UNIVERSIDAD POLITECNICA DE VALENCIA

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

TRABAJO FIN DE MÁSTER

MÁSTER EN INGENIERÍA DEL SOFTWARE, MÉTODOS FORMALES Y SISTEMAS DE INFORMACIÓN

DESARROLLO DE UNA ALTERNATIVA FOSS A ORACLE SPATIAL

Autora **MARTA GONZÁLEZ ALCAIDE**

Directores **JUAN CARLOS CASAMAYOR RÓDENAS**
JOSÉ CARLOS MARTÍNEZ LLARIO

Valencia, Febrero 2012

Índice de ilustraciones

Índice de Tablas

1. INTRODUCCIÓN

1.1. Antecedentes

La organización y el almacenamiento de información en una base de datos tienen como objetivo la consulta y recuperación de información cuando sea necesario. Para obtener la información, se deben explotar las relaciones entre los datos y extraer el subconjunto que nos interese.

Las bases de datos geográficas aportan una mayor riqueza a los datos convencionales al añadir la posición espacial. Así, dos elementos pueden estar relacionados por estar cerca, ser vecinos, estar uno sobre otro o ser adyacentes. Estás relaciones no se almacenan explícitamente con una relación entre tablas, sino que se obtienen al consultar la base de datos espacial.

Durante los últimos años han aparecido numerosas iniciativas orientadas al desarrollo de aplicaciones informáticas de Sistemas de Información Geográfica bajo la filosofía del software libre.

Sin embargo en cuanto a bases de datos espaciales, no había ninguna alternativa equiparable a PostGIS (extensión espacial en C para la base de datos PostgreSQL) que ha sido durante años la única extensión espacial libre con plena funcionalidad en el manejo de datos espaciales.

En el año 2010 se presentó Jaspa, una extensión espacial para sistemas gestores de bases de datos relacionales de software libre y con código abierto escrita en Java.

Los objetivos de Jaspa eran en primer lugar, cubrir la ausencia en el mundo del software libre (Free and Open Source Software FOSS) de una alternativa sólida a PostGIS basada en Java. En segundo lugar se pretendía aprovechar las ventajas de Java y las librerías Java geoespaciales en términos de portabilidad y extensibilidad.

1.2. Estructura

En la línea de explotar las capacidades de portabilidad de Jaspa, surge este proyecto cuyo objetivo principal es portar la funcionalidad de Jaspa a la base de datos Oracle 11g y dotarla de capacidades espaciales.

Oracle ya proporciona su propia extensión espacial, Oracle Spatial. Sin embargo esta extensión debe adquirirse por separado de la base de datos y sigue un modelo de datos propio no estándar y por tanto incompatible con otros Sistemas de Información Geográfica. Jaspa puede servir en ese aspecto para dotar de capacidad espacial a Oracle por sí misma o como complemento al sistema espacial propio de Oracle.

En este primer capítulo se van a tratar temas introductorios como los estándares, estado del arte de bases de datos espaciales, datos espaciales etc.

En el segundo capítulo se presentan las características de Jaspa, la extensión espacial FOSS en la que se basa el presente proyecto. Se tratan aspectos como el almacenamiento de geometrías, funcionalidad o qué características debe soportar una base de datos convencional para poder utilizar Jaspa.

En el tercer capítulo se muestran las características espaciales de Oracle actuales, los elementos que utilizaremos para la indexación espacial, la creación de una base de datos y de un usuario con los permisos necesarios para desplegar Jaspa en Oracle.

En el cuarto capítulo se aborda como se han desarrollado los diferentes elementos de Jaspa para Oracle. En concreto se trata la importación de librerías, la definición de las tablas de metadatos, la indexación espacial y la creación de diferentes tipos de funciones: simples, agregados espaciales y Arrays de geometrías.

En el quinto capítulo se explica la utilización básica Jaspa en Oracle. Se expone cómo importar datos espaciales y como se consultan los datos. Además, se explica cómo utilizar la indexación espacial y sus beneficios.

1.3. Estándares

Hoy en día existe un aumento constante de los proyectos de software relacionados con la gestión de la información geográfica.

Cuando se diseña un nuevo software, es necesario aplicar los estándares que definen las características básicas que hacen posible su integración con los productos existentes y con los que vendrán en el futuro.

