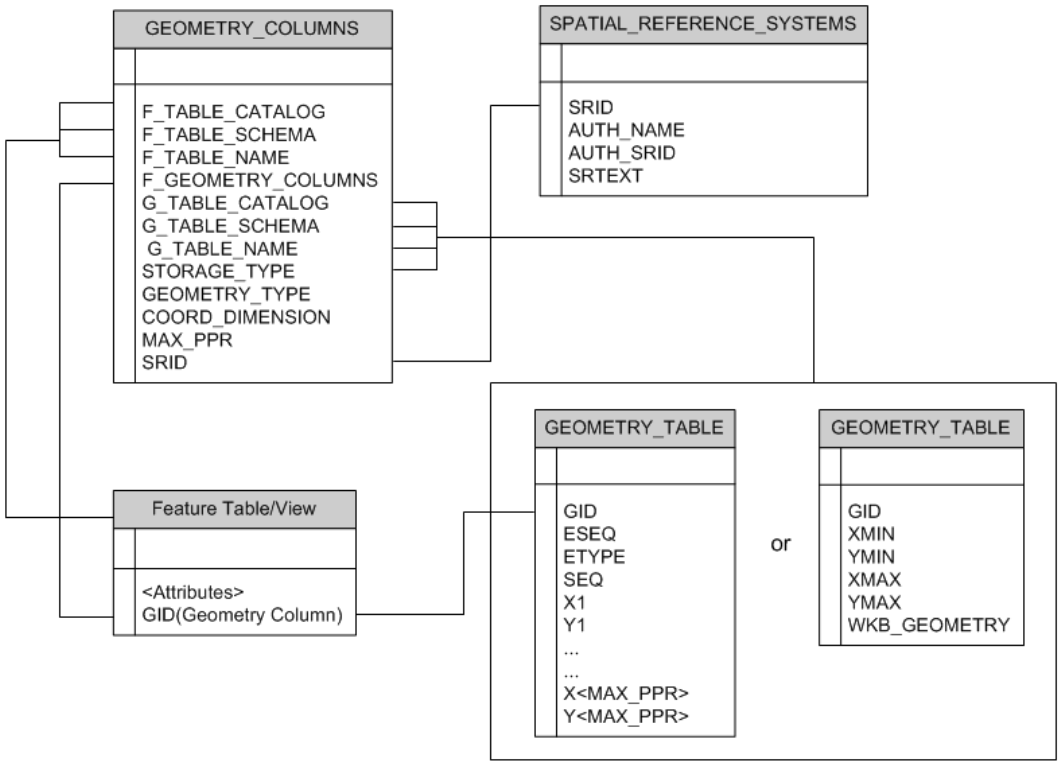

Figura 2: Estructura de tablas de la base de datos OpenGIS

A continuación se indican las tablas definidas por este estándar (véase la figura 2).

3 http://www.opengeospatial.org/standards/sfs

a) La tabla *GEOMETRY_COLUMNS* describe las tablas que contienen algún atributo de tipo geométrico o espacial.

b) La tabla *SPATIAL_REF_SYS* describe el sistema de coordenadas y de las transformaciones geométricas.

c) La tabla *FEATURE_TABLE* almacena los atributos de los objetos. Esta tabla contiene una clave foránea (*Geometry_Column*) que la relaciona con la tabla geométrica.

d) La tabla *GEOMETRY_TABLE* almacena los datos geométricos de los objetos.

A continuación, se indican la estructura de información de las tablas: *SPATIAL_REF_SYS* y *GEOMETRY_COLUMNS*.

SPATIAL_REFERENCES_SYSTEMS

Se compone de los siguientes campos:

- *SRID*: Valor entero que identifica el sistema de referencia espacial.

- *AUTH_NAME*: El nombre del estándar para el sistema de referencia. Por ejemplo: *EPSG* (*European Petroleum Survey Group*) que es un sistema de referencia (sistemas de coordenadas, proyecciones cartográficas...) utilizado en la representación cartográfica.

- *AUTH_SRID*: El identificador según el estándar AUTH_NAME. En el ejemplo anterior es el id según EPSG.

- *SRTEXT*: Almacena los datos en formato *WKT* del Sistema de Referencia Espacial.

GEOMETRY_COLUMNS

Está compuesta por los siguientes campos:

- *F_TABLE_CATALOG,F_TABLE_SCHEMA,F_TABLE_NAME*: Almacena el catálogo, esquema y el nombre de la tabla que contiene los atributos de las geometrías.

- *F_GEOMETRY_COLUMN*: Nombre de la columna geométrica que asocia la información geográfica con los atributos.

- *G_TABLE_CATALOG, G_TABLE_SCHEMA, G_TABLE_NAME*: Almacena el catálogo, esquema y nombre de la tabla que contiene la información geográfica.

- *STORAGE_TYPE*: Contiene el tipo de almacenamiento.

- *GEOMETRY_TYPE*: Indica el tipo de geometría (point, linestring, poligon, ...)

- *COORD_DIMENSION*: Almacena el número de dimensiones espaciales de la columna (2D o 3D o 3D+Capa).

- *MAX_PPR*: Contiene el máximo número de puntos por fila.

- *SRID*: Es una clave foránea que referencia *SPATIAL_REF_SYS*, contiene el identificador del sistema geográfico utilizado.

2.4 Indexación de datos geográficos

Un índice se define como una colección de elementos que permiten agilizar el acceso a los registros que forman parte de una base de datos.

Los índices se usan principalmente por dos razones:

- Para aumentar la velocidad en las consultas.

- Para asegurar la unicidad de los datos.

Básicamente el funcionamiento de un índice consiste en almacenar en una tabla parejas de elementos: por un lado el elemento que se desea indexar y por otro su posición en la base de datos. Cuando se trata de localizar un elemento previamente indexado, únicamente hay que buscar dicho elemento en el índice, y una vez localizado, acceder al registro indicado por la posición definida por el índice.

Los índices permiten acelerar la realización de consultas sobre los datos eliminando la necesidad de recorrer de forma secuencial todos los datos, facilitando el acceso a los campos sobre los que se realizan búsquedas frecuentes.

El planificador de consultas es el encargado de elaborar un plan de consulta en el que establezca qué índice se va a utilizar para resolver la consulta y en qué orden se realizarán las uniones (*joins*) de las tablas. En definitiva, el planificador a través de fórmulas matemáticas avanzadas estima la mejor vía para resolver la consulta, es decir, la forma de localizar datos y de resolver las relaciones entre tablas.

Los índices son estructuras de la base de datos que quedan almacenadas de forma independiente de las tablas que contienen los datos pero que a su vez dependen de ellas.

Los índices espaciales son índices diseñados para optimizar las búsquedas basadas en criterios espaciales sobre la información geográfica. Existen diferentes tipos de índices espaciales como se verá más adelante.

Las consultas espaciales más utilizadas son:

- **De apuntador**: Se selecciona un objeto cuya geometría contiene el punto p.

- **De ventana**: Se seleccionan los objetos tales que su geometría corta o está contenida en una ventana.

- **De unión (join) espacial:** Dadas dos colecciones de objetos, se seleccionan las parejas tales que su geometría se interceptan.

2.4.1 Principio de Indexación Espacial

La construcción de un índice espacial no está basada directamente en los objetos, sino en el mínimo rectángulo que engloba a cada uno de ellos (véase la figura 3). Al rectángulo mínimo que engloba a un objeto se le denomina *REM* (*Rectángulo Espacial Mínimo*), en inglés, *MBR* (*Minimum Bounding Rectangle*). Esto simplifica las operaciones sobre un índice espacial.

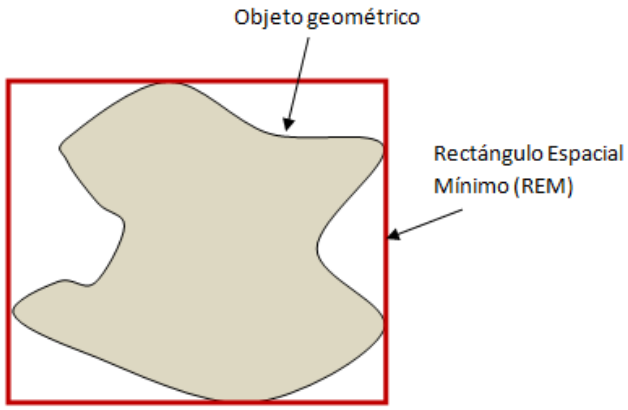

Figura 3: Ejemplo de rectángulo espacial mínimo (REM o MBR)

Para cubrir todo el espacio ocupado por una *capa* o *feature,* se parte de un nodo raíz, que es el nodo superior, es decir, el nodo que no tiene padre, y cuyo *MBR* contiene lógicamente todos los objetos del espacio de búsqueda.

Para encontrar los datos de un objeto dentro de la capa, habría que hacerlo de forma secuencial, con lo que no ahorraríamos tiempo ninguno. Para acelerar la búsqueda, se subdivide el conjunto de objetos en subconjuntos más pequeños, cada uno de los cuales están contenido en un *MBR*. Cada objeto estará en un solo subconjunto, y los subconjuntos se crean de forma que sus *MBRs* se solapen lo menos posible. Si los subconjuntos tienen un número de objetos mayor que una cantidad prefijada dentro del algoritmo, se subdividen. Para que los *MBRs* hijos de un *MBR* determinado se solapen lo menos posible, se insertan en el árbol objetos de uno en uno, de forma que al subdividirse en subgrupos éstos contengan cada uno de los objetos más cercanos.

Se tiene entonces que las entradas (*MBR*, *uid*[4]) son los nodos hoja del árbol ordenado de manera que se permite seleccionar en unos pocos pasos los *MBRs* que contienen los objetos involucrados en la consulta. Por tanto, toda operación que utiliza el índice es ejecutada en dos partes:

- Se recorre el índice para seleccionar los *MBRs* que corresponden a la consulta (apuntador, ventana, join, etc.) y extrae los identificadores de los registros que contienen datos .

- Se aplican las operaciones sobre los objetos (intersección y adyacencia de polígonos).

4 Indentificador

Los índices espaciales son estructuras de indexación que permiten recuperar objetos espaciales de manera eficiente. Una gran mayoría de estos índices han sido diseñados para hacer uso de la memoria secundaria. En la actualidad existen algoritmos que permiten trabajar únicamente en la memoria principal sin necesidad de acceder al disco.

En el campo del diseño de estructuras de indexación se pueden distinguir dos métodos de acceso [6], es decir, dos estructuras de datos que determinan cómo se realiza el acceso a los datos almacenados en páginas de disco de almacenamiento secundario:

- *MAPM*: Método de acceso de puntos multidimensionales, en inglés *PAM (Point Access Methods*): Trabajan únicamente con colecciones de puntos.

- *MAE*: Método de acceso espacial, en inglés *SAM (Spatial Access Methods*): Trabajan con colecciones de puntos, líneas, polígonos, ...

2.4.2 Tipos

Para poder maximizar su eficiencia, los índices espaciales usan una estructura de datos en forma de árbol como vimos en el aparatado anterior. Según el tipo de árbol usado, se clasifican los distintos índices. A continuación, se muestran los más usados, describiendo primero el *Árbol B*, que aunque es utilizado en índices de datos alfanuméricos, es la base de los demás.

Árbol B (B-tree)

Son estructuras de datos de árbol utilizadas frecuentemente en las implementaciones de bases de datos y sistemas de archivos.

Un Á*rbol-B* debe tener un número variable de nodos hijo dentro de un rango predefinido (véase la figura 4). Cuando se elimina o se inserta un dato de la estructura, la cantidad de nodos hijo varía dentro de un nodo. Para que siga manteniéndose el número de nodos dentro del rango predefinido, los nodos internos se unen o se dividen.

Un Á*rbol-B* se dice que está balanceado porque necesita que todos los nodos hoja se encuentren al mismo nivel.

Se utilizan para datos que pueden ser ordenados a lo largo de un eje como por ejemplo

números, letras o fechas y para datos alfanuméricos no geográficos.

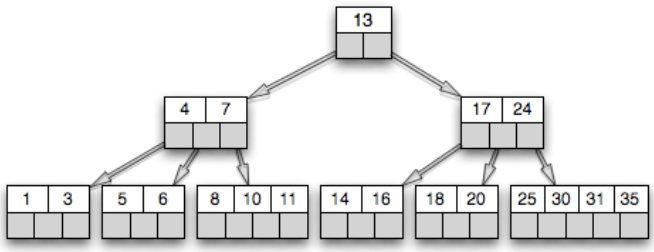

Figura 4: Representación Árbol-B

QuadTree

Un quadtree es un árbol cuaternario, es decir, una estructura de datos en forma de árbol en la que cada nodo del árbol tiene cuatro nodos como máximo [7]. Este tipo de árboles se utiliza para dividir espacios de dos dimensiones en piezas recursivamente pequeñas y, de esta forma, dividir el espacio en cuadrantes (véase la figura 5).

Este tipo de estructuras sirven para dividir geometrías puntuales, no son eficientes para indexar geometrías de tipo área o línea, que pueden estar diseminadas en varios cuadrantes a la vez.

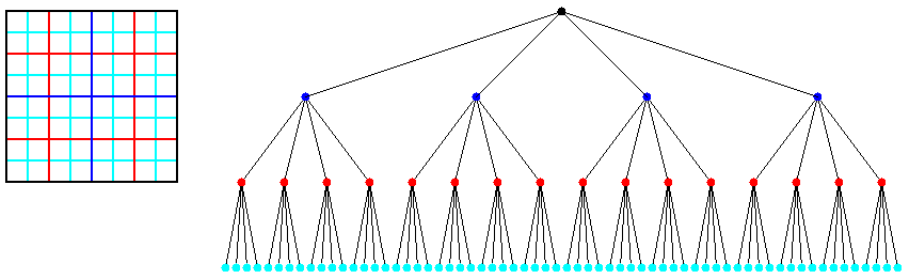

Figura 5: Representación Árbol Quadtree

Árbol-R (R-tree)

Un Árbol-R es una estructura de datos de tipo árbol que se usa para datos espaciales, es decir, permite indexar datos de contenido geométrico y multidimensional como por ejemplo las coordenadas *x*,*y* de un lugar geográfico [8] .

Es un método de indexación para objetos de tipo punto, línea y polígono (véase la

figura 6), es decir, tal y cómo se ha descrito anteriormente de tipo *MAE*. Esta técnica es la más utilizada en las Bases de Datos Espaciales y está considerada como un ejemplo paradigmático.

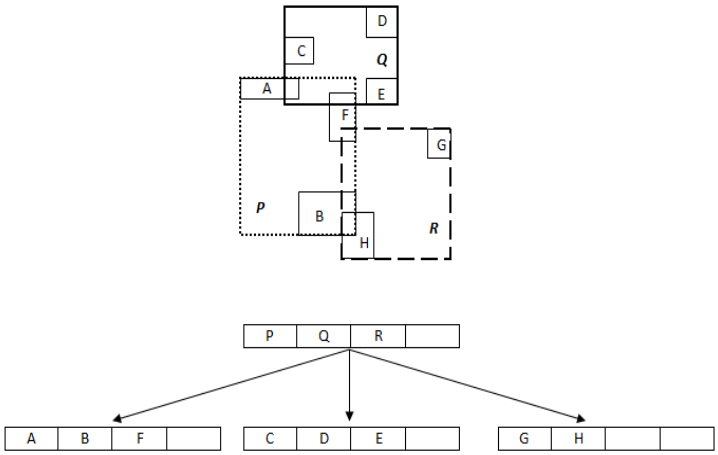

Figura 6: Represenación Árbol-R

K-d-tree

Es el método más extendido dentro de los que trabajan únicamente con colecciones de puntos (*MAPM*) debido principalmente a su eficiencia y sencillez. K-d-tree es un árbol binario que permite almacenar un conjunto finito de puntos en un espacio de k dimensiones (número de atributos) [9], es decir, en un ejemplo con dos dimensiones (*x* e *y*) cada nivel de nodos particiona el espacio que representa alternativamente por la dimensión *x* y por la dimensión *y*, de forma que se repartan equitativamente los puntos en el subárbol izquierdo y derecho (véase la figura 7).

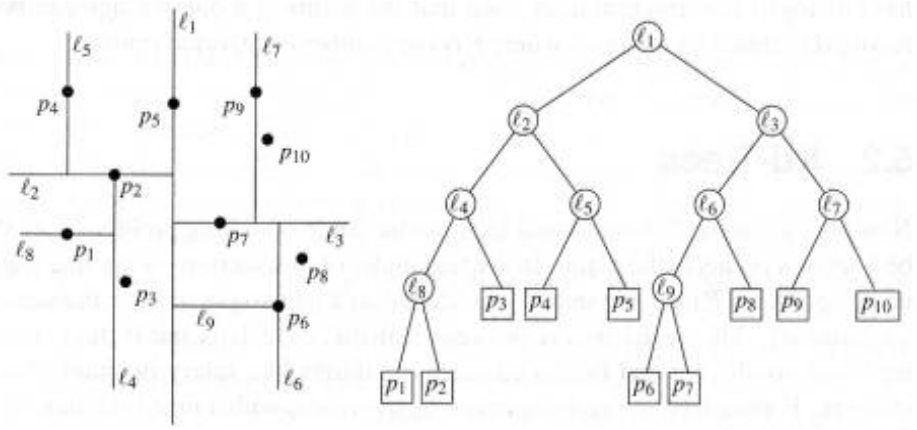

Figura 7: Representación Árbol K-d-Tree

Índices GIST (Generalized Search Tree)

Este tipo de índice agrupa los objetos considerando los que se superponen, los que están a un lado y los que están dentro.

GIST proporciona un Árbol de búsqueda y una forma generalizada de indexación sobre datos geográficos, además aumenta la velocidad de las búsquedas con toda clase de estructuras irregulares (datos espectrales, arrays de enteros, etc) las cuales no se pueden tratar con la indexación basada en B-Tree.

Además, GIST puede indexar columnas que contienen valores nulos

2.4.3 Ejemplos de implementación de índices espaciales en bases de datos de mercado.

Resulta interesante comprobar cómo algunas bases de datos de reconocido prestigio resuelven la indexación de datos espaciales. A continuación, se describen como implementan las bases de datos *Oracle Spatial*[5] y *PostGIS*[6] las diferentes técnicas de indexación de información geográfica.

Oracle Spatial

Oracle Spatial es un componente opcional del gestor de bases de datos relacionales

5 http://www.oracle.com/technetwork/database/options/spatial/index.html
6 http://postgis.refractions.net/

Oracle, que requiere una licencia disponible únicamente para la base de datos *Oracle Enterprise Edition*. *Oracle Spatial* es un conjunto integrado de funciones y procedimientos que habilitan datos espaciales para ser almacenados, que estén accesibles y analizados rápidamente y eficientemente en una base de datos Oracle.

En cambio, *Oracle Locator* se incluye gratuitamente en *Oracle Standar e* incorpora los objetos de almacenamiento *SDO_GEOMETRY* y también las opciones de creación de índices espaciales.

Oracle Spatial consiste en los siguientes componentes:

- Un esquema (*MDSYS*) que describe el almacenamiento, sintaxis y semántica de tipos de datos geométricos soportados (*SDO_GEOMETRY*), es decir, incorpora las tablas de metadatos que requiere el estándar *Simple Features for SQL*. Estas tablas almacenan los metadatos espaciales de los objetos geográficos almacenados en las tablas de oracle.

- Un mecanismo espacial de indexación.

- Un conjunto de operadores y funciones para el funcionamiento de consultas, consultas de unión espacial y otras operaciones de análisis espacial.

- Utilidades administrativas.

Una geometría es una secuencia ordenada de vértices que son conectadas por segmentos de una línea recta o arcos circulares. En el caso del tipo *SDO_POINT(punto)* las coordenadas no se explicitan dentro de un array de coordenadas (*sdo_ordinate_array*) sino que el objeto *sdo_geometry* puede almacenar directamente un tipo *sdo_point*. Las semánticas de la geometría son determinadas por estos tipos.

Oracle Spatial soporta varios tipos primitivos y geometrías compuestas de colecciones de esos tipos, incluyendo dos dimensiones:

- Puntos y grupos de puntos

- Cadenas de líneas
- Polígonos de n-puntos
- Cadenas de líneas de arcos
- Polígonos de arcos
- Polígonos compuestos
- Cadenas de líneas compuestas
- Círculos
- Rectángulos optimizados

Oracle Spatial usa indexación *Árbol-R* (por defecto) o indexación *Quadtree*, o ambos.

PostGis

PostGIS es una extensión para la base de datos *PostgreSQL*. Se trata de un proyecto de software libre que es desarrollado y mantenido por la compañía Refractions Research.

PostGIS ha sido certificado por la *OGC* ya que cumple la especificación *Simple Features for SQL 1.1*, además permite usar todos los objetos que aparecen en la especificación OpenGIS como puntos, líneas, polígonos, multilíneas, multipuntos y colecciones geométricas.

PostgreGIS soporta 2 tipos de índices espaciales por defecto: *R-Tree* y *GIST*.

- *R-Tree* divide los datos en rectángulos, subrectángulos, subsubrectángulos, etc. La implementación de PostgreSQL de los R-Tree no es tan robusta como la implementación GIST.
- *GIST* (*Generalized search trees*) Estos índices dividen los datos en "cosas que están a un lado", "cosas que se superponen" y "cosas que están dentro". Además soporta un amplio rango de tipos de datos, incluidos los datos GIS. PostGIS usa índices Rtree sobre GIST para indexar los datos GIS.

Características	**Oracle Spatial**	**PostGis 8.2**
Funciones espaciales	400 funciones espaciales y operadores como centroides y agregados	300 funciones y operadores
Algoritmos de análisis espacial.	R-Tree y Quadtree	B-Tree, R-Tree, GIST
Soporte de Geometría	Puntos, líneas, polígonos con o sin agujeros, Strings de arcos, polígonos compuestos, círculos y rectángulo, multipunto, multilinea, multipolígono, superficie, sólido, etc...	Puntos, líneas y polígonos. Multipuntos, multilíneas poligonales, multipolígonos, conjuntos de geometrías

Tabla 1: Tabla comparativa de las principales características espaciales de Oracle Spatial y PostGis 8.2

2.4.4 Descripción del tipo de índice seleccionado para la implementación: Algorítmo Árbol-R (R-Tree)

Aunque se conoce que los *árboles R* no es el algoritmo de indexación más eficiente [10] para datos espaciales (ya que su rendimiento se deteriora a medida que aumenta la cantidad de datos y no previene el solapamiento), sigue siendo el más representativo de todos y está considerado como el más paradigmático. Se puede comprobar cómo este tipo de índice multidimensional se encuentra disponible en la mayoría de las bases de datos con soporte de indexación de datos espaciales.

Vamos a profundizar a continuación en algunos conceptos relativos al *Árbol-R* que será necesario conocer para el desarrollo del índice. Para una capa con objetos geométricos, un índice *Árbol-R* consiste en un índice jerárquico sobre *REMs* de las geometrías de la capa [2].

Un *Árbol-R* posee las siguientes propiedades [12]:

- Cada nodo del *Árbol-R* está asociado con una zona rectangular de un espacio. Por tanto la **raíz** está asociada con todo el espacio. El rectángulo asociado a un **nodo interno** es el *MBR* de los rectángulos asociados a cada uno de sus hijos. Y el rectángulo asociado a cada entrada de un **nodo hoja** contiene un objeto (Polígono).

- Cada nodo del árbol, exceptuando la raíz contiene un número de entradas n tal que $m<n<M$. El límite inferior (o grado) *m* evita la degradación del árbol y asegura un almacenamiento eficaz de los objetos. El límite superior *M* se escoge tal que sea igual al tamaño de una página disco.

- Si el nodo raíz no es una hoja, tiene al menos dos entradas.

- El *Árbol-R* es una estructura equilibrada, es decir todos los nodos hojas están al mismo nivel.

- Un nodo interno referencia los nodos del nivel siguiente en el árbol. Una entrada es una pareja (*MBR*, *apuntador*) donde *MBR* es el rectángulo del nodo hijo, y apuntador su dirección.

- Un nodo hoja referencia el conjunto de objetos espaciales contenidos en su rectángulo. Una hoja contiene entradas (*MBR*, *id-tupla*) donde *MBR* es el rectángulo que engloba el objeto espacial e id-tupla es la dirección física de la tupla en el disco. La tupla es el registro que contiene los datos de dicho objeto.

El *Árbol-R* puede ser utilizado para almacenar datos espaciales de la siguiente manera:

- Las entradas de un nodo no hoja son parejas (*MBR*, *apuntador*) donde *MBR* es el rectángulo que engloba el nodo hijo y apuntador es la dirección donde se almacena el nodo. Un *MBR* está definido por los valores máximos y mínimos de las dimensiones horizontal y vertical respectivamente.

- Un nodo hoja contiene parejas (*MBR*, *id-tupla*) donde *MBR* es el rectángulo del registro e *id-tupla* su dirección.

Como el *Árbol-R* es equilibrado, el costo para recuperar el objeto es igual a la altura del árbol. El inconveniente es que a veces es necesario recorrer varios caminos desde la raíz del árbol a las hojas debido a los solapamientos de los rectángulos asociados a los objetos. Por tanto, el rendimiento de la operación de búsqueda se degrada.

2.4.5 Operaciones básicas con Árboles R

Son muchas las tareas que se pueden realizar sobre una estructura de árbol. A continuación, se describen las operaciones de más interés para implementar el algoritmo de indexación.

2.4.5.1 Búsqueda en Árboles R

Cada nodo interno contiene una serie de tuplas cada una con un rectángulo (*MBR*) y un puntero al nodo hijo; en cambio cada nodo de la hoja contiene una serie de tuplas cada una con un rectángulo (*MBR*) y otro puntero al objeto espacial.

Si es la raíz del árbol, se buscan todos los registros indexados cuyos rectángulos se solapan parcial o totalmente con el *MBR* del objeto buscado. Se va pasando por los hijos del o de los nodos que solapan.

Si estamos en un nodo interno entonces se aplica la búsqueda en cada nodo hijo cuya raíz esté apuntada por el puntero hijo y se comprueba su solape con el *MBR* del objeto. Se va a los hijos del o de los nodos que solapan, y se repite la operación recursivamente hasta que se encuentra un nodo hoja. Si en un nodo dado no hay solape con ninguno de los *MBRs* de un nodo hijo, el objeto no se encuentra en el árbol.

Cuando se alcanza un nodo hoja, se devuelven todas las entradas (a cada uno de los rectángulos mínimos que contiene cada objeto espacial) que se solapan con *S* como conjunto resultado. Si no hay solape con ninguno de los *MBRs* de los objetos del nodo hoja, el objeto no se encuentra en el árbol.

2.4.5.2 Inserción en Árboles R

Para insertar un objeto, se recorre recursivamente el árbol comenzando desde el nodo raíz. Se examinan todos los rectángulos del nodo interno actual. A continuación, se selecciona el rectángulo que menos ampliación necesita para incluir el nuevo objeto. En el caso donde hay más de un rectángulo que cumple este criterio, se elige el que tiene el área más pequeña. El algoritmo continúa por el nodo interno seleccionado y una vez alcanzado el nodo hoja, se hace una inserción directa siempre que este nodo hoja no esté lleno.

Si el nodo hoja estuviera lleno, debe dividirse antes de realizar la inserción, es decir, se

divide el nodo en dos conteniendo los viejos elementos más el nuevo objeto. Si el nodo padre no permitiera insertar el nuevo hijo creado con la división, se divide el padre del padre, y así sucesivamente, si fuera necesario. Si la división alcanza la raíz y debe dividirse, se crea una nueva raíz cuyos hijos son los dos nodos que se crearon con la división de la raíz.

2.4.5.3 Borrado en Árboles R

Consiste en localizar la hoja que contiene un determinado elemento cuando se encuentre, sólo se necesita borrarlo de la hoja.

Si hubiera dos nodos que tienen solamente un hijo luego del ajuste, se juntarán estos dos nodos, y el nodo padre quedará con un hijo menos Esto puede, si es necesario, continuar recursivamente subiendo niveles hacia la raíz. Si se alcanza la raíz, entonces el hijo único será la raíz y por último la altura del árbol se decrementa.

Si la raíz tiene solamente un hijo luego del ajuste, entonces este hijo será la raíz y por último la altura del árbol se decrementa.

3. Características y funcionamiento de H2 y de la extensión JASPA

Una vez descritos los conceptos básicos relativos a la indexación de información espacial, se pasará a describir dos de los componentes externos con los que se integrará la aplicación. En primer lugar, se profundizará en el estudio de H2 para identificar los recursos que aporta para alojar el índice espacial que se pretende implementar. En segundo lugar, se estudiará la extensión JASPA para descubrir las posibilidades de integración con H2.

3.1 H2

H2 es una base de datos relacional programada completamente en Java [2]. Esto implica que se puede integrar con otras aplicaciones Java y acceder a ella ejecutando SQL directamente, sin tener que pasar por una conexión a través de sockets, como ocurre con otras bases de datos.

H2 también permite crear tablas en memoria y en disco. Además todas las operaciones de manipulación de datos son transaccionales.

3.1.1 Características más destacables

A continuación se indican las características más relevantes de *H2*:

- Permite la definición de tipos de datos por el usuario.
- Dispone de optimizador de consultas.
- Permite la encriptación de datos.
- Admite bloqueo a nivel de tabla y a nivel de fila.
- Preparada para multiconcurrencia.
- Admite búsquedas avanzadas de textos.
- Permite utilizar conexiones JDBC externas lo que supone una mejor integración y un considerable incremento en el rendimiento.
- Permite crear tablas en memoria y en disco.

- Todas las operaciones de manipulación de datos son transaccionales.
- Compatibilidad con conexiones a través de ODBC.
- Permite la personalización de funciones agregadas y admite secuencias.
- Permite columnas calculadas.
- Dispone de seguridad basada en roles.

3.1.2 Tipos de datos

Como todos los Sistemas de Gestión de de Bases de Datos, *H2* implementa una serie de tipos de datos. A continuación, se escriben los tipos de datos agrupados por las siguientes categorías: lógicos, numéricos, cadena de caracteres, fecha y array.

Lógicos

- BOOLEAN: almacena datos que presenten dos estados como son TRUE o FALSE.

Numéricos

- Enteros
  - INT: almacena valores comprendidos entre -2147483648 y 2147483647
  - TINYINT: almacena valores comprendidos entre -128 y 127
  - SMALLINT: almacena valores comprendidos entre -32768 y 32767
  - BIGINT: almacena valores comprendidos entre -9223372036854775808 y 9223372036854775807
  - IDENTITY: almacena valores autoincrementables comprendidos entre -9223372036854775808 y 9223372036854775807
- Coma flotante
  - REAL: almacena números de precisión simple.

- DOUBLE: almacena números de doble precisión.

- Exacto
  - DECIMAL: almacena números reales.

Cadena de caracteres

- BINARY: Representa un array de bytes de hasta 2 Gb.
- BLOB: Almacena datos binarios como ficheros o imágenes.
- CHAR: Almacena una cadena de longitud fija que tiene como longitud máxima *Integer.MAX_VALUE*, es decir, 2^{31} -1.
- VARCHAR: Almacena una cadena de longitud variable que tiene como longitud máxima *Integer.MAX_VALUE*, es decir, 2^{31} -1.
- CLOB: Almacena documentos o textos de tamaño arbitrario como ficheros HTML o XML.

Fecha

- TIME: Almacena una Hora en formato horas, minutos, segundos.
- DATE: Almacena una fecha sin incluir la hora.
- TIMESTAMP: Almacena la fecha y hora con zonificación.

Array

- ARRAY: Permite almacenar datos de longitud *multidimensional*. Usa una lista de valores separados por coma. Este tipo de datos es mapeado a un array de objetos de java (*javalangObject[]*)

Tipos de datos definidos por el usuario

H2 permite la creación de tipos de datos definidos por el usuario lo que combinado al

tipo de dato *ARRAY* permite la creación y manejo de forma ágil de tipos de datos geométricos.

3.1.3 Índices

No soporta ningún tipo de índice espacial aunque si dispone de índices del tipo *B-Tree*.

3.1.4 Desarrollo con H2

H2 soporta la creación de nuevas clases y métodos. También permite la definición por el usuario de funciones Java que a su vez pueden usarse como procedimientos almacenados. Además, una función puede estar declarada (registrada) antes de ser usada pero puede definirse usando código fuente, o como una referencia a una clase compilada siempre que se encuentre disponible en el *CLASSPATH*.

H2 también permite la creación de nuevos tipos de datos.

3.2 Extensión JASPA

JASPA (*JAva SPAtial*) [9] es una extensión con licencia *GNU GPL* que añade la funcionalidad espacial a bases de datos relacionales que admitan procedimientos almacenados en Java. Actualmente *PostgreSQL* y *H2* están soportados incorporando más de 200 funciones espaciales vectoriales. *JASPA* tiene una funcionalidad espacial similar a *PostGIS* 1.4. ya que soporta predicados y operadores espaciales, arreglos de geometrías, agregados espaciales, etc. Además, es bastante fácil de extender ampliando funcionalidades utilizando procedimientos almacenados en Java.

3.2.1 Características más destacables

- Completamente escrita en Java.
- En la actualidad puede usarse junto a *PostgreSQL* y *H2*.
- Posibilidad de migración a otras Bases de datos Java.
- Dispone de más de 200 funciones espaciales.
- Compatible con *PostGIS*.

- Fácilmente extensible usando procedimientos almacenados en Java.
- Uso de librerías *JTS*[7] (*Java Topology Suite*) y *GeoTools*[8]
    - *JTS* (*Java Topology Suite*). Biblioteca ampliamente utilizada en *JASPA* y muchos otros proyectos de código abierto, ofreciendo la posibilidad de realizar análisis espaciales.
    - *GeoTools*. Biblioteca utilizada para las proyecciones y el soporte de *KML* y los importadores shape.
- *JASPA* para *PostgreSQL* provee indexación espacial basada en *PostgreSQL Gist*
- Sistema de referencia espacial.
- Soporta los estándares *SHP*, *KML* y *GML* como formatos de entrada.
- Posee funciones topológicas propias.

3.2.2 Objetos SIG

Los objetos SIG soportados por *JASPA* son los definidos en la especificación *OpenGIS* como puntos, líneas, polígonos, multilíneas, multipuntos, multipolígonos y colecciones geométricas.

JASPA soporta diferentes tipos de datos espaciales: define las dos formas de representar objetos espaciales que establece *OpenGIS* (*WKT* y *WKB,* ya descritos anteriormente) donde se guarda información del tipo de objeto y sus coordenadas. Otros formatos también disponibles en *JASPA* son *EWKB (Extended Well-known Binary)*, *EWKT (Extended Well-known Text Representation)*, *GML (Geography Markup Language) y KML (Keyhole Markup Language)*.

Además define las tablas de metadatos que establece *OpenGIS* como son: *SPATIAL_REF_SYS* y *GEOMETRY_COLUMNS*, también descritas anteriormente.

JASPA ofrece sus propios operadores y predicados espaciales para resolver las consultas

7 JTS:http://www.vividsolutions.com/jts/jtshome.htm
8 Geotools:http://www.geotools.org/

de unión espacial, también dispone de una gran variedad de funciones geométricas para realizar operaciones de análisis espacial relacionadas con referencia lineal, proyecciones, sistemas de referencia espacial y también, con la construcción y edición de geometrías.

4. Descripción de la solución

Esta parte de la memoria se describe la solución propuesta a alto nivel.

El objetivo que se plantea en este trabajo consiste en ofrecer una solución para la creación de un índice espacial para la extensión *JASPA* sobre la base de datos *H2*. Esta solución está limitada a operaciones espaciales en dos dimensiones.

Se ha tratado de que la solución que se propone sea lo suficiente flexible como para que no se necesite modificar ni el código fuente de *JASPA*, ni de *H2*.

Para la implementación del índice espacial, se necesitará crear una tabla auxiliar por cada uno de los índices que se cree. Esta tabla estará relacionada a través de un identificador con la tabla que contiene los objetos geométricos.

Esta tabla auxiliar tendrá la siguiente estructura:

- Un identificador que se definirá como clave primaria y que permitirá relacionarla con la tabla que contiene los datos geométricos.
- Cuatro campos que alojarán la definición del rectángulo mínimo (*MBR*) que contiene el objeto espacial. Este rectángulo vendrá definido a partir de las coordenadas *x* e *y* de los puntos que identifican el vértice inferior izquierdo y el vértice superior derecho del rectángulo.

Además, esta nueva tabla auxiliar se creará automáticamente al crear el índice espacial, y deberá estar sincronizada de forma automática con la tabla que contiene los objetos espaciales por medio de los disparadores (*triggers*) que la base de datos *H2* permite construir. De esta forma se persigue que el índice siempre esté actualizado, es decir, que refleje todas las variaciones que la tabla de datos espaciales pueda sufrir (modificaciones, inserciones y borrados de registros).

El *Árbol-R* se generará en memoria a partir de la tabla auxiliar descrita anteriormente cada vez que el usuario introduzca desde consola de H2 la orden adecuada para activar el índice.

Por último, mencionar que se implementará una función *Java* en *H2* que incluirá los métodos externos que podrán ser invocados por el usuario desde la consola *Sql* de *H2* para el manejo de los índices espaciales. Para facilitar al usuario el uso de éstos métodos se crearán en *H2* unos seudónimos o *alias,* con un formato similar a las funciones espaciales definidas por la extensión *JASPA*.

5. Análisis, diseño e implementación

Ahora que se han descrito tanto los conceptos necesarios para comprender la implementación como cada uno de los componentes necesarios sobre los que trabajará la aplicación, a continuación se describirán las etapas del ciclo de vida de su desarrollo.

5.1 Análisis

A lo largo de etapa de esta etapa se comenzará explicando las principales decisiones tomadas en esta fase de análisis.

A continuación, y a través de diferentes diagramas casos de uso, de componentes y de modelo de datos se irá describiendo la solución propuesta.

5.1.1 Requisitos no funcionales

A continuación se indican las decisiones tomadas más importantes durante esta fase de análisis:

- Se opta por seguir una arquitectura de tres capas por componentes con la idea de organizar conceptualmente los elementos de diseño en grupos independientes.

- Se decide crear un conjunto de componentes que integren las funcionalidades requeridas con *H2* y *JASPA*.

- Se rechazan las soluciones que necesiten modificar el código fuente de *H2* y/o *JASPA*.

- Se prevé que las nuevas funcionalidades a implementar para el manejo de los índices espaciales sigan el mismo formato que las proporcionadas por la extensión *JASPA*.