Las estándares proporcionan tres ventajas principales: portabilidad, facilidad de mantenimiento e interoperabilidad (Groot and McLaughlin, 2000). Su uso es una garantía de eficacia, ya que se ahorra tiempo al eliminar la necesidad de reinventar métodos para almacenar y manipular datos espaciales (Kralidis, 2008).

Los estándares deben ser independientes de la industria o intereses particulares. Tienen que ser desarrollados por instituciones oficiales o por consorcios ampliamente aceptados e integrados por administraciones, empresas y centros de investigación. Existen varios organismos a nivel nacional e internacional que han impulsado la normalización en la información geográfica.

La referencia internacional para los estándares de gestión de datos es el consorcio OGC (Open Geospatial Consortium), que publicó en 1999 la especificación SQL/ SFS (OpenGIS Consortium, 1999) cuya última versión actualizada es del año 2006 (OpenGIS Consortium, 2006).

El estándar Simple Features Access - SQL Option (abreviado como SQL/ SFS) estandariza las llamadas a funciones SQL para funciones de almacenamiento, recuperación, consulta y actualización de entidades geográficas, independientemente del SGBD que se esté utilizando. Además, propone un modelo de datos en el que una colección de entidades geográficas de un tipo de geometría determinado se almacena en una tabla en la que uno de sus atributos es la geometría de la entidad geográfica. Cada entidad geográfica representa un registro en esa tabla. En resumen, se estandariza como añadir funcionalidad espacial a sistemas de bases de datos.

El estándar internacional ISO 13249-3, (también denominado “SQL/MM Part 3: Spatial”) representa la evolución de la especificación OGC SFS, en el que se define como almacenar, recuperar y procesar datos espaciales utilizando SQL. Se define cómo se representan los datos, los tipos definidos por el usuario, las rutinas y los esquemas para el manejo genérico de datos espaciales. También establece las funciones disponibles para convertir, comparar y procesar los datos espaciales. (Stolze,2003).

1.4. Arquitectura de las bases de datos espaciales

Los sistemas de gestión de bases de datos relacionales (SGBDRs) se utilizan para mantener y administrar eficientemente grandes colecciones de datos. Son rasgos comunes para lograrlo las transacciones, vistas, integridad referencial, indexación etc.

Sin embargo, los SGBDRs no son compatibles por defecto con las capacidades espaciales, estando básicamente diseñados para datos alfanuméricos. Este problema es especialmente destacable en los proyectos de código abierto.

Para resolver este problema, (Vijlbrief and van Oosterom, 1992) señala tres tipos de arquitecturas de sistemas que se han utilizado para enlazar los datos espaciales con los SGBDRs.

El primer enfoque es la arquitectura dual, en la que se almacena los datos espaciales fuera del SGBDR (por ejemplo, Esri ARC/INFO o Intergraph MGE). La segunda, la arquitectura de capas, traduce los datos espaciales al modelo relacional (por ejemplo, GeoView o SIRODBMS). El tercer enfoque es la arquitectura integrada, que consiste en la ampliación del SGBDR con tipos de datos espaciales y funciones para gestionar las características geográficas (por ejemplo, TIGRIS o PostGIS).

En la arquitectura dual las bases de datos se utilizan como un mero contenedor de la información y las operaciones espaciales son realizadas por las aplicaciones externas a la base de datos. En consecuencia, esta arquitectura no se aprovecha de las propiedades ACID (Atomicity, Consistency, Isolation and Durability). Este inconveniente se resuelve en la arquitectura de capas, pero los métodos de recuperación y acceso son más complejos y los sistemas son en general menos eficientes.

En la arquitectura integrada, los tipos requeridos los datos geométricos y funciones se implementan en el núcleo de SGBDRs. Este enfoque ha demostrado ser el más efectivo, y es el utilizado por la extensión espacial Jaspa por lo que lo explicamos con más detalle en la siguiente sección.

1.5. Arquitectura integrada de Base de datos espacial

En la arquitectura integrada de sistemas de bases de datos, se utilizan sistemas de bases de datos normales a los cuales se les agrega una capa para el manejo de la geometría. Así los usuarios pueden utilizar estos datos espaciales de un modo transparente.

Por tanto, una base de datos espacial es una base de datos que está optimizada para almacenar y consultar los datos con una posición en el espacio, incluyendo puntos, líneas y polígonos. Además para manejar los datos espaciales se necesita dotar a la base de datos estándar de cierta funcionalidad.