5.1.2 Diagrama de casos de uso

En la imagen siguiente se muestra el diagrama de casos de uso de la aplicación.

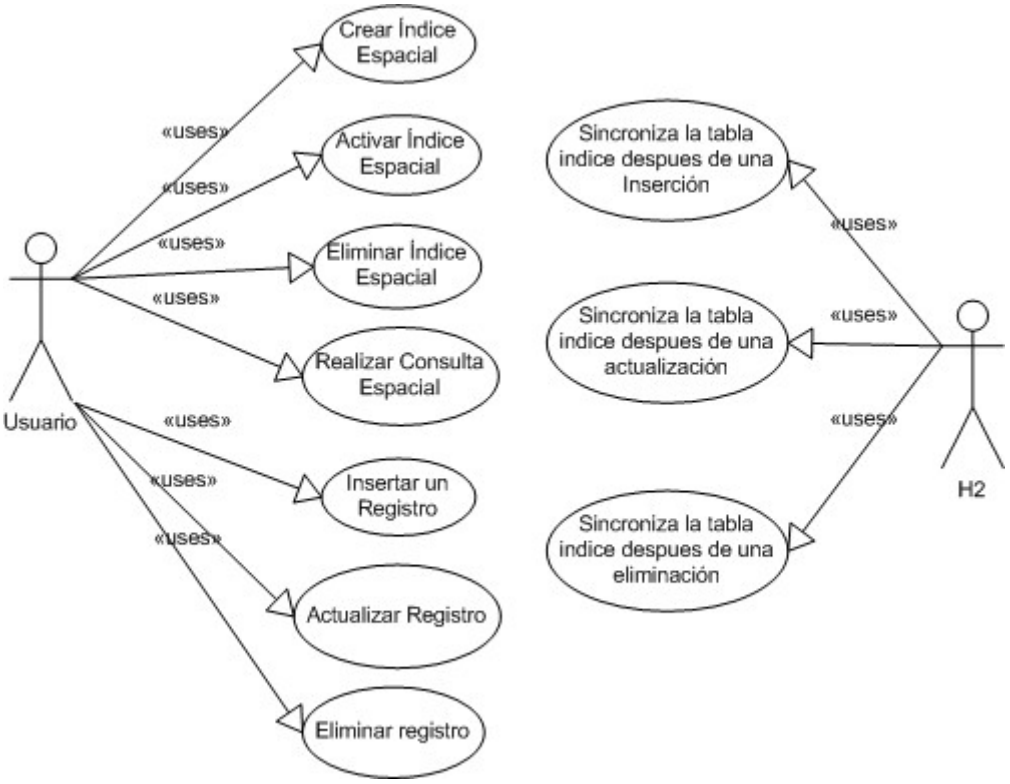

Figura 8: Diagrama de casos de uso

Se puede comprobar cómo en este diagrama existen dos actores independientes, que no interactúan directamente entre sí ni tienen en común ningún caso de uso. Esto indica que hay dos grupos de funcionalidades independientes. En el caso del actor H2, son acciones internas de la base de datos para mantener la sincronización, totalmente automáticas, pero activadas por el usuario al insertar, modificar o actualizar un registro en la tabla que contiene los datos geométricos.

5.1.3 Diagrama de componentes

En la imagen siguiente se muestra un diagrama de componentes y clases de la aplicación. Se ve cómo en la capa de *Lógica de negocio* se implementan dos componentes con la funcionalidad separada que corresponde a cada actor, y cómo se relacionan estos componentes con los externos de la capa de *Integración*. También se observa cómo la capa de *Presentación* sólo se relaciona con uno de los componentes de la capa de *Lógica de negocio*, la que da servicio al actor *Usuario*, de acuerdo con el diagrama de *Casos de uso*.

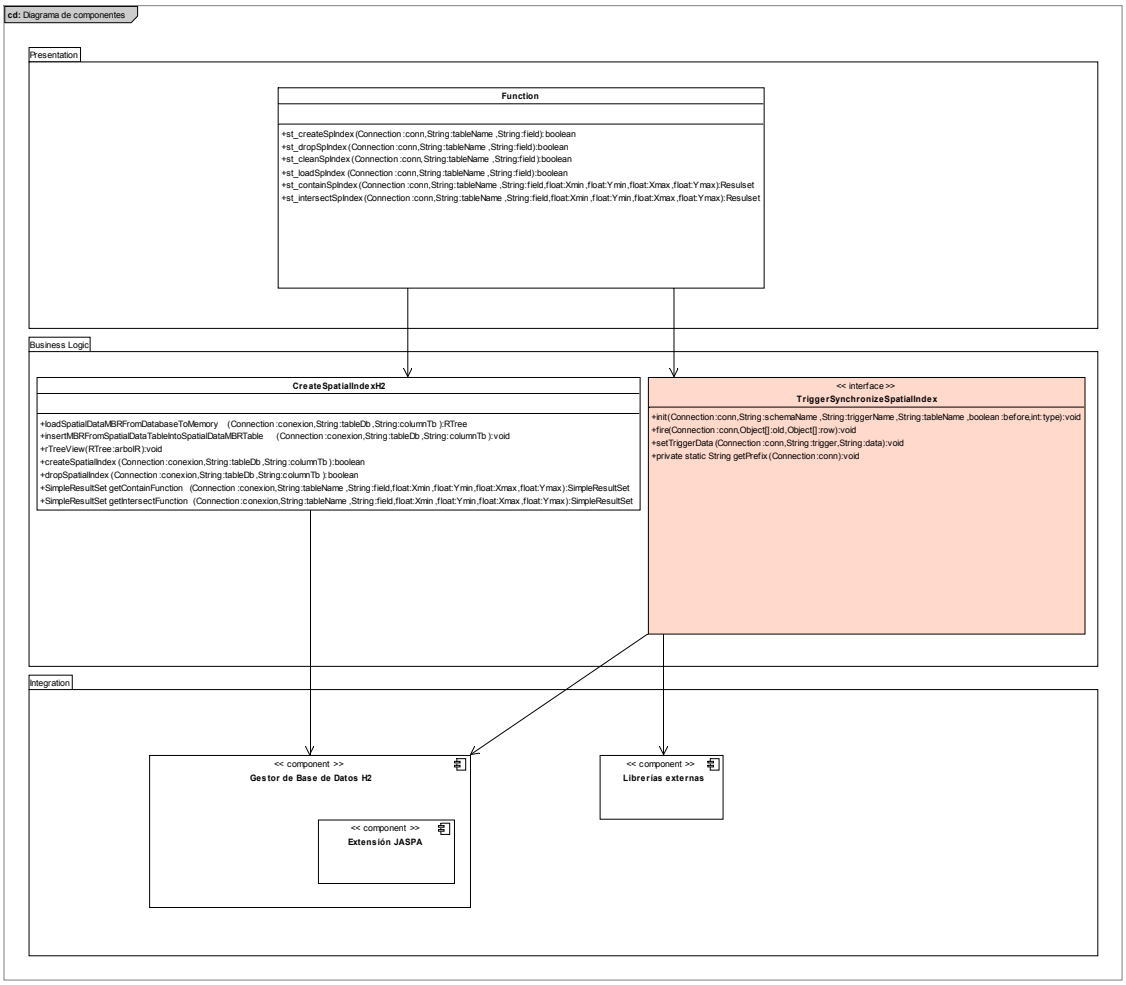

Figura 9: Diagrama de componentes

Nivel de presentación.

Este nivel es el encargado de generar la interfaz de usuario ya que contiene los componentes necesarios para habilitar la interacción del usuario con la aplicación, es decir, los alias de las funciones creadas para el manejo del índice espacial que se pueden invocar desde la consola de *H2*.

Estas funciones son:

- ST_CREATE_SPIDX: permite crear la tabla con los *MBRs* a partir de la tabla que contiene los datos espaciales.

- ST_LOAD_SPIDX: permite cargar el contenido de la tabla que contiene los

MBRs en memoria (en una estructura de datos de Árbol-R).

- ST_CONTAIN_SPIDX: permite realizar una consulta espacial para calcular los objetos completamente contenidos dentro de una ventana o rectángulo dado.

- ST_INTERSECT_SPIDX: permite realizar una consulta espacial para calcular los objetos parcial o completamente contenidos dentro de una ventana o rectángulo dado.

- ST_CLEAN_SPIDX: permite vaciar la estructura de datos de memoria que contiene el Árbol-R.

- ST_DROP_SPIDX: permite eliminar un índice espacial.

Nivel de negocio.

Contiene la lógica que modela los procesos de negocio y es donde se realiza todo el procesamiento necesario para atender las peticiones del usuario.

Existen dos clases diferentes una destinada para la programación de los *triggers* (relacionada con los casos de uso de actor *H2*) y la otra que contiene, entre otros, los métodos que implementan la lógica de cada uno de las funciones disponibles para la interacción del usuario (descritos anteriormente en la capa de presentación).

Nivel de administración de datos.

Es el encargado de la persistencia de la información, así como de suministrar y almacenar la información para el nivel de negocio. En esta capa se encuentran los componentes de *H2* y la extensión *JASPA* necesarios para la interacción con la aplicación por medio de una conexión *JDBC*.

Dada la complejidad prevista de la programación del índice, finalmente también se decide usar un componente externo, una implementación del *Árbol-R* (que a su vez usa componentes de varias librerías externas), haciendo las adaptaciones precisas.

5.1.4 Modelo de datos

Es interesante comenzar indicando que la extensión *JASPA* cuenta con un modelo de datos que da soporte al manejo de datos espaciales desde *H2*. Para la implementación de esta aplicación ha sido necesario ampliar este modelo para que incluya la nueva funcionalidad de indexación de datos espaciales. Para ello se ha tenido que crear una nueva tabla de datos para almacenar los *MBRs* de la tabla que contiene los datos geométricos.

A continuación (véase la tabla 2), se describe la estructura de esta nueva tabla de datos que contendrá el índice, finalmente y a través de una imagen se verá cómo está relacionada esta nueva tabla con la estructura de tablas de *JASPA*.

Tabla Índice espacial		
Definición: Es la estructura de datos que almacenará los ***MBRs***[9] de cada uno de los registros de la tabla que contiene los datos geométricos que se desea indexar. **Organización:** Indexada por el campo *SGID* y está relacionada con la tabla que contiene los datos geométricos a indexar por este mismo campo.		
Datos elementales que la componen:		
	Descripción	**Tipo**
GID	Corresponde al identificador de cada registro. Este campo será el campo índice de esta tabla para relacionarlo con la tabla base que contiene los datos geométricos.	*Integer*
XMIN	Corresponde a la coordenada *x* mínima del rectángulo del *MBR* que engloba a los datos espaciales incluidos en el campo geométrico identificado por el campo *SGID*.	*Float*
YMIN	Corresponde a la coordenada *y* mínima del rectángulo del MBR que engloba a los datos espaciales incluidos en el campo geométrico identificado por el campo *SGID*.	*Float*
XMAX	Corresponde a la coordenada *x* máxima del rectángulo del MBR que engloba a los datos espaciales incluidos en el campo geométrico identificado por el campo *SGID*.	*Float*
YMAX	Corresponde a la coordenada *y* máxima del rectángulo del MBR que engloba a los datos espaciales incluidos en el campo geométrico identificado por el campo *SGID*.	*Float*

Tabla 2: Descripción de la estructura de datos que almacena el índice

9 Al rectángulo mínimo que engloba a un objeto se le denomina *REM* (Rectángulo Espacial Mínimo), en inglés, *MBR* (*Minimum bounding rectangle*)

En la imagen (véase la figura 10) siguiente se muestra cómo se ha incluido y relacionado la tabla índice dentro de la estructura de tablas existentes en la extensión *JASPA*.

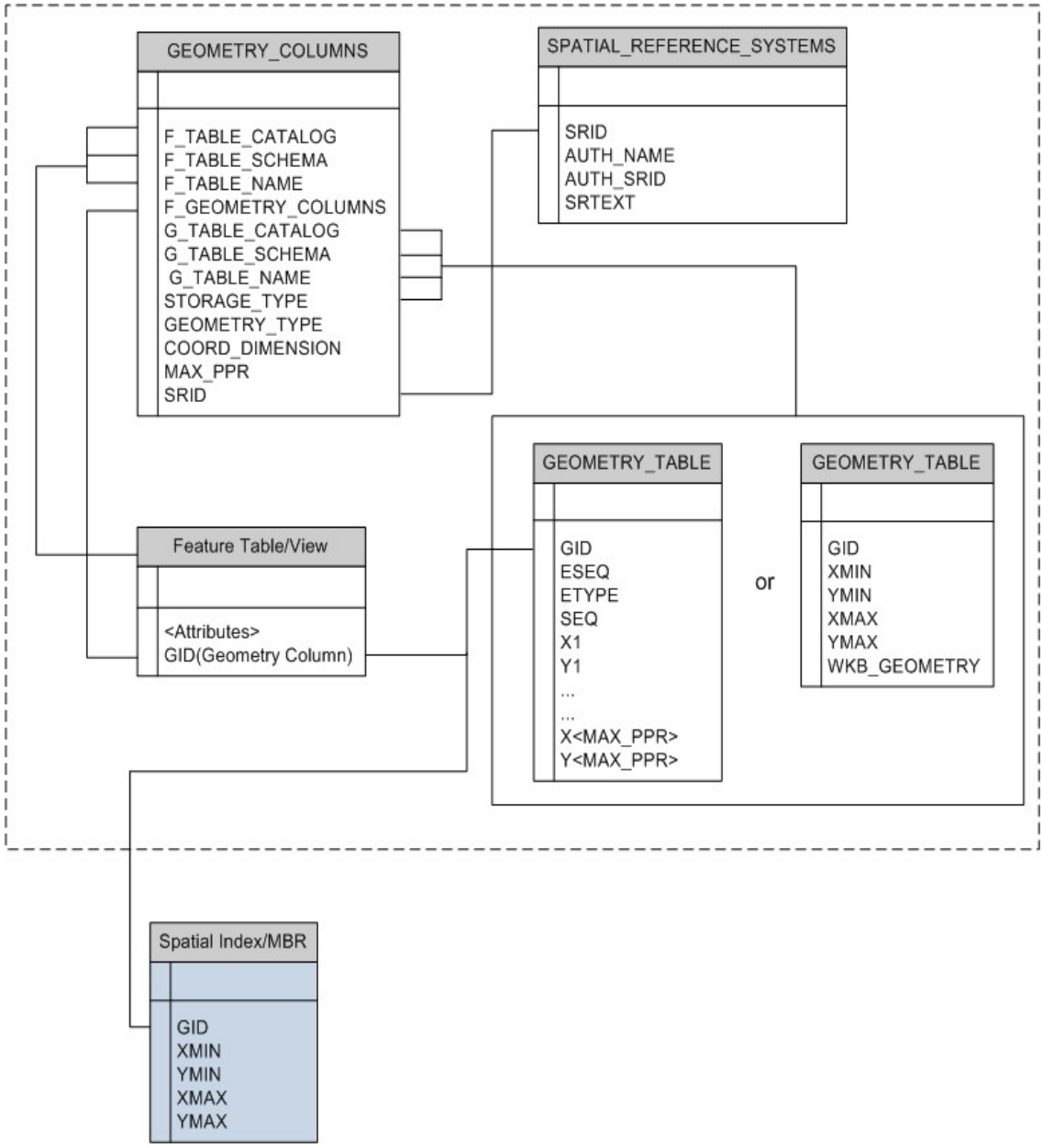

Figura 10: Estructura de tablas de la extensión JASPA (según estándar OpenGIS) que incluye la tabla índice creada.

La tabla índice creada dentro de esta implementación se ha relacionado con la estructura de tablas definida por la extensión *JASPA*, más concretamente se ha relacionado con la tabla que contiene los datos geométricos a través del campo *GID* por medio de una relación de 1:1, es decir, cada registro de la tabla de datos geométricos tiene asociado en la tabla índice un registro con el *MBR* que contiene a los datos geométricos de ese registro.

5.2 Diseño e Implementación

Se ha dividido el contenido de este capítulo en dos partes bien diferenciadas:

- *Parte 1*: destinada a recopilar las decisiones previas a la realización del diseño e implementación.

- *Parte 2*: Incluye la descripción de la Estructura de la solución.

5.2.1 Especificaciones

En esta primera parte se describen las decisiones más importantes que se han tomado para realizar este trabajo.

Se ha optado por no implementar desde cero el algoritmo *Árbol-R* al identificar que esto excede el alcance del PFC. Después de analizar diferentes implementaciones en java del algoritmo *Árbol-R*, se opta por usar la *JSI (Java Spatial Index) RTree Library* disponible en:

http://sourceforge.net/projects/jsi/files/jsi/1.0b6/jsi-1.0b6.zip/download?use_mirror=kent

Se trata de un proyecto de código abierto y liberado bajo la licencia *GNU Lesser General Public License* que tiene como objetivo mantener un alto rendimiento sobre la implementación en Java del algoritmo de indexación espacial *Árbol-R*. Esta librería realiza deliberadamente un conjunto reducido de funciones pero lo hace de manera muy eficiente.

Esta biblioteca contiene métodos que permiten:

- Añadir y eliminar entradas en un Árbol-R.

- Realizar consultas de intersección y contenido

La documentación de esta librería asegura la estabilidad de la implementación con diferentes tipos de tests.

El Árbol-R se genera en memoria cada vez que se activa un índice, en vez de cargarlo desde la base de datos, que es lo habitual. Esto se hizo así por dos motivos:

1. Extraer la información del árbol y transformarla en un formato adecuado para almacenarla en la base de datos consume tiempo de cálculo y la tabla almacenada es más grande que la tabla índice usada.

2. Dado que la tabla índice sólo almacena el *SRID* y el *MBR*, y que además está indexada, la recreación del índice es muy rápida y no necesita cálculos adicionales. Además una vez cargada en memoria se puede utilizar para sucesivas consultas.

Para usar la implementación del Árbol-R se han tenido que realizar las siguientes adaptaciones:

- Los métodos *contain* e *insersect* de la clase *rtree* tienen provisto un parámetro para ejecutar un método externo sobre los resultados de la búsqueda. Se han modificado para quitar dicho parámetro, renombrándolos como *getContainFunction* y *getIntersectFunction*.

- Se ha cambiado el numero de hijos de cada nodo por defecto. El original era 10 y se ha puesto en 6. Aunque según la documentación del algoritmo *Árbol-R* [13] esto es algo menos eficiente para muy grandes cantidades de objetos (>50000), la mayoría de las búsquedas involucran bastantes menos. Aunque este cambio incrementa en dos el número de niveles del *Rtree* para un conjunto del orden de 40000 objetos, reduce la carga correspondiente a recorrer secuencialmente los nodos hijo de cada nodo, por lo que es más eficiente para los tamaños de conjunto de datos habituales.

En cuanto a la generación de los *disparadores* (*triggers)* hay que mencionar que se han implementado a partir de la interfaz *Trigger* de *H2*.

Las nuevas funciones espaciales se han implementado empaquetándolas en un archivo *jar* e indicándole a *H2* dónde está el *jar* que contiene este código ejecutable.

5.2.2 Estructura de la implementación

A continuación se describen cada uno de los ficheros que contienen el código fuente de la solución propuesta:

- *CreateSpatialIndexH2.java:* Contiene la lógica de la aplicación, también puede ejecutarse de forma independiente desde una consola del sistema operativo para crear desde ahí un índice espacial sin necesidad de entrar en la consola de *H2*.

- *Function.java:* Fichero que incluye los métodos externos que pueden ser invocados por cualquier usuario desde la consola *Sql* de *H2* para el manejo de los índices espaciales.

- *TriggerSynchronizeSpatial.java:* Fichero que contiene los disparadores y sus acciones asociadas para mantener sincronizadas la tabla inicial de datos espaciales y la tabla que contiene el índice asociado.

- *Librerías externas*: Para la correcta ejecución del programa que se ha implementado se necesitan las siguientes librerías: *h2-1.2.139.jar*, *log4j.jar*, *trove.jar y jsi-1.0b6*

5.2.2.1 Descripción de la implementación de la clase CreateSpatialIndexH2

Esta clase es la más importante ya que contiene la lógica de la aplicación. A continuación, se describen los principales métodos de la aplicación creada.

Método rTreeView.

Permite recorrer y mostrar el contenido del árbol que se encuentra en memoria.

Admite como parámetro de entrada el árbol que se pretende recorrer. A continuación se indica el código fuente comentado del método:

```
public static void rTreeView(RTree arbolR)
        throws Exception {
                int i;
                int k;
                int h;
                int nLevel=0;
                int maxLevel=0;
                Node newNode = null;
                Node raizNode = null;
                Node childNode = null;

                // Se recorre el arbolR que se pasa como argumento y se muestra
                for (k=0; k<=arbolR.getHighestUsedNodeId(); k++){
```

```
                    newNode=arbolR.getNode(k);
                    System.out.println("Node id="+k+", Rectangle= "+
                                        Float.toString(newNode.mbrMinX)+":"+
                                        Float.toString(newNode.mbrMinY)+":"+
                                        Float.toString(newNode.mbrMaxX)+":"+
                                        Float.toString(newNode.mbrMaxY));
                    // SI el nodo es hoja, devuelve los 'id' de los registros.
                    // Si el nodo no es hoja, devuelve los identificadores de los nodos hijo.
                    System.out.print("   punteros: ");
                    for (h = 0; h < arbolR.maxNodeEntries; h++){
                            System.out.print("  "+newNode.ids[h]);
                            System.out.print("  Rectangle= ("+newNode.entriesMinX[h]+":"
                                    +newNode.entriesMinY[h]+":"+newNode.entriesMaxX[h]+":"
                                    +newNode.entriesMaxY[h]+")");
                                    System.out.println();
                    };
                    System.out.println("");
            };
    }
```

Método loadSpatialDataMBRFromDatabaseToMemory

Es el encargado de pasar a memoria (a la estructura de datos *Árbol-R*) el contenido de la tabla auxiliar del índice que contiene los *MBRs y los SRIDs*.

Este método admite como parámetros de entrada la conexión con la base de datos, el nombre de la tabla y el nombre del campo que contienen los datos geométricos a partir del cual se crea el índice.

Como salida este método devuelve el *Árbol-R*.

A continuación se indica el código fuente comentado de este método:

```
public static RTree loadSpatialDataMBRFromDatabaseToMemory(Connection conexion, String tableDb, String
columnTb)
        throws Exception {

                int id2;
                int level2;
                int maxLevel2=0;
                float Xmin=0;
                float Xmax=0;
                float Ymin=0;
                float Ymax=0;
                int pos1;
                int pos2;
```

```
int h;

Node newNode = null;
Node raizNode = null;
Node childNode = null;

String content;
//Object content;

arbolR = new RTree();
Properties          props = new Properties();

arbolR.init(props);
int[] ids2 = new int[arbolR.maxNodeEntries];
//arbol2R.highestUsedNodeId=mapSize;

Statement instruccion = conexion.createStatement();
ResultSet conjuntoResultados = instruccion.executeQuery("select * from " +
"SPATIALDATAMBRTABLE_" + tableDb +"_" +columnTb);

// Carga los datos (SRID y MBRs) que se leen de la tabla que contiene el índice al RTree.
while(conjuntoResultados.next()) {

        id2 = Integer.valueOf(conjuntoResultados.getObject(1).toString()).intValue() ;
        Xmin = Float.valueOf(conjuntoResultados.getObject(2).toString()).floatValue() ;
        Ymin = Float.valueOf(conjuntoResultados.getObject(3).toString()).floatValue() ;

        Xmax = Float.valueOf(conjuntoResultados.getObject(4).toString()).floatValue() ;
        Ymax = Float.valueOf(conjuntoResultados.getObject(5).toString()).floatValue() ;

        arbolR.add(new Rectangle(Xmin,Ymin,Xmax,Ymax),id2);


};
arbolR.rootNodeId=maxLevel2;
return arbolR;
```

Método insertMBRFromSpatialDataTableIntoSpatialDataMBRTable

Crea una tabla índice con los *MBRs* a partir del contenido de la tabla que contiene los datos espaciales y además los carga en memoria en el *Árbol-R*.

A continuación se indica el código fuente comentado del método:

public static void insertMBRFromSpatialDataTableIntoSpatialDataMBRTable(Connection conexion, String tableDb, String columnTb)

```
throws Exception {

        // Comprueba la posicion del campo geometrico dentro de la estructura de la tabla
        ResultSet rs2 = conexion.getMetaData().getColumns(null,null,tableDb,null);
        int pos = 0;
        String column="";
        String iDColName="";

        // Se recorren todos los campos de la tabla para comprobar si contiene un campo de igual nombre que el
pasado como argumento
        while (rs2.next()) {
           if (pos==0) iDColName = rs2.getString("COLUMN_NAME");
                column = rs2.getString("COLUMN_NAME");
                if (column.equals(columnTb)) break;
                else pos++;
        }

        // El primer campo de la tabla que contiene los datos geometricos debe ser la clave
        Statement stmt = conexion.createStatement();
        ResultSet conjuntoResultados = stmt.executeQuery("select " + iDColName + " ,ST_xmin(" + columnTb +
"),ST_ymin(" + columnTb + "),ST_xmax(" + columnTb + "),ST_ymax(" + columnTb + ") from " + tableDb);
    //stmt.close();
        // Se crea una estructura de árbol vacía
        arbolR = new RTree();
        Properties          props = new Properties();
        props.setProperty("maxNodeEntries","6");
        props.setProperty("minNodeEntries","3");

        // Se actualizan las propiedades de la estructura árbol creada.
        arbolR.init(props);

        int nSigNodos=20;
        int i;
        int k;
        int h;
        String l;
        int nLevel=0;
        int maxLevel=0;
        int childNumber=0;
        int childId=0;
        Node newNode = null;
        Node raizNode = null;
        Node childNode = null;

        i=1 ;
        Statement ps = conexion.createStatement();
        while(conjuntoResultados.next()) {

                float x0,y0,x1,y1;
```

```
                    // Se obtiene el Identificador de cada registro
                    int id = Integer.valueOf(conjuntoResultados.getObject(1).toString()).intValue() ;
                    //System.out.println("ID:"+conjuntoResultados.getObject(1).toString());


                    // Se calcula el MBR de un campo geomátrico
                    x0 = Float.valueOf(conjuntoResultados.getObject(2).toString()).floatValue() ;
                    //System.out.println("A:"+conjuntoResultados.getObject(2).toString());
                            y0 = Float.valueOf(conjuntoResultados.getObject(3).toString()).floatValue() ;
                    //System.out.println("B:"+conjuntoResultados.getObject(3).toString());
                            x1 = Float.valueOf(conjuntoResultados.getObject(4).toString()).floatValue() ;
                    //System.out.println("C:"+conjuntoResultados.getObject(4).toString());

                    y1 = Float.valueOf(conjuntoResultados.getObject(5).toString()).floatValue() ;
                    //System.out.println("D:"+conjuntoResultados.getObject(5).toString());

                    // Se va cargando el ArbolR a partir de los datos espaciales
                    arbolR.add(new Rectangle(x0,y0,x1,y1),id);

                    // También se va cargando en la tabla indice los MBRs de los datos que se van leyendo de la tabla
de datos
                    ps.execute("INSERT INTO SpatialDataMBRTable_"+ tableDb + "_" + columnTb + " VALUES("
+Integer.toString(id) + ","+ Float.toString(x0) + ","+ Float.toString(y0)+
","+Float.toString(x1)+","+Float.toString(y1)+")");


                    }
            stmt.close();
                    ps.close();

            }
```

Método createSpatialIndex

Se encarga de crear el índice espacial, creando la tabla que almacenará los *MBRs* y *SRIDs* de los objetos y también los disparadores que mantendrán sincronizada la tabla de datos geométricos con la nueva tabla creada.

Esta función admite como parámetros la conexión con la base de datos, el nombre de la tabla, el nombre del campo que contienen los datos geométricos y devuelve un valor booleano para indicar si la operación se ha realizado correctamente.

El nombre de la tabla índice se formará automáticamente a partir de los datos anteriores de la siguiente forma:

SPATIALDATAMBRTABLE_+ *nombre de la tabla a indexar + carácter de subrayado + nombre de la columna que contiene los datos geométricos*.

Ejemplo: ***SPATIALDATAMBRTABLE_SOILS_GEOM***
(*Esta tabla contiene el índice de la columna geométrica GEOM que pertenece a la tabla SOILS)*

A continuación se muestra el código fuente:

```
public static boolean createSpatialIndex(Connection conexion, String tableDb, String columnTb) throws
Exception {

        // Acceder a la tabla de metadatos y comprueba si el campo a indexar existe como columna geometrica
        Statement stmt = conexion.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT count(*) as cont from
JASPA.GEOMETRY_COLUMNS_EXTEND WHERE (F_TABLE_NAME='"+ tableDb.toUpperCase()+ "' or
F_TABLE_NAME='" + tableDb.toLowerCase() +"') and (F_GEOMETRY_COLUMN='" + columnTb.toUpperCase()
+ "' or F_GEOMETRY_COLUMN='" + columnTb.toLowerCase()+"')");

        rs.next();
        int num = rs.getInt("cont");
        System.out.println(num);
        stmt.close();

        // El campo a indexar existe
        if (num>0) {

                //Modifica las tablas de los metadatos de JASPA para activar el indice
                try {
                        Statement stmt2 = conexion.createStatement();
                        stmt2.executeUpdate("UPDATE JASPA.GEOMETRY_COLUMNS_EXTEND SET
F_SPATIAL_INDEX_DEDICATED_COLUMN='TRUE' WHERE(F_TABLE_NAME='"+ tableDb.toUpperCase()+ "'
or F_TABLE_NAME='" + tableDb.toLowerCase() +"') and (F_GEOMETRY_COLUMN='" +
columnTb.toUpperCase()+ "' or F_GEOMETRY_COLUMN='" + columnTb.toLowerCase()+"')");
                        stmt2.close();
                } catch(SQLException ex) {
                        System.err.println("SQLException: " + ex.getMessage());
                }

                // Se crea la tabla que alojara el indice espacial
                try {
                        Statement stmt3 = conexion.createStatement();
                                stmt3.execute("DROP TABLE IF EXISTS SpatialDataMBRTable_" + tableDb
+"_" +columnTb + ";CREATE TABLE SpatialDataMBRTable_" + tableDb +"_" +columnTb + "(SGID INT
PRIMARY KEY,XMIN REAL, YMIN REAL, XMAX REAL, YMAX REAL)");
                        stmt3.close();
                        System.out.println();
                        System.out.println("La estructura de la tabla indice se ha creado correctamente y se
```

```
llama: SpatialDataMBRTable_" + tableDb + "_" + columnTb);
                        System.out.println();
                } catch(SQLException ex) {
                        System.err.println("SQLException: " + ex.getMessage());
                }

                //Recorre la tabla que tiene los datos geometricos y genera otra con los MBRs de los datos
                insertMBRFromSpatialDataTableIntoSpatialDataMBRTable(conexion, tableDb, columnTb);
                System.out.println();
                System.out.println("La tabla indice se ha rellenado convenientemente con los MBRs de los datos
geometricos y se han cargado en memoria");
                System.out.println();

                // Sincroniza la tabla de datos con el indice
                try {
                Statement stmt4 = conexion.createStatement();
                        stmt4.execute("DROP ALIAS IF EXISTS TRIGGER_SET_" + tableDb + "_"
+columnTb + ";CREATE ALIAS TRIGGER_SET_" + tableDb +"_" +columnTb + " FOR \"" +
TriggerSynchronizeSpatialIndex.class.getName() + ".setTriggerData\"");

                        stmt4.close();
                } catch(SQLException ex) {
                        System.err.println("SQLException: " + ex.getMessage());
                }
                //Crea un trigger para actualizar la tabla de los MBRs cuando se inserte un registro en la tabla
de datos
                //Una misma tabla puede tener varios triggers, una por cada columna geometrica que se quiera
indexar.
                try {
                        Statement stmt5 = conexion.createStatement();

                        stmt5.execute("DROP TRIGGER IF EXISTS " + tableDb +"_" +columnTb + "_INS;
CREATE TRIGGER "+ tableDb +"_" +columnTb +"_INS AFTER INSERT ON " + tableDb + " FOR EACH ROW
CALL \"TriggerSynchronizeSpatialIndex\" ");
                        stmt5.close();
                } catch(SQLException ex) {
                        System.err.println("SQLException: " + ex.getMessage());
                }
                //Crea un trigger para sincronizar la tabla de los MBRs cuando se elimine un registro en la tabla
de datos
                try {
                        Statement stmt6 = conexion.createStatement();
                                stmt6.execute("DROP TRIGGER IF EXISTS " + tableDb +"_" +columnTb +
"_DEL; CREATE TRIGGER "+ tableDb +"_" +columnTb +"_DEL AFTER DELETE ON " + tableDb + " FOR
EACH ROW CALL \"TriggerSynchronizeSpatialIndex\" ");
                        stmt6.close();
                } catch(SQLException ex) {
                        System.err.println("SQLException: " + ex.getMessage());
                }
```

```
		//Crea un trigger para sincronizar la tabla de los MBRs cuando se modifique un registro en la tabla de datos
		try {
			Statement stmt7 = conexion.createStatement();
				stmt7.execute("DROP TRIGGER IF EXISTS " + tableDb +"_" +columnTb + "_UPD; CREATE TRIGGER "+ tableDb +"_" +columnTb +"_UPD AFTER UPDATE ON " + tableDb + " FOR EACH ROW CALL \"TriggerSynchronizeSpatialIndex\" ");
			stmt7.close();
		} catch(SQLException ex) {
			System.err.println("SQLException: " + ex.getMessage());
		}

		//El trigger realiza diferentes actuaciones dependiendo del tipo de acción realizada en la tabla de datos: Insertar, Eliminar o Actualizar un dato
		try {
			Statement stmt8 = conexion.createStatement();
				stmt8.execute("CALL TRIGGER_SET_"+ tableDb +"_" +columnTb +"("+ tableDb +"_" +columnTb + "_INS',"+columnTb+")");
			stmt8.close();
		} catch(SQLException ex) {
			System.err.println("SQLException: " + ex.getMessage());
		}


		try {
			Statement stmt9 = conexion.createStatement();
				stmt9.execute("CALL TRIGGER_SET_"+ tableDb +"_" +columnTb +"("+ tableDb +"_" +columnTb + "_DEL',"+columnTb+")");
			stmt9.close();
		} catch(SQLException ex) {
			System.err.println("SQLException: " + ex.getMessage());
		}

		try {
		Statement stmt10 = conexion.createStatement();
		stmt10.execute("CALL TRIGGER_SET_"+ tableDb +"_" +columnTb +"("+ tableDb +"_" +columnTb + "_UPD',"+columnTb+")");
		stmt10.close();
	} catch(SQLException ex) {
			System.err.println("SQLException: " + ex.getMessage());
		}


		System.out.println();
		System.out.println("Los triggers asociados a la tabla de datos y a la tabla indice se han creado y activado correctamente");
		System.out.println();
```

```
		// Se crean los alias asociados a las nuevas funciones creadas:
ST_CREATESPINDEX,ST_DROPSPINDEX,ST_LOADSPINDEX y ST_USESPINDEX
		try {
			Statement stmt12 = conexion.createStatement();

			stmt12.execute("CREATE ALIAS IF NOT EXISTS JASPA.ST_CREATE_SPIDX
FOR \"Function.st_createSpIndex\" ");
			stmt12.close();
		} catch(SQLException ex) {
			System.err.println("SQLException: " + ex.getMessage());
		}
		try {
			Statement stmt12 = conexion.createStatement();
				stmt12.execute("CREATE ALIAS IF NOT EXISTS JASPA.ST_DROP_SPIDX
FOR \"Function.st_dropSpIndex\" ");
			stmt12.close();
		} catch(SQLException ex) {
			System.err.println("SQLException: " + ex.getMessage());
		}

		try {
			Statement stmt11 = conexion.createStatement();

			stmt11.execute("CREATE ALIAS IF NOT EXISTS JASPA.ST_LOAD_SPIDX
FOR \"Function.st_loadSpIndex\" ");
			stmt11.close();
		} catch(SQLException ex) {
			System.err.println("SQLException: " + ex.getMessage());
		}

		try {
			Statement stmt11 = conexion.createStatement();

				stmt11.execute("CREATE ALIAS IF NOT EXISTS JASPA.ST_CLEAN_SPIDX
FOR \"Function.st_cleanSpIndex\" ");
			stmt11.close();
		} catch(SQLException ex) {
			System.err.println("SQLException: " + ex.getMessage());
		}

		try {
			Statement stmt12 = conexion.createStatement();
				stmt12.execute("CREATE ALIAS IF NOT EXISTS
JASPA.ST_CONTAIN_SPIDX FOR \"Function.st_containSpIndex\" ");
			stmt12.close();
		} catch(SQLException ex) {
			System.err.println("SQLException: " + ex.getMessage());
		}
```

```
                try {
                        Statement stmt12 = conexion.createStatement();
                        stmt12.execute("CREATE ALIAS IF NOT EXISTS JASPA.ST_INTERSECT_SPIDX FOR
\"Function.st_intersectSpIndex\" ");
                        stmt12.close();
                } catch(SQLException ex) {
                        System.err.println("SQLException: " + ex.getMessage());
                }

                /*try {
                        Statement stmt12 = conexion.createStatement();

                        stmt12.execute("CREATE ALIAS IF NOT EXISTS JASPA.ST_NEAR_SPIDX
FOR \"Function.st_nearSpIndex\" ");
                        stmt12.close();
                } catch(SQLException ex) {
                        System.err.println("SQLException: " + ex.getMessage());
                }*/

                System.out.println();
                System.out.println("Los alias
ST_CREATE_SPIDX,ST_DROP_SPIDX,ST_CLEAN_SPIDX,ST_LOAD_SPIDX,ST_CONTAIN_SPIDX y
ST_INTERSECT_SPIDX se han creado satisfactoriamente y pueden ser invocados desde la consola de H2");
                System.out.println();
                return true;
        }
        // La tabla o campo a indexar no existe
        else {
                System.out.println("Error: La tabla a indexar no existe o la columna a indexar no existe o no es
de tipo geometrico");
           return false;
        }
}
```

Método dropSpatialIndex

Se encarga de eliminar el índice espacial, eliminando la tabla que almacenó los *MBRs* de los objetos y también los disparadores que mantenían sincronizada la tabla de datos geométricos con la nueva tabla creada.

Esta función admite como parámetros la conexión con la base de datos, el nombre de la tabla, el nombre del campo que contienen los datos geométricos y devuelve un valor booleano para indicar si la operación se ha realizado correctamente.