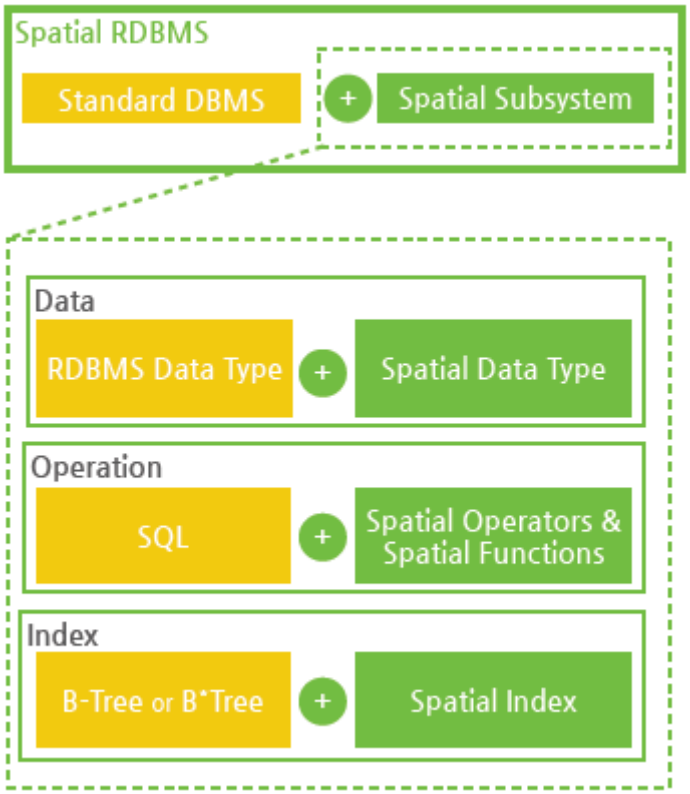


Fig. 1 Extensión espacial sobre una base de datos estándar. Fuente: Ki Sun Song

Los datos espaciales requieren también de un acceso eficiente. Sin embargo, por su propia naturaleza bidimensional o tridimensional se necesita de una indexación diferente a la habitual de las bases de datos estándar.

Los datos espaciales vienen asociados con atributos de carácter alfanumérico. Con la arquitectura integrada se utiliza toda la capacidad del SGBDR para manejar los atributos de carácter alfanumérico asociados a los datos espaciales. Y se hace uso de SQL que es una solución muy contrastada para almacenar los datos.

1.6. Estado actual de las extensiones espaciales

El estado del arte muestra que Oracle Spatial y PostGIS son las extensiones espaciales más consolidadas (Martinez-Llario et al., 2009). Con el fin de comparar las capacidades de Jaspa con otras extensiones espaciales libres, la Tabla 1 (Ramsey, 2007) compara diferentes aspectos como la arquitectura o la funcionalidad.

Tabla 1 Estado actual de las extensiones espaciales

Lang.	Spatial extension	Backend RDBMS	Initial release	Architecture	Functionality	Standards	Spatial index
C	PostGIS	PostgreSQL	2001	Integrated	Full	OGC SFS SQL-MM	GIST
	Ingress geospatial	Ingress	2010	Integrated	Limited	OGC SFS	Yes
	MySQL spatial	MySQL	2004	Integrated	Limited	SQL with geometry types. Not precise spatial operations	MyISAM tables R-tree indexes
	SpatiaLite	SQLite		Layered	Full	OGC-SFS via GEOS	Rtree via SQLite
Java	Spatial DBBox	–	2005		JTS	–	No
	H2 Spatial	H2	2008	Integrated	JTS	–	No
	Hatbox	H2 Derby	2009	Layered	Limited	Partial OGC SFS	Rtree
	GeoDB	H2	2010	Integrated	Limited	Partial OGC SFS	Rtree via HatBox
	Jaspa	PostgreSQL H2	2010	Integrated	Full	OGC SFS SQL-MM	GIST No

Enfoques C

Como muestra la tabla, los proyectos en lenguaje C son considerablemente más maduros. PostGIS proporciona capacidad espacial para el SGBDR PostgreSQL, incluye los tipos de datos geométricos, funciones e indexación espacial. Es la extensión espacial más extendida. Además muchas aplicaciones de los SIG de escritorio o servidores de mapas son capaces de trabajar con datos almacenados en PostgreSQL+PostGIS.