A continuación se muestra el código fuente:

```
public static boolean dropSpatialIndex(Connection conexion, String tableDb, String columnTb) throws Exception
{

	// Acceder a la tabla de metadatos y comprueba si el campo a indexar existe como columna geometrica
	Statement stmt = conexion.createStatement();
	ResultSet rs = stmt.executeQuery("SELECT count(*) as cont from
JASPA.GEOMETRY_COLUMNS_EXTEND WHERE (F_TABLE_NAME='"+ tableDb.toUpperCase()+ "' or
F_TABLE_NAME='" + tableDb.toLowerCase() +"') and (F_GEOMETRY_COLUMN='" + columnTb.toUpperCase()
+ "' or F_GEOMETRY_COLUMN='" + columnTb.toLowerCase()+"')");

	rs.next();
	int num = rs.getInt("cont");
	System.out.println(num+ "t:"+tableDb +"c:"+columnTb);
	stmt.close();

	// El campo a indexar existe
	if (num>0) {
		//Modifica las tablas de los metadatos de JASPA para activar el indice
		try {
			Statement stmt2 = conexion.createStatement();
			stmt2.executeUpdate("UPDATE JASPA.GEOMETRY_COLUMNS_EXTEND SET
F_SPATIAL_INDEX_DEDICATED_COLUMN='FALSE' WHERE(F_TABLE_NAME='"+ tableDb.toUpperCase()+
"' or F_TABLE_NAME='" + tableDb.toLowerCase() +"') and (F_GEOMETRY_COLUMN='" +
columnTb.toUpperCase()+ "' or F_GEOMETRY_COLUMN='" + columnTb.toLowerCase()+"')");
			stmt2.close();
		} catch(SQLException ex) {
			System.err.println("SQLException: " + ex.getMessage());
		}

		// Se elimina la tabla que aloja el indice espacial
		try {
			Statement stmt3 = conexion.createStatement();
			stmt3.execute("DROP TABLE IF EXISTS SpatialDataMBRTable_" + tableDb +"_"
+columnTb );
			stmt3.close();
			System.out.println();
			System.out.println("La estructura de la tabla indice se ha elimado correctamente y se
llamaba: SpatialDataMBRTable_" + tableDb + "_" + columnTb);
			System.out.println();
		} catch(SQLException ex) {
			System.err.println("SQLException: " + ex.getMessage());
		}

		// Se eliminan los triggers asociados
		try {
			Statement stmt4 = conexion.createStatement();
			stmt4.execute("DROP ALIAS IF EXISTS TRIGGER_SET_" + tableDb + "_"
+columnTb);
			stmt4.close();
```

```
                } catch(SQLException ex) {
                        System.err.println("SQLException: " + ex.getMessage());
                }
                //Crea un trigger para actualizar la tabla de los MBRs cuando se inserte un registro en la tabla
de datos
                //Una misma tabla puede tener varios triggers, una por cada columna geometrica que se quiera
indexar.
                try {
                        Statement stmt5 = conexion.createStatement();
                        stmt5.execute("DROP TRIGGER IF EXISTS " + tableDb +"_" +columnTb + "_INS");
                        stmt5.close();
                } catch(SQLException ex) {
                        System.err.println("SQLException: " + ex.getMessage());
                }
                //Crea un trigger para sincronizar la tabla de los MBRs cuando se elimine un registro en la tabla
de datos
                try {
                        Statement stmt6 = conexion.createStatement();

                        stmt6.execute("DROP TRIGGER IF EXISTS " + tableDb +"_" +columnTb + "_DEL");
                        stmt6.close();
                } catch(SQLException ex) {
                        System.err.println("SQLException: " + ex.getMessage());
                }

                //Crea un trigger para sincronizar la tabla de los MBRs cuando se modifique un registro en la
tabla de datos
                try {
                        Statement stmt7 = conexion.createStatement();

                        stmt7.execute("DROP TRIGGER IF EXISTS " + tableDb +"_" +columnTb + "_UPD");
                        stmt7.close();
                } catch(SQLException ex) {
                        System.err.println("SQLException: " + ex.getMessage());
                }

                System.out.println();
                System.out.println("Los triggers asociados a la tabla de datos y a la tabla indice se han
eliminado correctamente");
                System.out.println();
         return true;
        }
        // La tabla o campo a indexar no existe
        else {
                System.out.println("Error: La tabla a eliminar no existe o la columna a indexar no existe o no es
de tipo geometrico");
                return false;
        }
```

}

Método getContainFunction

Obtiene los identificadores de aquellos objetos que se encuentran completamente contenidos dentro de una ventana o rectángulo dado.

El algoritmo usa una pila y una cola para recorrer el árbol. La pila (lista de datos donde el último elemento en entrar es el primero en salir) se usa para subir y bajar en el árbol. La cola (lista de datos donde el primer elemento en entrar es el primero en salir) se usa para recorrer las entradas de los nodos hijos de un nodo determinado.

El algoritmo recorre el árbol desde el nodo raíz, y baja al nodo hijo si su *MBR* intersecta con el rectángulo de búsqueda. Si no es así se sigue con el siguiente hijo. En el nodo hijo, se sigue buscando en los nodos hijos, y así hasta que llega a un nodo hoja. En el nodo hoja se comprueba si los *MBRs* de las entradas del nodo hoja están dentro del rectángulo de búsqueda, y si es así introduce los *IDs* de las entradas en el conjunto resultado. Posteriormente se sigue recorriendo el árbol hasta su finalización.

Esta función admite como parámetros la conexión con la base de datos, el nombre de la tabla, el nombre del campo que contienen los datos geométricos y las coordenadas de los vértices inferior izquierdo y superior derecho del rectángulo a partir del cual se desean calcular los objetos que están en él contenidos.

A continuación se muestra el código fuente:

```
public static SimpleResultSet getContainFunction(Connection conexion, String tableName, String field, float
Xmin, float Ymin, float Xmax,float Ymax) throws Exception {
        int sgid;
        float x0,y0,x1,y1;

        // Se crea la estructura de los registros que almacenará los identificadores de los registros
        // como resultado de la consulta
        SimpleResultSet rs4 = new SimpleResultSet();
        rs4.addColumn("ID", Types.INTEGER, 10, 0);


        //      Implementación de la búsqueda de todos los objetos contenidos completamente
        //      dentro de un rectángulo dado.
        //      Se introducen en el array IDsIn[] todos los SRID de los objetos que están contenidos dentro
        //      del rectánguo que se pide.
        //      El numero de SRID devueltos es IDsCont. Si no hay ninguno, queda IDsCont=0
        //      (Basado en el método RTree.delete)

        int startIndex=1;
```

```
        int IDsIn[] = new int[1000000];
        int IDsCont=0;
        int i,j,k,h;
        TIntStack parents = new TIntStack();
        TIntStack parentsEntry = new TIntStack();


        parents.reset();
        //parents.push(a.getRootNodeId());
        parents.push(nRoot);

        parentsEntry.reset();
        parentsEntry.push(-1);
        Node n = null;
        IDsCont=0;

        Node nRaiz = a.getNode(a.getRootNodeId());

        //System.out.println("ROOTNODEID:--"+ nRoot + " LEVEL--: "+ nlevel +" HIGHT:"+ a.getHighestUsedNodeId()
+"Antes:" + Xmin +"-"+Ymin +"-"+Xmax +"-"+Ymax +"-" );

        while (parents.size() > 0){
                n = a.getNode(parents.peek());
                startIndex = parentsEntry.peek() + 1;
                if (!n.isLeaf()){
                        boolean contains = false;
                        for (i = startIndex; i < n.entryCount; i++){
                                if (Rectangle.intersects(Xmin, Ymin, Xmax, Ymax, n.entriesMinX[i],
                                        n.entriesMinY[i], n.entriesMaxX[i],n.entriesMaxY[i])){
                                        parents.push(n.ids[i]);
                                        parentsEntry.pop();
                                        parentsEntry.push(i);
                                        parentsEntry.push(-1);
                                        contains=true;
                                        break;
                                };
                        };
                        if (contains) { continue; };
                } else{
                        for (h = 0; h < n.entryCount; h++){
                                if ( (Xmin <= n.entriesMinX[h]) && (Ymin <= n.entriesMinY[h]) &&
                                        (Xmax >= n.entriesMaxX[h]) && (Ymax >= n.entriesMaxY[h]) )
                                {
                                        // Encontrado objeto
                                        IDsCont++;
                                        IDsIn[IDsCont]=n.ids[h];
                                };
                    };
//                      parentsEntry.pop();
                };
                parents.pop();
                parentsEntry.pop();
        };

        // Presenta los objetos encontrados o una indicacion de que no se ha localizado
```

```
        if (IDsCont>0){
        //          System.out.println("Objetos que tiene dentro el Rectangulo:");
                    for (k=1; k<=IDsCont; k++){
                                System.out.println("Encontrado Objeto "+k+" SRID="+IDsIn[k]);
                                rs4.addRow(IDsIn[k]);
                    };
        } else{
                     System.out.println("No se ha encontrado ningun objeto dentro del Rectangulo"+a.size() );
        }

        // Devuelve los identificadores (en forma de registros) que cumplen los criterios de la consulta espacial realizada
        return rs4;


}
```

Método getIntersectFunction

Obtiene los identificadores de aquellos objetos que se encuentran parcial o completamente contenidos dentro de una ventana o rectángulo dado.

El algoritmo usa una pila y una cola para recorrer el árbol y se utilizan de igual forma que en el método anterior.

El algoritmo recorre el árbol desde el nodo raíz, y baja al nodo hijo si su *MBR* intersecta con el rectángulo de búsqueda. Si no es así se sigue con el siguiente hijo. En el nodo hijo, se sigue buscando en los nodos hijos, y así hasta que llega a un nodo hoja. En el nodo hoja, se comprueba si los *MBRs* de las entradas del nodo hoja intersectan con el rectángulo de búsqueda, y si es así introduce los *IDs* de las entradas en el conjunto resultado. Posteriormente se sigue recorriendo el árbol hasta su finalización.

Esta función admite como parámetros la conexión con la base de datos, el nombre de la tabla, el nombre del campo que contienen los datos geométricos y las coordenadas de los vértices inferior izquierdo y superior derecho del rectángulo a partir del cual se desean calcular los objetos que están en él contenidos total o parcialmente.

A continuación se muestra el código fuente:

```
public static SimpleResultSet getIntersectFunction(Connection conexion, String tableName, String field, float
Xmin, float Ymin, float Xmax,float Ymax) throws Exception {

        int sgid;
        float x0,y0,x1,y1;
```

```
// Se crea la estructura de los registros que almacenará los identificadores de los registros
// como resultado de la consulta
SimpleResultSet rs4 = new SimpleResultSet();
rs4.addColumn("ID", Types.INTEGER, 10, 0);


//        Implementación de la búsqueda de todos los objetos contenidos parcial o completamente
//        dentro de un rectángulo dado.
//        Se introducen en el array IDsIn[] todos los SRID de los objetos que están contenidos dentro
//        del rectánguo que se pide.
//        El numero de SRID devueltos es IDsCont. Si no hay ninguno, queda IDsCont=0
//        (Basado en el método RTree.delete)
int startIndex=1;
int IDsIn[] = new int[1000000];
int IDsCont=0;
int i,j,k,h;
TIntStack parents = new TIntStack();
TIntStack parentsEntry = new TIntStack();


parents.reset();
//parents.push(a.getRootNodeId());
parents.push(nRoot);

parentsEntry.reset();
parentsEntry.push(-1);
Node n = null;
IDsCont=0;

Node nRaiz = a.getNode(a.getRootNodeId());

//System.out.println("ROOTNODEID:--"+ nRoot + " LEVEL--: "+ nlevel +" HIGHT:"+
a.getHighestUsedNodeId() +"Antes:" + Xmin +"-"+Ymin +"-"+Xmax +"-"+Ymax +"-" );

while (parents.size() > 0){
        n = a.getNode(parents.peek());
        startIndex = parentsEntry.peek() + 1;
        if (!n.isLeaf()){
                boolean intersects = false;
                for (i = startIndex; i < n.entryCount; i++){
                        if (Rectangle.intersects(Xmin, Ymin, Xmax, Ymax, n.entriesMinX[i],
                                n.entriesMinY[i], n.entriesMaxX[i],n.entriesMaxY[i])){
                                parents.push(n.ids[i]);
                                parentsEntry.pop();
                                parentsEntry.push(i);
                                parentsEntry.push(-1);
                                intersects=true;
                                break;
                        };
```

```
                        };
                        if (intersects) { continue; };
                } else{
                                for (h = 0; h < n.entryCount; h++){
                                if (Rectangle.intersects(Xmin, Ymin, Xmax, Ymax, n.entriesMinX[h],
                                        n.entriesMinY[h], n.entriesMaxX[h],n.entriesMaxY[h])){
                                                // Encontrado objeto
                                                IDsCont++;
                                                IDsIn[IDsCont]=n.ids[h];
                                };
                };

                };
                parents.pop();
                parentsEntry.pop();
        };
                // Presenta los objetos encontrados o una indicacion de que no se ha localizado
        if (IDsCont>0){
        //      System.out.println("Objetos que tiene dentro el Rectangulo:");
                for (k=1; k<=IDsCont; k++){
                        System.out.println("Encontrado Objeto "+k+" SRID="+IDsIn[k]);
                        rs4.addRow(IDsIn[k]);
                };
        } else{
                         System.out.println("No se ha encontrado ningun objeto dentro del
Rectangulo"+a.size() );
        }

        // Devuelve los identificadores (en forma de registros) que cumplen los criterios de la consulta espacial
realizada
        return rs4;
}
```

5.2.2.2 Descripción de la Implementación de los Triggers en H2

Los *disparadores* o *triggers* se encargarán de mantener sincronizada la tabla original que contiene los datos geométricos y su respectiva tabla índice.

La implementación de los métodos relacionados con los disparadores se encuentra localizados en el fichero *TriggerSynchronizeSpatialIndex.java*

A continuación, se indica el código fuente del método más representativo donde se puede comprobar cómo se gestionan las operaciones de inserción, actualización y eliminación en la tabla de datos espaciales.

public void fire(Connection conn, Object[] old, Object[] row) throws SQLException{

```
        // Muestra el tipo de operacion realizada sobre la tabla que está asociada al trigger
        System.out.println("Tipo de operacion:"+trtype);
        System.out.println("Trigerdata disparo:"+triggerData);

        // Se ha detectado una inserción en la tabla de datos y a continuación se realiza una inserción en la tabla
índice
        if (trtype==1) {
                String geomFromTrigger      =  byteArrayToString((byte[]) row[pos]);
                try {
                        PreparedStatement prep = conn.prepareStatement("INSERT INTO
SpatialDataMBRTable_"+ tbname + "_" + triggerData+" VALUES("+row[0]+",ST_xmin(\'" + geomFromTrigger +
"\'),ST_ymin(\'" + geomFromTrigger + "\'),ST_xmax(\'" + geomFromTrigger + "\'),ST_ymax(\'" + geomFromTrigger
+ "\'))");
                        prep.execute();
                        prep.close();
                } catch(SQLException ex) {
                        System.err.println("SQLException: " + ex.getMessage());
                }
        }
                // Se ha detectado una actualización en la tabla de datos y a continuación se realiza una
inserción en la tabla índice
                if (trtype==2) {
                        String geomFromTrigger      =  byteArrayToString((byte[]) row[pos]);
                        try {
                                PreparedStatement prep = conn.prepareStatement("UPDATE
SpatialDataMBRTable_"+ tbname + "_" +triggerData+" SET XMIN = ST_xmin(\'" + geomFromTrigger + "\'),
YMIN = ST_ymin(\'" + geomFromTrigger + "\'), XMAX = ST_xmax(\'" + geomFromTrigger + "\'), YMAX =
ST_ymax(\'" + geomFromTrigger + "\') WHERE SGID=" + row[0]);
                                prep.execute();
                                prep.close();
                        } catch(SQLException ex) {
                                System.err.println("SQLException: " + ex.getMessage());
                        }
                }

                // Se ha detectado la eliminación de un registro en el fichero de datos y a continuación se realiza
la eliminación del registro en la tabla índice
                if (trtype==4) {
                        try {
                                PreparedStatement prep = conn.prepareStatement("DELETE FROM
SpatialDataMBRTable_"+ tbname + "_" +triggerData+" WHERE SGID="+ old[0]);
                                prep.execute();
                                prep.close();
                        } catch(SQLException ex) {
                                System.err.println("SQLException: " + ex.getMessage());
                        }
                }
                System.out.println("Tabla en fire:" + tbname + " Campo:" + triggerData );
        }
```

5.2.2.3 Descripción de la Implementación de las nuevas funciones en H2

Se han creado una serie de métodos externos (incluidos en la clase *Function.java)* que pueden ser invocados por cualquier usuario desde la consola *Sql* de *H2* para el manejo de los índices espaciales. Cada método permite una acción concreta sobre un índice espacial. A continuación (véase la tabla 3) se indica el nombre de cada método, la acción que realiza y el nombre del alias que se le ha dado para facilitar su uso desde consola.

Nombre del Método externo	Alias	Descripción
st_createSpIndex	ST_CREATE_SPIDX	*Permite crear la tabla con los MBRs a partir de la tabla que contiene los datos espaciales.*
st_loadSpIndex	ST_LOAD_SPIDX	*Permite cargar el contenido de la tabla que contiene los MBRs en memoria (en una estructura de datos de Árbol-R).*
st_containSpIndex	ST_CONTAIN_SPIDX	*Permite realizar una consulta espacial para calcular los objetos completamente contenidos dentro de una ventana o rectángulo dado.*
st_intersectSpIndex	ST_INTERSECT_SPIDX	*Permite realizar una consulta espacial para calcular los objetos parcial o completamente contenidos dentro de una ventana o rectángulo dado.*
st_cleanSpIndex	ST_CLEAN_SPIDX	*Permite vaciar la estructura de datos de memoria que contiene el Árbol-R.*
st_dropSpIndex	ST_DROP_SPIDX	*Permite eliminar un índice espacial.*

Tabla 3: Descripción de las funciones creadas por la aplicación para el manejo de un índice espacial.

A continuación, se describe cada uno de los métodos y se muestra su código fuente.

La primera de ellas, **ST_CREATE_SPIDX** se encarga de crear el índice espacial, creando la tabla que almacenará los *MBRs* de los objetos y también los disparadores que mantendrán sincronizada la tabla de datos geométricos con la nueva tabla creada.

Esta función admite como parámetros la conexión con la base de datos, el nombre de la tabla y el nombre del campo que contienen los datos geométricos a partir del cual se creará el índice y devuelve un valor *booleano* indicando si la operación se ha ejecutado correctamente.

A continuación se muestra el código fuente comentado de esta función:

```
public static boolean st_createSpIndex(Connection conn, String tableName, String field) throws Exception {

        String tb = tableName;
        String fd = field;
        String idx = "SpatialDataMBRTable_" + tb + "_" + fd;

        // Carga el contenido de la tabla que contiene los MBRs en memoria en la estructura del ArbolR.
return CreateSpatialIndexH2.createSpatialIndex(conn, tb, fd);

}
```

La siguiente función, **ST_LOAD_SPIDX** se encargará de pasar el contenido de la tabla creada a partir de la función **ST_CREATE_SPIDX** en un *Árbol-R* en memoria principal, con la idea de realizar las consultas espaciales sobre esta estructura de datos que al estar en memoria permitirá obtener los resultados mucho más rápido que accediendo a memoria secundaria.