Enfoques Java

Por contra, los proyectos de extensiones espaciales programados en Java denotan menor madurez. En 2005 el equipo de GeoServer publicó SpatialDDBox y más tarde lanzó H2 Spatial. En el año 2009 Peter Yuli publicó la primera versión del proyecto Hatbox y en 2010 Justin Deolivera presentó GeoDB.

El proyecto de código abierto Hatbox[1] implementa la funcionalidad espacial sobre los SGBDRs Derby y H2. Su característica clave es que implementa un índice RTree y únicamente soporta una sola columna de geometría espacial por cada tabla, proporcionando persistencia de indexación RTree. A diferencia de H2 Spatial, se basa en la arquitectura de capas.

La extensión espacial GeoDB[2] está concebida para utilizarse con la base de datos H2. Se basa en la librería JTS y en Hatbox para la indexación espacial. Estos proyectos presentan en general carencias como funcionalidad limitada o falta de índices espaciales.

[1] http://hatbox.sourceforge.net/
[2] http://github.com/jdeolive/geodb

1.7. JTS

JTS[3] es una librería Java que proporciona un modelo de objetos espaciales y funciones fundamentales geométricas en dos dimensiones. Cumple con las especificaciones SFS revisión 1.1 y proporciona una implementación completa, consistente y robusta de algoritmos espaciales bidimensionales.

Entre las ventajas que esta librería aporta al tratamiento de geometrías está:

- el soporte de los objetos definidos en el SFS (puntos, líneas, polígonos y colecciones).
- Las funciones y operaciones que posteriormente se definirán como la intersección, unión, contiene, inclusión, etc.
- La lectura y escritura de datos geométricos en formatos estándares como GML.

Las funciones que proporciona JTS son de análisis espacial de 2 dimensiones. Sin embargo, en la actualidad no existe ninguna biblioteca espacial que trabaje de forma extensa y fiable con predicados u operadores espaciales aplicados a geometrías 3D.

El código fuente de la biblioteca JTS fue modificado ya en la primera versión de Jaspa para almacenar las coordenadas M en las geometrías. Además algunos métodos de JTS se han reprogramado para ampliar su funcionalidad y que utilicen las coordenadas Z y M (por ejemplo ST_Envelope, ST_Locate_Along_Measure o ST_Locate_Between_Measures).

Para el proyecto Jaspa se han modificado asimismo los métodos de lectura y escritura de los formatos WKB, WKT, EWKB y EWKT. Así, se ha conseguido que dichos formatos soporten las coordenadas M y Z. Por último, se introdujo la modificación de tratar las geometrías vacías como valores nulos.

[3] http://www.vividsolutions.com/jts/jtshome.htm

1.8. Datos espaciales

Los datos espaciales se almacenan en las columnas con el tipo de datos “Geometry”. El tipo de geometría “Geometry” en sí es una clase abstracta (no instanciable). Es la clase raíz de la jerarquía de tipos de geometría.

La jerarquía de la geometría se define en el documento de OGC "OpenGIS Simple Features Specification for SQL". La siguiente figura muestra la jerarquía del tipo de datos de geometría y sus subtipos.

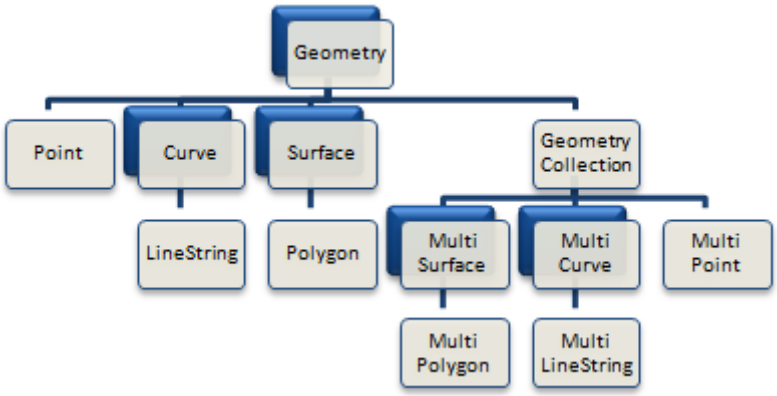


Fig. 2 Diagrama jerárquico tipos de geometrías

En la figura se destacan en azul los tipos no instanciables (Geometry, Curve, Surface, MultiSurface y MultiCurve). Por tanto, para estos tipos no se definen constructores.