Esta función admite como parámetros la conexión con la base de datos, el nombre de la tabla y el nombre del campo que contienen los datos geométricos y devuelve un valor *booleano* indicando si la operación se ha ejecutado correctamente.

A continuación se muestra el código fuente comentado de esta función:

```
public static boolean st_loadSpIndex(Connection conn, String tableName, String field) throws Exception {

        String tb = tableName;
        String fd = field;
        String idx = "SpatialDataMBRTable_" + tb + "_" + fd;
```

```
        // Carga el contenido de la tabla que contiene los MBRs en memoria en la estructura del ArbolR.
        a = CreateSpatialIndexH2.loadSpatialDataMBRFromDatabaseToMemory(conn, tb, fd);

        nRoot=0;
        nlevel=0;

        for (int i=0; i <= a.getHighestUsedNodeId();i++){
                Node newNode = a.getNode(i);
                if (newNode.level>nlevel){
                                nlevel=newNode.level;
                                nRoot=i;
                        }
                }
        return true;
}
```

La siguiente función, **ST_CONTAIN_SPIDX** permite realizar una consulta espacial, exactamente devuelve los objetos completamente contenidos dentro de una ventana o rectángulo dado.

Esta función admite como parámetros la conexión con la base de datos, el nombre de la tabla, el nombre del campo que contienen los datos geométricos y las coordenadas de los vértices inferior izquierdo y superior derecho del rectángulo a partir del cual se desean calcular los objetos que están en él contenidos. Devuelve los identificadores de los objetos encontrados.

A continuación se muestra el código fuente comentado de esta función:

```
public static SimpleResultSet st_containSpIndex(Connection conn, String tableName, String field, float Xmin,
float Ymin, float Xmax,float Ymax) throws Exception {

        String tb = tableName;
        String fd = field;
        String idx = "SpatialDataMBRTable_" + tb + "_" + fd;

        SimpleResultSet rs = new SimpleResultSet();
        rs = CreateSpatialIndexH2.getContainFunction(conn, tb, fd, Xmin, Ymin, Xmax, Ymax );

        return rs;
        }
```

La siguiente función, **ST_INTERSECT_SPIDX** permite realizar una consulta espacial, exactamente devuelve los objetos total o parcialmente contenidos dentro de una ventana

o rectángulo dado.

Esta función admite como parámetros la conexión con la base de datos, el nombre de la tabla, el nombre del campo que contienen los datos geométricos y las coordenadas de los vértices inferior izquierdo y superior derecho del rectángulo a partir del cual se desean calcular los objetos que están total o parcialmente en él contenidos. Devuelve los identificadores de los objetos encontrados.

A continuación se muestra el código fuente comentado de esta función:

```
public static ResultSet st_intersectSpIndex(Connection conn, String tableName, String field, float Xmin, float
Ymin, float Xmax,float Ymax) throws Exception {

        String tb = tableName;
        String fd = field;
        String idx = "SpatialDataMBRTable_" + tb + "_" + fd;

        SimpleResultSet rs = new SimpleResultSet();
        rs = CreateSpatialIndexH2.getIntersectFunction(conn, tb, fd, Xmin, Ymin, Xmax, Ymax );

        return rs;

}
```

La siguiente función, **ST_CLEAN_SPIDX** permite vaciar el *Árbol-R* que se encuentra en memoria con el índice activo.

Esta función admite como parámetros la conexión con la base de datos, el nombre de la tabla y el nombre del campo que contienen los datos geométricos. Devuelve un valor *booleano* indicando si la operación se ha ejecutado correctamente.

A continuación se muestra el código fuente comentado de esta función:

```
public static boolean st_cleanSpIndex(Connection conn, String tableName, String field) throws Exception {

        String tb = tableName;
        String fd = field;
        String idx = "SpatialDataMBRTable_" + tb + "_" + fd;
        a=null ;
        return true;
}
```

Por último, la función **ST_DROP_SPIDX** se encarga de eliminar el índice espacial, borrando la tabla que contiene los *MBRs* de los objetos y también los disparadores que mantenían sincronizada la tabla de datos geométricos con la tabla índice.

Esta función admite como parámetros la conexión con la base de datos, el nombre de la tabla y el nombre del campo que contienen los datos geométricos. Devuelve un valor *booleano* indicando si la operación se ha ejecutado correctamente.

```
public static boolean st_dropSpIndex(Connection conn, String tableName, String field) throws Exception {

        String tb = tableName;
        String fd = field;
        String idx = "SpatialDataMBRTable_" + tb + "_" + fd;

        // Carga el contenido de la tabla que contiene los MBRs en memoria en la estructura del ArbolR.
        return CreateSpatialIndexH2.dropSpatialIndex(conn, tb, fd);

}
```

6. Plan de pruebas y evaluación

El Plan de pruebas que se ha seguido se compone de:

- *Pruebas unitarias de cada método, clase y componente:* Se han realizado en cada momento las pruebas necesarias para comprobar con bastante seguridad su funcionamiento unitario.
- *Pruebas de integración:* Se han realizado pruebas para comprobar la integración de todos los componentes que forman parte de la aplicación.
- Pruebas de evaluación: Se han realizado *tests* para comparar la eficiencia de la consultas espaciales con y sin índice espacial.

A continuación, los apartados siguientes reflejarán el resultado de un conjunto de pruebas de evaluación.

En la imágenes siguientes (véase las figuras 11, 12 y 13) se puede ver una representación de los datos espaciales con los que realizaremos las pruebas, concretamente con la tabla *SOILS, RIVERS Y PAISES.*

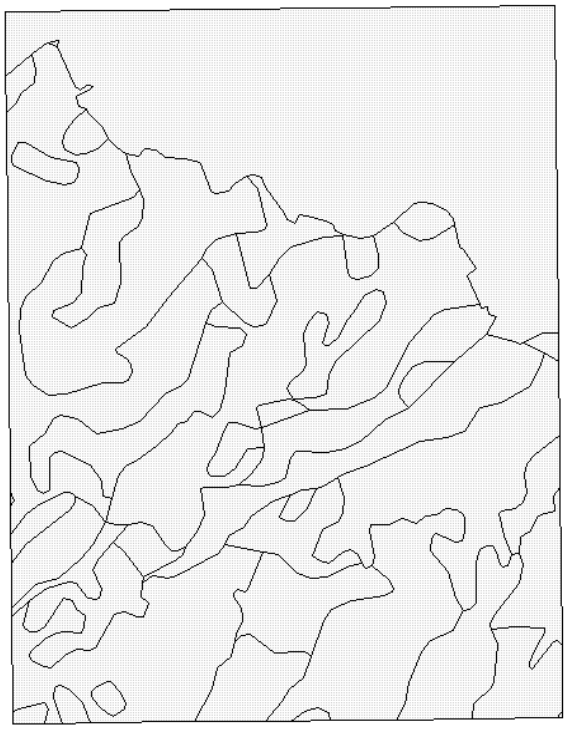

Figura 12: Representación de la tabla SOILS que contiene datos de prueba multipolíginonos

Figura 11: Representación de la tabla RIVERS que contiene datos de prueba multilíneas

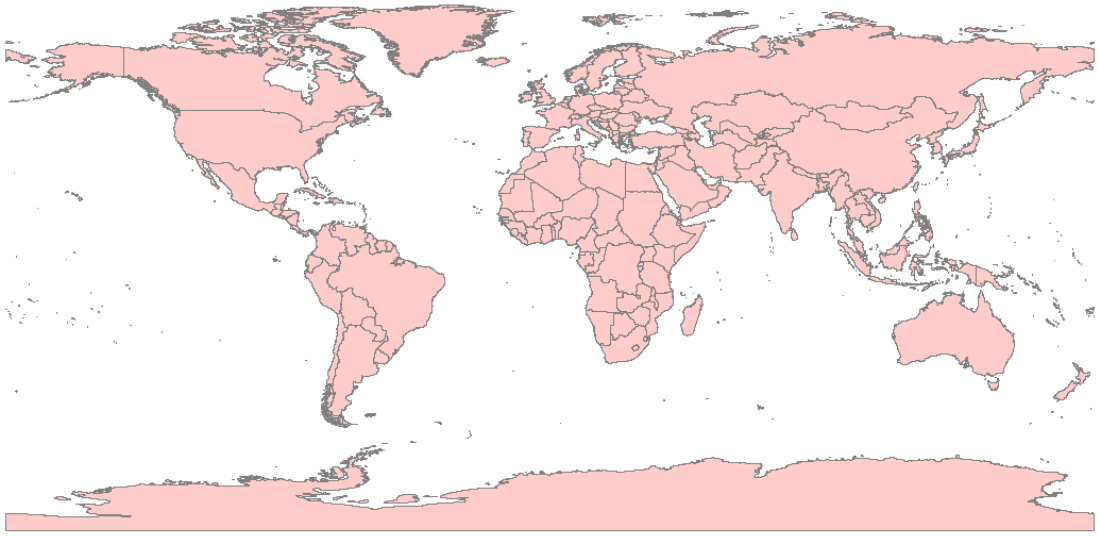

Figura 13: Representación de la capa PAISES que se usará para la demostración de la eficiencia de la aplicación

Después de la creación de la base de datos y posterior carga de datos, es importante ejecutar la siguiente orden que permitirá crear un seudónimo amigable a la función encargada de crear un índice. Esta orden sólo será necesario ejecutarla una vez pero antes de la creación de un índice espacial.

```
CREATE ALIAS IF NOT EXISTS JASPA.ST_CREATE_SPIDX FOR
"Function.st_createSpIndex"
```

En los subapartados siguientes se describirán los pasos a realizar para la creación y activación de un índice espacial. Posteriormente se usarán éstos índices para la realización de consultas espaciales y también se mostrará cómo se elimina un índice.

A continuación se realizará una consulta espacial para demostrar la eficiencia de los índices creados.

Y por último se mostrará cómo se puede crear un índice espacial desde la propia consola del sistema operativo.

6.1 Creación en H2 de un índice espacial

En el ejemplo siguiente se creará una tabla índice para el campo geométrico *GEOM* de la tabla *SOILS*:

```
SELECT ST_CREATE_SPIDX('SOILS','GEOM')
```

En la imagen siguiente (véase la figura 14) se puede comprobar cómo desde la consola de *H2* se ha creado una nueva tabla que se llama *SPATIALDATAMBRTABLE_SOILS_GEOM* y que contiene los MBRs de los datos.

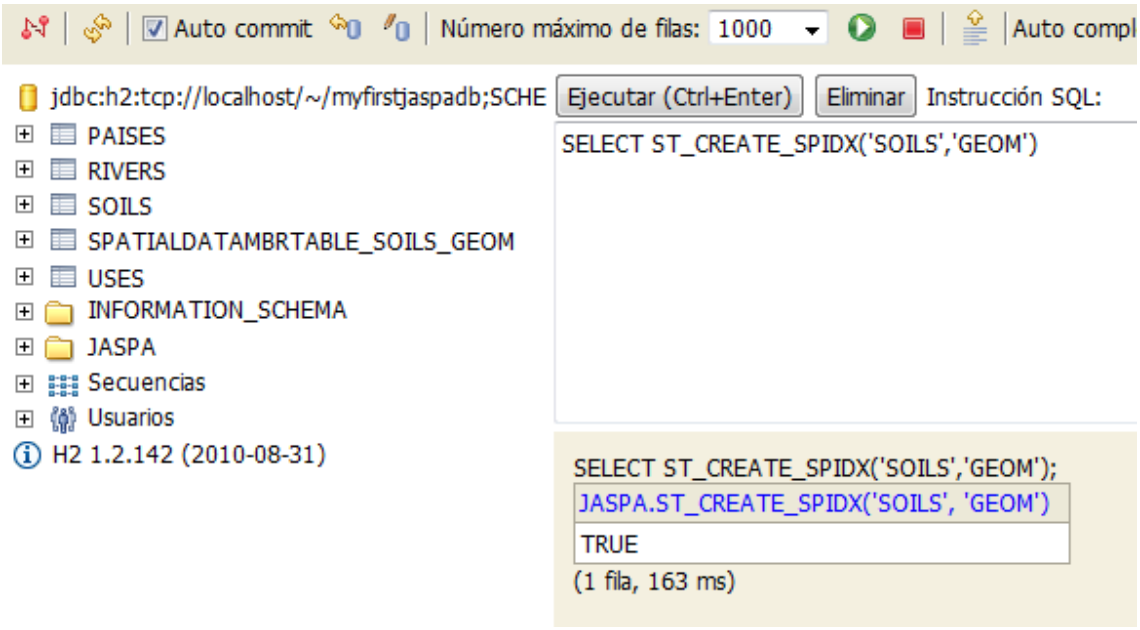

Figura 14: Creación de un índice espacial desde la consola de H2

6.2 Activación en H2 del índice espacial

Una vez que se dispone de la tabla índice, se procederá a cargar el *Árbol-R* en memoria con los datos del fichero índice, eso se realizará desde la consola de *H2* con la instrucción siguiente:

```
SELECT ST_LOAD_SPIDX('SOILS','GEOM')
```

La función que activará el índice espacial necesitará como argumentos el nombre de la tabla de datos y el nombre del campo geométrico que se indexó.

Es importante tener en cuenta que el nombre de la tabla que se pasa como primer argumento es el nombre de la tabla que contiene los datos y no la tabla que contiene el índice.

En la imagen siguiente (véase la figura 15) se puede comprobar el resultado de dicha operación. Si la consola nos devuelve *TRUE* significará que todo ha ido bien.

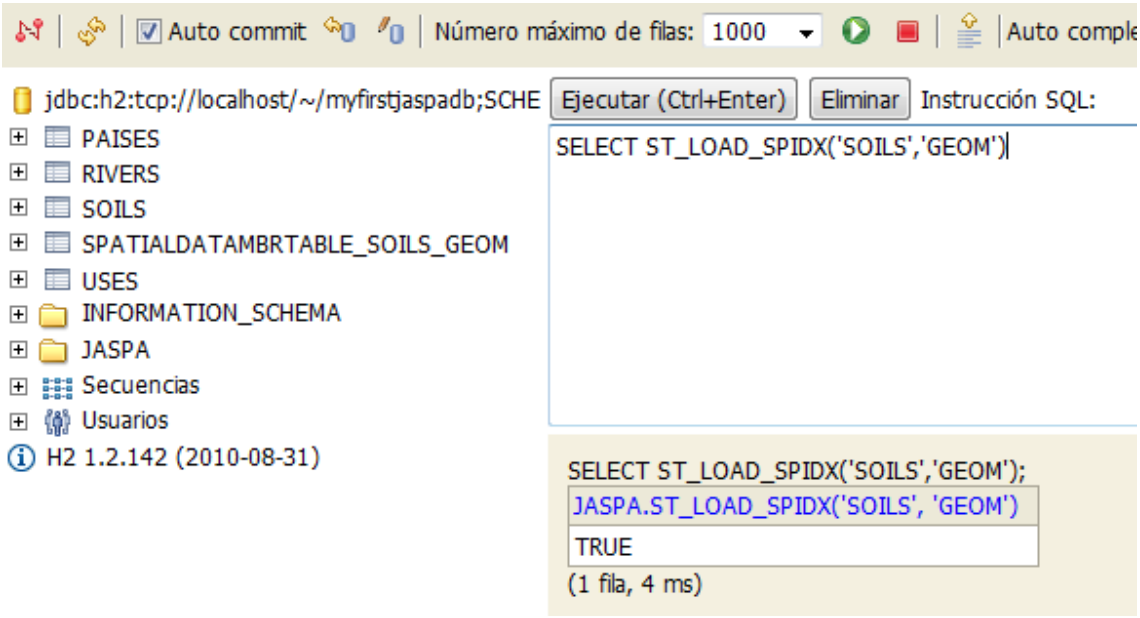

Figura 15: Activación de un índice espacial desde la consola de H2

6.3 Realización de consultas espaciales haciendo uso del índice espacial

Una vez que se ha activado el índice espacial se está en disposición de realizar consultas espaciales desde la consola de *H2*.

El formato de la función que permite realizar consultas espaciales de ventana es el siguiente:

```
ST_CONTAIN_IDX('nombredelatabladedatos','nombredelcampoindexado',XMIN,YMIN,XMA
X,YMAX))
```

Donde:

- *nombredelatabladedatos:* es el nombre de la tabla de datos que se desea consultar espacialmente.

- *nombredelcampoindexado:* es el nombre del campo sobre el que se desea consultar espacialmente.

- *XMIN*,*YMIN:* coordenadas *x* e *y* del vértice inferior izquierdo del rectángulo sobre el que se desea localizar los datos geométricos que indique la operación espacial.

- *XMAX,YMAX:* coordenadas *x* e *y* del vértice superior derecho del rectángulo sobre el que se desea localizar los datos geométricos que indique la operación espacial.

ST_CONTAIN_IDX devuelve los identificadores de los datos geométricos que se encuentran dentro del rectángulo delimitados por los pares de coordenadas que definen el vértice inferior izquierdo (*XMIN*,*YMIN)* y el vértice superior derecho *(XMAX*,*YMAX)* que se pasan como argumentos a la función *ST_USESPINDEX*.

Esta primera consulta espacial de ventana se realizará sobre los datos multipolígonos que contiene la tabla *SOILS*.

A continuación, se muestra un ejemplo de este tipo de consulta espacial que hace uso del índice espacial activado anteriormente:

```
SELECT ST_CONTAIN_SPIDX('SOILS','GEOM',1,1,9000,9000)
```

Esta orden devuelve los identificadores de aquellos datos geométricos de la columna

GEOM perteneciente a la tabla *SOILS,* que estén contenidos dentro del rectángulo definido por *XMIN,YMIN (1,1)* y *XMAX,YMAX (9000,9000)*.

En la imagen siguiente (véase la figura 16) se muestra el resultado de la ejecución de la orden anterior:

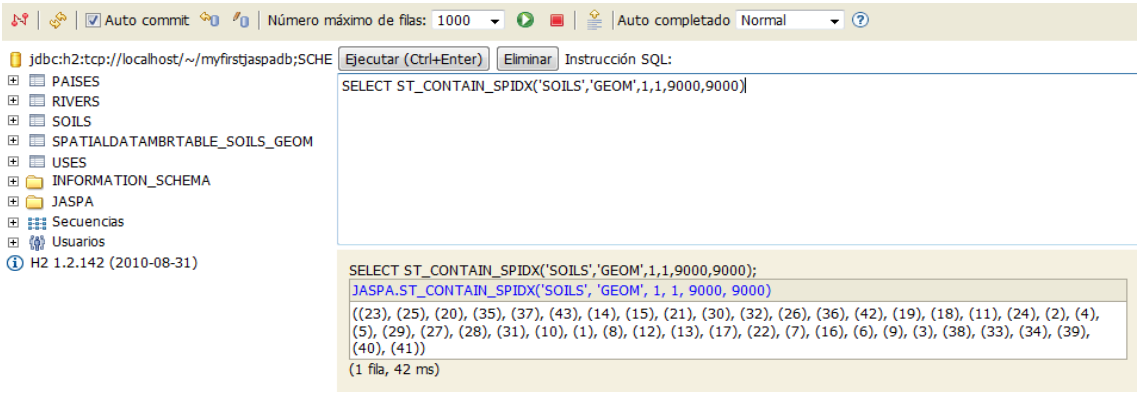

Figura 16: Realización de una consulta espacial de ventana

Se puede comprobar cómo devuelve los identificadores de todos los registros ya que el rectángulo dado contiene a todos los *MBRs* de cada dato geométrico.

A continuación procedemos a realizar una nueva consulta con un rectángulo más pequeño para obtener menos resultados:

```
SELECT ST_CONTAIN_SPIDX('SOILS','GEOM',4034,4995,5000,6000)
```

Se puede comprobar en la imagen siguiente (véase la figura 17) cómo ahora solo devuelve los identificadores de tan sólo tres registros.

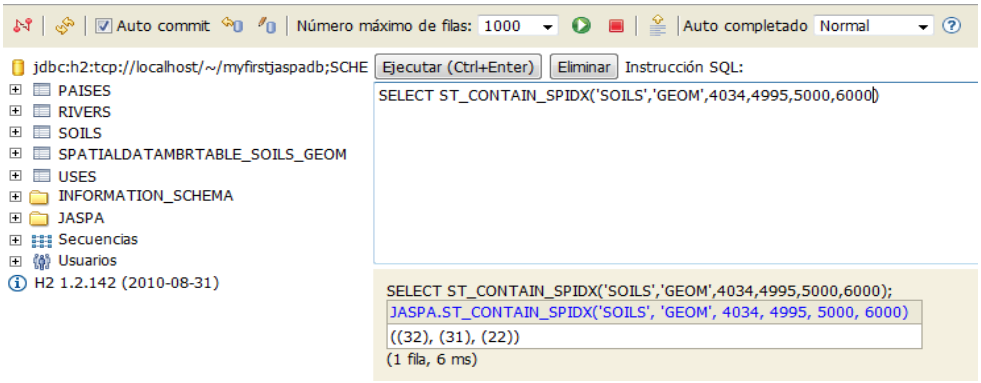

Figura 17: Realización de una consulta espacial de ventana

Ahora procedemos a modificar la orden anterior para que devuelva los registros en formato tabla en lugar de entre paréntesis y comas como se hacía anteriormente.

Para ellos generamos una nueva sentencia *SQL* donde la tabla que se quiere consultar se genera de forma dinámica partir del resultado de la ejecución de la consulta espacial.

Para ellos introducimos la siguiente orden:

```
SELECT ID FROM ST_CONTAIN_SPIDX('SOILS','GEOM',4034,4995,5000,6000)
```

En la imagen siguiente (véase la figura 18) se puede comprobar cómo el formato en el que se devuelven los resultados ha pasado a formato tabular.

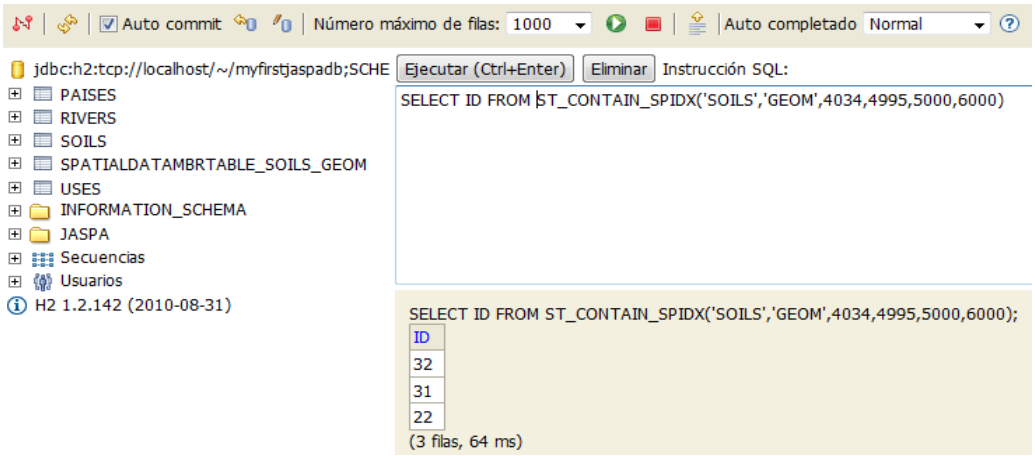

Figura 18: Realización de una consulta espacial que devuelve un conjunto de registros en formato tabular

Hasta ahora sólo hemos obtenido los identificadores de los registros, para obtener el resto de campos asociados a los identificadores se podría introducir una nueva orden como la que se muestra a continuación:

```
SELECT GID, SOIL_CODE, ST_ASTEXT(GEOM) FROM SOILS WHERE GID IN (SELECT ID FROM
ST_CONTAIN_SPIDX('SOILS','GEOM',4034,4995,5000,6000))
```

En la imagen siguiente (véase la figura 19) se puede comprobar el resultado de la ejecución del comando anterior y cómo ahora se puede ver la información asociada a cada identificador de registro. También se puede comprobar cómo cumplen perfectamente con el criterio de búsqueda fijado.

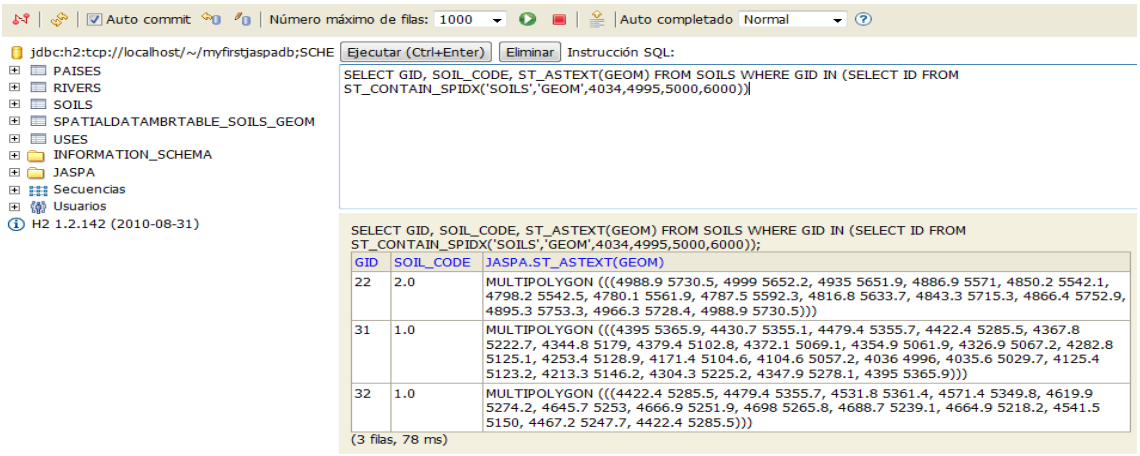

Figura 19: Realización de una consulta espacial con claúsula SELECT IN

Por último, introducimos las coordenadas de un rectángulo con unas dimensiones muy pequeñas para seguir comprobando el funcionamiento del índice. En este caso no debe de devolver ningún resultado tal y cómo se muestra en la imagen siguiente (véase la figura 20):

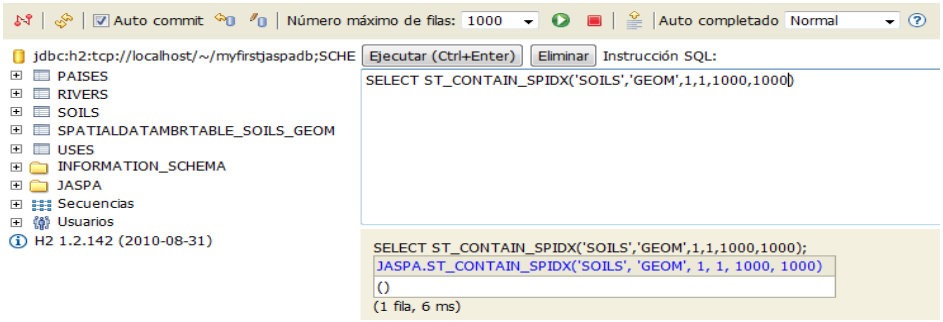

Figura 20: Realización de consulta espacial que no devuelve ningún resultado

Por último, se realizará una consulta espacial de tipo INTERSECT sobre la los datos *multilíneas* que contiene la tabla *RIVERS*. Dado un rectángulo el sistema devolverá aquellos objetos que estén parcial o totalmente incluidos dentro de él.

El formato de la orden que permite realizar consultas espaciales de ventana es el siguiente:

```
ST_INTERSECT_SPIDX('nombredelatabladedatos','nombredelcampoindexado',XMIN,YMIN
,XMAX,YMAX))
```

Donde:

- *nombredelatabladedatos:* es el nombre de la tabla de datos que se desea consultar espacialmente.

- *nombredelcampoindexado:* es el nombre del campo sobre el que se desea consultar espacialmente.

- *XMIN*,*YMIN:* coordenadas *x* e *y* del vértice inferior izquierdo del rectángulo sobre el que se desea localizar los datos geométricos que indique la operación espacial.

- *XMAX,YMAX:* coordenadas *x* e *y* del vértice superior derecho del rectángulo sobre el que se desea localizar los datos geométricos que indique la operación espacial.

Antes de realizar la consulta sobre la tabla *RIVERS* hay que crear el índice y activarlo de la misma forma que se ha hecho anteriormente con la tabla *SOILS*.

A continuación, se muestra un ejemplo de una consulta espacial que hace uso del índice espacial activado anteriormente:

```
SELECT ST_INTERSECT_SPIDX('RIVERS','GEOM',5700,4600,6200,6000);
```

Esta orden devuelve los identificadores de aquellos datos geométricos de la columna *GEOM* perteneciente a la tabla *RIVERS,* que estén contenidos total o parcialmente dentro del rectángulo definido por *XMIN,YMIN (5700,4600)* y *XMAX,YMAX (6200,6000)*.

En la imagen siguiente (véase la figura 21) se puede comprobar el resultado de la

ejecución de la orden anterior:

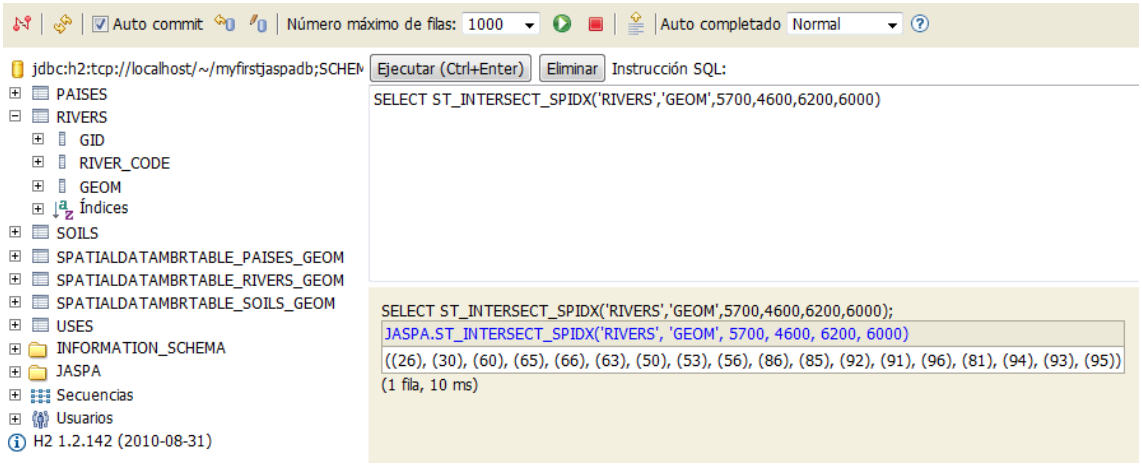

Figura 21: Ejecución de una consulta espacial de Intersección

6.4 Eliminación de un índice espacial

El formato de la orden que permite realizar eliminar un índice espacial es el siguiente:

```
ST_DROP_SPIDX('nombredelatabladedatos','nombredelcampoindexado')
```

La orden siguiente eliminaría el índice creado anteriormente, para ello eliminará la tabla con los MBRs, los alias y los disparadores.

```
SELECT ST_DROP_SPIDX('SOILS','GEOM');
```

En la imagen siguiente (véase la figura 22) se puede comprobar el resultado de la ejecución de la orden anterior:

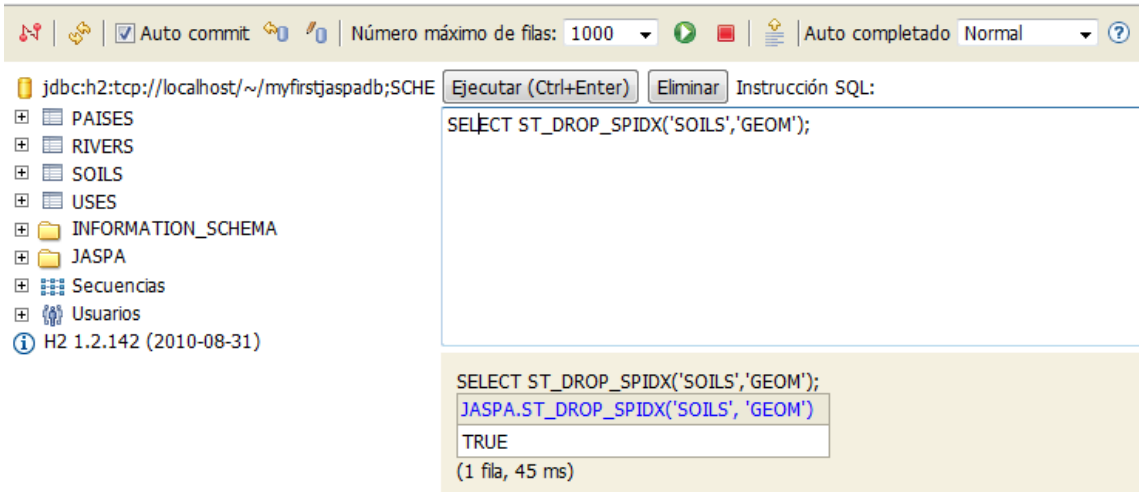

Figura 22: Eliminación de un índice espacial desde la consola de H2

Se puede comprobar cómo la tabla llamada *SPATIALDATAMBRTABLE_SOILS_GEOM* que contenía el índice ha desaparecido al haber sido eliminada.

6.5 Comparativa entre una consulta espacial que usa un índice espacial y una equivalente que no lo usa

En este apartado se tratará de demostrar la eficiencia de la aplicación. Para ello se realizará una misma consulta espacial de dos formas diferentes: la primera de ellas se realizará sin hacer uso del índice y la segunda haciendo uso del índice.

Teniendo en cuenta que el propio gestor H2 dispone de su propia *caché*, para evitar que que haga uso de ella en las consultas que se van a realizar se desactiva a través de la orden siguiente:

```
SET CACHE_SIZE 0
```

Para esta demostración se usará una nueva capa llamada PAISES de tamaño considerablemente mayor que las usadas anteriormente, en concreto 16 *Megabytes*.

La primera consulta a realizar se muestra a continuación y obtiene los identificadores incluidos dentro de un rectángulo dado, en concreto, el definido por el vértice inferior izquierdo (1,1) y el vértice superior derecho (9000,9000).

```
SELECT GID FROM PAISES where st_xmin(GEOM) >1 and st_ymin(GEOM) > 1 and
st_xmax(GEOM)<9000 and st_ymax(GEOM)<9000
```

En la imagen siguiente (véase la figura 23) se puede observar el tiempo que lleva realizar la consulta.

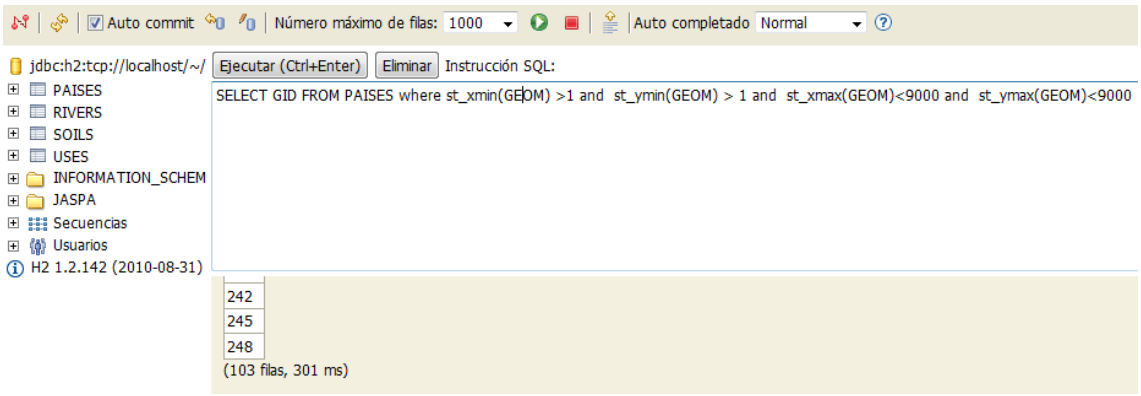

Figura 23: Ejemplo de realización de una consulta espacial sin hacer uso de un índice espacial

Es necesario recordar que antes de la ejecución de esta consulta es necesario crear el índice espacial, en este caso se ha omitido este paso ya que se ha descrito con detalle anteriormente.

A continuación se muestra la sentencia equivalente a la consulta anterior pero esta vez haciendo uso del índice espacial.

```
SELECT ID FROM ST_CONTAIN_SPIDX('PAISES','GEOM',1,1,9000,9000)
```

En la imagen siguiente (véase la figura 24) se puede comprobar el resultado de la ejecución de la orden anterior:

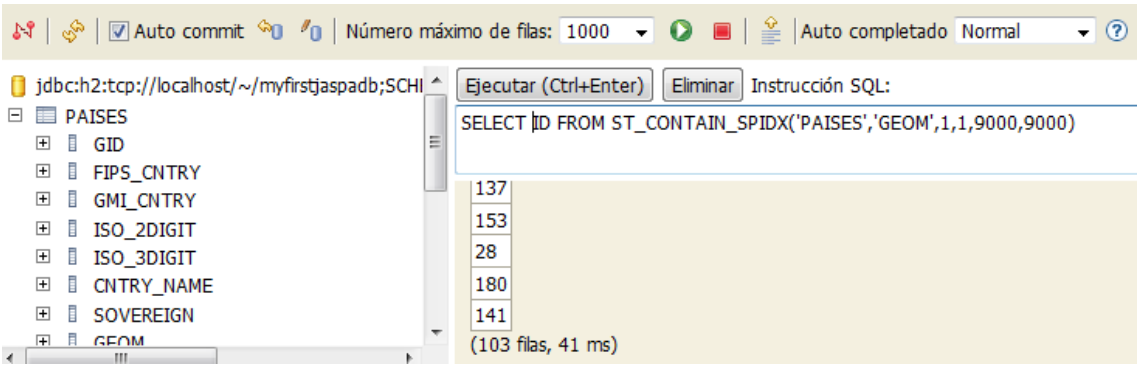

Figura 24: Ejemplo de realización de una consulta espacial haciendo uso de un índice espacial

Se puede comprobar fácilmente como la primera consulta consume *301 ms* y la segunda consulta consume *41 ms*. Hay que tener en cuenta que en la segunda consulta se ha despreciado el tiempo que lleva crear la tabla que contiene los *MBRs* y el tiempo que se tarda en regenerar el *Arbol-R* en memoria. Ambas operaciones se harán una única vez y servirán para poder realizar un número indefinido de consultas sobre el índice activo.

Tras repetir las mismas consultas en diferentes sesiones, se observa como los resultados son similares y la diferencia de tiempo entre una consulta y otra son del mismo orden.

En la tabla 4 se puede comprobar los resultados de la ejecución de las dos consultas anteriores en cinco sesiones diferentes:

	Tiempo consumido por la consulta 1 (sin usar índice espacial)	*Tiempo consumido por la consulta 2 (usando índice espacial)*
Sesión 1	*301ms*	*41ms*
Sesión 2	*456ms*	*45ms*
Sesión 3	*465ms*	*54ms*
Sesión 4	*442ms*	*41ms*
Sesión 5	*443ms*	*40ms*

Tabla 4: Comparativa de tiempos de respuesta de una consulta espacial con y sin índice espacial.

6.6 Creación del índice espacial desde el sistema operativo

Hasta el momento se ha visto como la consola del gestor *H2* es el lugar adecuado tanto para la creación de índices espaciales como para la realización de consultas espaciales.

Resulta interesante mencionar que la aplicación se ha implementado de forma que también permite la creación de un índice espacial sobre *H2*, desde el propio sistema operativo, sin necesidad de entrar en la consola del gestor de base de datos. Esto facilita por ejemplo, la creación masiva de índices espaciales que en algunos casos y dependiendo de las dimensiones de las tablas de datos, puede llevar grandes intervalos de tiempo resultando interesante conocer en todo momento el estado del proceso.

Para ejecutar la aplicación, se necesita estar situado en el directorio *class* y ejecutar la aplicación *CreateSpatialIndexH2*. A continuación se indican los argumentos que permite:

--b: Indica el nombre de una base de datos creada previamente y en la que se desea crear el índice espacial.

--u: Indica el nombre de usuario que tiene los privilegios adecuados para acceder a la base de datos que le sigue al argumento anterior --*b.*

--p: Indica la contraseña asociada al usuario que le sigue al argumento anterior --*u*.

--t: Indica el nombre de la tabla que se desea indexar.

--c: Indica el nombre del campo que se desea indexar y que pertenece a la tabla que le sigue al argumento anterior -*t.*

Para la ejecución de la aplicación se necesitan las siguientes librerías: *h2-1.2.139.jar, log4j.jar; trove.jar* y *jsi-1.0b6.jar***.** Para evitar problemas derivados de la no localización de alguna de las librerías necesarias a causa de la para la ejecución de la aplicación desde la propia consola del sistema operativo, éstas se pueden pasar como argumentos tal y cómo se explica en el manual de usuario (*apartado 8*).

Para ejecutar la aplicación Cambiar al directorio ..\class y ejecutar el comando

siguiente, tal y como se muestra en la imagen:

```
java CreateSpatialIndexH2 nombre_tabla nombre_columna_geometrica
```

Para facilitar aun más las pruebas de creación y uso de índices espaciales desde el propio sistema operativo, se pueden omitir los tres primeros argumentos descritos anteriormente *–b*, *--u* y *--p*. En este caso la aplicación tomará por defecto la base de datos, el nombre de usuario y contraseña adecuada para acceder a la base de datos y a las tablas creadas en el apartado anterior, es decir, tomará los argumentos: *-b:myfirstjaspadb -u:sa -p:123*.

A continuación se muestran los dos posibles formatos que puede tener la instrucción de creación del índice espacial.

En primer lugar se muestra el formato largo de la orden que incluye todos los argumentos posibles:

```
java CreateSpatialIndexH2 --b:nombre_base_datos  --u:usuario_base_datos
--p:constraseña_base_datos --t:nombre_tabla --c:nombre_columna_geometrica
```

A continuación, se muestra el formato breve de la orden en que sólo se pasará como argumento el nombre de la tabla y el nombre del campo que contiene los datos geométricos:

```
java CreateSpatialIndexH2 --t:nombre_tabla --c:nombre_columna_geometrica
```

En el ejemplo siguiente se creará una tabla índice para el campo geométrico *GEOM* de la tabla *SOILS*:

```
java CreateSpatialIndexH2 SOILS GEOM
```

En la imagen siguiente (véase la figura 25) de puede comprobar cómo se crearía el índice espacial desde la línea de comandos:

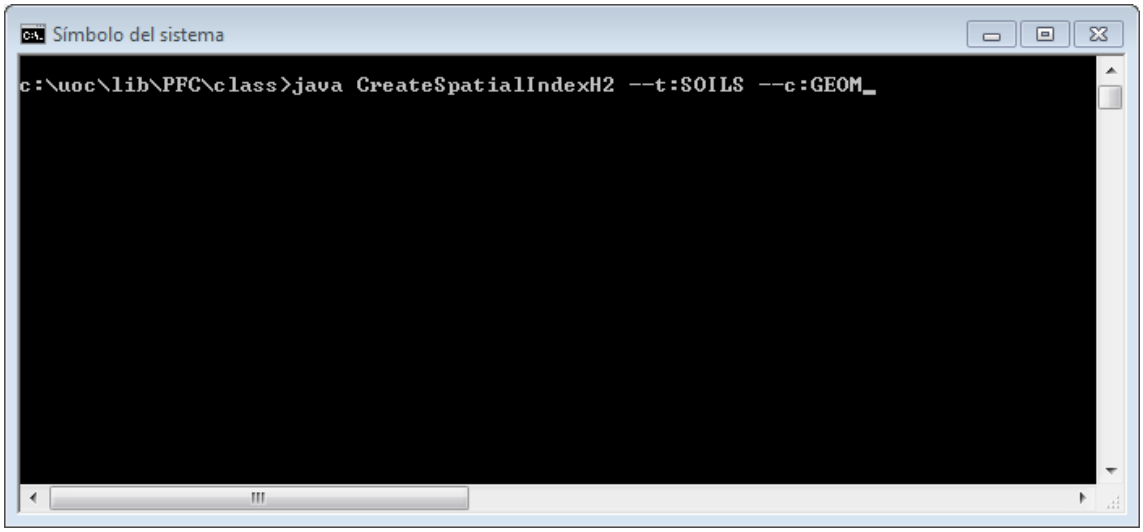

Figura 25: Creación del índice espacial desde la línea de comandos

Si se decide realizar la creación de un índice desde la consola del sistema operativo habrá que tener especial cuidado de incluir en el *CLASSPATH* la ruta donde se incluyen las librerías externas necesarias para la correcta ejecución del programa, o en su defecto pasarlas con el argumento:

-cp .;..\lib\h2-1.2.139.jar;..\lib\log4j.jar;..\lib\trove.jar;..\lib\jsi-1.0b6.jar

El resultado de esta operación confirmará que se ha creado correctamente la tabla índice, los *triggers* y los *alias* para las funciones:

A continuación (véase la figura 26) se puede verificar el resultado de la creación del índice, una tabla llamada *SPATIALDATAMBRTABLE_SOILS_GEOM* con la estructura de datos correcta y con las coordenadas del vértice inferior izquierdo y las del vértice superior derecho del MBR de cada registro de la tabla *SOILS*.

SELECT * FROM SPATIALINDEXTABLE_SOILS_GEOM;

SGID	XMIN	YMIN	XMAX	YMAX
1	4009.0	6054.8	6124.3	7377.0
2	4173.1	7217.1	4214.0	7240.0
3	4013.5	5314.9	4838.1	7228.5
4	4011.9	6957.3	4125.2	7188.3
5	4230.2	6796.3	4404.7	6975.0
6	4034.7	6679.6	4294.0	6845.1
7	4062.9	5859.8	5012.1	6817.8
8	4894.1	6273.1	5301.9	6718.7
9	4188.9	6119.9	4538.1	6656.3
10	5495.3	6461.4	5728.2	6611.9
11	4395.0	5251.9	5824.4	6530.3
12	4762.9	6124.4	5048.8	6486.4
13	5301.6	6311.1	5439.0	6484.5
14	5086.3	5844.0	5468.3	6269.8
15	4901.1	5504.1	5885.5	6212.3
16	4101.6	5456.3	4934.4	6141.4
17	5988.9	5989.5	6125.7	6102.7
18	4535.6	5126.6	6071.9	6094.7
19	5184.2	5021.3	6129.0	6021.9
20	5514.8	5807.6	5730.1	5989.5
21	4969.2	5730.5	5175.4	5841.5
22	4780.1	5542.1	4999.0	5753.3
23	5899.0	5237.8	6131.4	5703.5
24	4848.6	5146.7	5372.8	5535.9
25	5862.3	4792.3	6138.6	5500.7
26	5060.6	5367.8	5198.3	5496.2
27	4030.4	5426.1	4043.5	5485.5
28	4032.4	5205.4	4277.1	5481.8
29	4033.6	5029.7	4395.0	5465.4
30	5007.0	4588.4	5766.8	5416.5
31	4035.6	4996.0	4479.4	5365.9
32	4422.4	5150.0	4698.0	5361.4
33	4078.3	4797.1	4556.7	5285.5
34	4036.0	4578.5	4993.4	5277.8
35	5506.4	4594.2	5988.0	5273.6
36	4884.6	4837.6	5492.8	5245.4
37	5797.0	4597.5	6141.8	4955.9
38	4733.4	4585.2	5178.4	4926.9
39	4342.6	4608.4	4471.4	4744.5
40	4194.9	4579.4	4334.9	4730.2
41	4040.2	4577.1	4176.1	4660.0
42	5102.6	4589.5	5253.6	4641.3
43	6121.8	4601.3	6142.0	4621.4

(43 filas, 13 ms)

Figura 26: Contenido de la tabla índice creada a partir de la tabla SOILS

A partir de este momento, se podría ejecutar de la misma forma cualquiera de las funciones relacionadas con el índice espacial descritas anteriormente ya siempre desde la consola del gestor *H2*.

7. Conclusiones

Se puede afirmar que se ha logrado con éxito el desarrollo de un índice espacial para la extensión *JASPA* sobre el gestor de base de datos *H2*. A continuación, se indican sus funcionalidades y algunas posibles mejoras.

Para la realización del proyecto se ha necesitado estudiar en profundidad:

- El SGBD H2.
- La extensión JASPA.
- La creación de índices espaciales, incluyendo la estructuras de datos necesarias (tabla índice, Árbol-R).

También fue necesario:

- Crear los componentes que implementan las funciones previstas.
- Integrar todos los componentes anteriores, tanto los externos cómo los nuevos, estudiando y creando las interfaces necesarias para ello.
- Implementación de los disparadores para que la tabla del índice permanezca siempre sincronizada con la tabla que contiene los datos.

Se ha implementado una solución que permite crear, activar y eliminar índices espaciales con una forma de uso similar a las funciones espaciales proporcionadas por la extensión *JASPA*.

La aplicación permite realizar dos tipos de consultas espaciales que son:

- De contenido: dado un rectángulo devuelve los objetos incluidos completamente dentro de él.
- De intersección: dado un rectángulo devuelve aquellos objetos incluidos parcial o completamente dentro de él.

Para ello se ha necesitado crear una tabla que almacene los *MBRs* de los datos geométricos y se ha precisado de la implementación de unos *triggers* para que dicha tabla permanezca siempre sincronizada con la tabla que contiene los datos.

Una característica importante de esta solución es que para su implementación no ha sido necesario modificar ni el código fuente de *JASPA*, ni de *H2* lo que la hubiera hecho totalmente dependiente de ambas tecnologías y también menos flexible.

Desde el punto de vista de la usabilidad de la aplicación y considerando el contexto de la implementación enmarcado entre diferentes componentes externos, se ha tratado de que las funciones creadas para el usuario final sigan el mismo patrón de uso que las ya existentes en la extensión *JASPA*.

Por último, mencionar la posibilidad que incorpora la aplicación para crear uno o varios índices espaciales desde la consola del sistema operativo, sin tener que acceder obligatoriamente a la consola de H2 y con la posibilidad de hacer un seguimiento al proceso de creación de los índices desde la propia consola a través de los mensajes de *logs* emitidos.

7.1 Recomendaciones para futuros desarrollos

Durante la realización del proyecto han surgido nuevas ideas y opciones que añadir, además de mejorar las realizadas. Algunas de ellas se indican a continuación:

- Adaptar la solución propuesta para que trabaje con datos en tres dimensiones.

- Aunque el programa permite tener varias columnas indexadas espacialmente por tabla, sólo se puede tener un índice activo, sería interesante poder tener más de un índice activo simultáneamente.

- Se han realizado únicamente dos tipos de consulta, resultaría relativamente fácil añadir nuevos tipos de consultas por ejemplo obtener lo objetos más cercanos a una distancia determinada de punto dado.

- El algoritmo *Árbol-R* está implementado para trabajar en memoria RAM, esto supondría una limitación física en el caso de que se trabaje con varios índices espaciales simultáneamente o con bases de datos muy grandes. Una mejora sería

plantear una alternativa que supliera esta carencia.

- Adaptación de esta solución al resto de bases de datos compatibles con *JASPA*.

8. Bibliografía

8.1 Por órden de aparición

[1] Universitat de Washington. Portal sobre SIG y tecnologías de información espacial. What is a Geographic Information System (GIS)? (NCGIA lecture by David Cowen, 1989) Disponible en: http://gis.washington.edu/phurvitz/professional/SSI/whatis.html (Consultado el 30 de Septiembre de 2010)

[2] Pérez A., Botella A., Muñoz A., Olivella R., Olmedillas J.C., Rodríguez J. (2009) Sistemas de Información Geográfica y GeoTelemática.. Universitat Oberta de Cataluña.

[3] Domínguez J. (2000)
Breve introducción a la cartografía y a los Sistemas de Información Geográfica.[en línea]
Disponible en:
http://www.preval.info/programa/wp-content/uploads/2008/09/itc_cartografia_sig.pdf
(Consultado el 1 de Octubre de 2010)

[4] Vélez J.F. (1999)
Indexacion directa de información temporal para el Postgres IDITPos
Disponible en: http://jaibana.udea.edu.co/grupos/revista/revistas/nro018/indexacion.htm
(Consultado el 2 de Octubre de 2010)

[5] Autor no indicado en la referencia. OGC® Standards and Specifications [en línea]
Disponible en: http://www.opengeospatial.org/standards/
(Consultado el 1 de Noviembre de 2010)

[6] Besembel I. y Roberts S. (1998)
Comparación entre métodos de acceso espaciales y de puntos multidimensionales. [en línea]
Disponible en: http://www.ing.ula.ve/~ibc/maemp.pdf
(Consultado el 14 de Noviembre de 2010)

[7] Autor anónimo. Wikipedia, la enciclopedia libre
Quadtree [en línea]
Disponible en: http://es.wikipedia.org/wiki/Quadtree
(Consultado el 2 de Octubre de 2010)

[8] Autor anónimo. Wikipedia, la enciclopedia libre
Árbol-R [en línea]
Disponible en: http://es.wikipedia.org/wiki/Árbol-R
(Consultado el 2 de Octubre de 2010)

[9] Moore A. (2001). Carnegie Mellon University
An intoductory tutorial on kd-trees [en línea]
Disponible en:
http://www.ri.cmu.edu/pub_files/pub1/moore_andrew_1991_1/moore_andrew_1991_1.pdf
(Consultado el 16 de Noviembre de 2010)

[10] Brisaboa N., Luaces M., Navarro G., Seco D. (2006)
Indexación espacial de puntos empleando wavelet trees. [en línea]
Disponible en: http://www.dcc.uchile.cl/~gnavarro/ps/jisbd09.pdf
(Consultado el 17 de Noviembre de 2010)

[12] Rigaux P., Scholl M. y Voisard A.(2002)
Spatial Databases - with Application to GIS [Capítulos 5, 6 y 8]. Ed. Morgan Kaufmann Publishers.

[13] Guttman A. (1984). University of California.
R-Trees a dinamice index structure for spatial searching. [en línea]
Disponible en: http://postgis.refractions.net/support/rtree.pdf
(Consultado el 5 de Octubre de 2010)

Cuenca M.J., Nicolau F., Campos A., Ribera J., Marco M.J., Segret R. (2010)
Competencia comunicativa para profesionales de las TIC. Universitat Oberta de Cataluña.

Lamarca I., Rodríguez J.R., Codolá S. (2007)
Metodología de gestión de proyectos TIC. Universitat Oberta de Cataluña.

8.2 Por órden alfabético

Besembel I. y Roberts S. (1998)
Comparación entre métodos de acceso espaciales y de puntos multidimensionales. [en línea]
Disponible en: http://www.ing.ula.ve/~ibc/maemp.pdf
(Consultado el 14 de Noviembre de 2010)

Brisaboa N., Luaces M., Navarro G., Seco D. (2006)
Indexación espacial de puntos empleando wavelet trees. [en línea]
Disponible en: http://www.dcc.uchile.cl/~gnavarro/ps/jisbd09.pdf
(Consultado el 17 de Noviembre de 2010)

[3] Cuenca M.J., Nicolau F., Campos A., Ribera J., Marco M.J., Segret R. (2010)
Competencia comunicativa para profesionales de las TIC. Universitat Oberta de Cataluña.

Domínguez J. (2000)
Breve introducción a la cartografía y a los Sistemas de Información Geográfica.[en línea]
Disponible en:
http://www.preval.info/programa/wp-content/uploads/2008/09/itc_cartografia_sig.pdf
(Consultado el 1 de Octubre de 2010)

Guttman A. (1984). University of California.
R-Trees a dinamice index structure for spatial searching. [en línea]
Disponible en: http://postgis.refractions.net/support/rtree.pdf
(Consultado el 5 de Octubre de 2010)

[6] Lamarca I., Rodríguez J.R., Codolá S. (2007)
Metodología de gestión de proyectos TIC. Universitat Oberta de Cataluña.

Moore A. (2001). Carnegie Mellon University
An intoductory tutorial on kd-trees [en línea]
Disponible en:
http://www.ri.cmu.edu/pub_files/pub1/moore_andrew_1991_1/moore_andrew_1991_1.pdf
(Consultado el 16 de Noviembre de 2010)

Pérez A., Botella A., Muñoz A., Olivella R., Olmedillas J.C., Rodríguez J. (2009)
Sistemas de Información Geográfica y GeoTelemática.. Universitat Oberta de Cataluña.

Rigaux P., Scholl M. y Voisard A.(2002)
Spatial Databases - with Application to GIS [Capítulos 5, 6 y 8]. Ed. Morgan Kaufmann

Publishers.

Vélez J.F. (1999)
Indexacion directa de información temporal para el Postgres IDITPos
Disponible en: http://jaibana.udea.edu.co/grupos/revista/revistas/nro018/indexacion.htm
(Consultado el 2 de Octubre de 2010)

Universitat de Washington. Portal sobre SIG y tecnologías de información espacial.
What is a Geographic Information System (GIS)? (NCGIA lecture by David Cowen, 1989)
Disponible en: http://gis.washington.edu/phurvitz/professional/SSI/whatis.html
(Consultado el 30 de Septiembre de 2010)

Autor anónimo. Wikipedia, la enciclopedia libre
Árbol-R [en línea]
Disponible en: http://es.wikipedia.org/wiki/Árbol-R
(Consultado el 2 de Octubre de 2010)

Autor anónimo. Wikipedia, la enciclopedia libre
Quadtree [en línea]
Disponible en: http://es.wikipedia.org/wiki/Quadtree
(Consultado el 2 de Octubre de 2010)

Autor no indicado en la referencia. OGC® Standards and Specifications [en línea]
Disponible en: http://www.opengeospatial.org/standards/
(Consultado el 1 de Noviembre de 2010)