Los tipos de geometrías instanciables son siete (Point, LineString, Polygon, GeometryCollection, MultiPoint, MultiLineString, and MultiPolygon).

El estándar SQL-MM añade los tipos CircularString, CompoundCurve y CurvePolygon. Estos nuevos tipos extienden el modelo jerárquico de geometrías OGC con arcos y superficies con límites circulares. Jaspa no ofrece soporte para este tipo de datos.

En la tabla siguiente mostramos un resumen de los subtipos de geometrías instanciables junto con una descripción.

Tabla 2 Subtipos de Geometrias

Subtipos de Geometrías	Descripción
Point	Geometría 0-dimensional. Representa una única ubicación en el espacio de coordenadas.
LineString	Geometría 1-dimensional. Se trata de una curva con interpolación lineal entre puntos.
Polygon	Geometría 2-dimensional. Se define por un anillo exterior de delimitación y cero o más anillos interiores.
MultiPoint	Geometría 0-dimensional. Representa una colección de puntos.
MultiLineString	Geometría 1-dimensional. Representa una colección de líneas.
MultiPolygon	Geometría 2-dimensional. Representa una colección de polígonos.
GeometryCollection	Geometría 0, 1 or 2-dimensional. Colección de una o más geometrías.

1.9. Ejemplos de objetos geométricos

En este apartado se mostrará un ejemplo de cada uno de los tipos de objetos geométricos:

Punto.
Punto tridimensional con coordenadas x=0, y=2, z=10

```
POINT (0 2 10)
```


Línea.
Línea de 3 vértices y 4 dimensiones (x,y,z,m)

```
LINESTRING (1 1 2 5, 2 1 4 8, 2 2 6 10)
```

Polígono.
Polígono con 4 vértices

```
POLYGON ((1 1, 1 3, 3 3, 3 1, 1 1))
```

Multipunto.

Multipunto compuesto por dos puntos. Tiene dos posibles sintaxis.

```
MULTIPOINT ((0 0), (1 1))
```

```
MULTIPOINT (0 0, 1 1)
```

MultiLínea.

Multilinea compuesto por dos líneas

```
MULTILINESTRING ((1 1, 2 1, 2 2), (2 3, 3 3, 3 2))
```

MultiPolígono.

Multipolígono compuesto por dos polígonos que se almacena en una tabla de la base de datos como un único registro. El primer polígono de la colección tiene un hueco.

```
MULTIPOLYGON (((177 199, 186 240, 189 230, 233 137, 228 224,
299 184, 289 73, 195 74, 177 199), (247 143, 253 97, 281 126, 275 161,
247 143)),
((51 5, 13 60, 44 115, 99 117, 105 58, 67 56, 51 5)))
```

Colección de geometrías

Colección de geometrías compuesta por un punto, una línea y un polígono.

```
GEOMETRYCOLLECTION (POINT (0 2),
  LINESTRING (2 3, 3 3, 3 2),
  POLYGON ((5.5 2, 4 4, 5 5, 6.5 4, 5.5 2)))
```

1.10. Interior, Frontera y Exterior

Los conceptos de interior, frontera y exterior se tratan en general en topología, y también están definidos en el estándar comentado anteriormente del OGC "Simple Features for SQL" SFS.

El interior de una geometría es el conjunto de puntos que quedan cuando se eliminan los puntos fronterizos.

La frontera (*boundary*) es la delimitación de los objetos. Está formado por un conjunto de geometrías con una dimensión inmediatamente inferior a la suya.

El exterior de una geometría es el conjunto de puntos, que no están en el interior o la frontera.

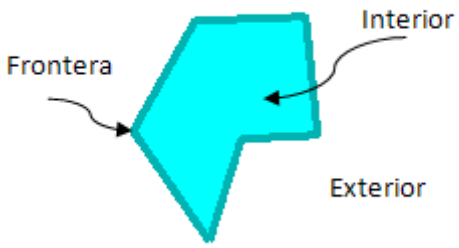


Fig. 3 Frontera, interior y exterior de un polígono

1.11. Operadores espaciales

El establecimiento de las relaciones espaciales entre objetos geométricos es un objetivo primordial con el manejo de información geográfica. Los operadores espaciales son métodos que indican si un objeto cumple una relación específica. Estas relaciones se definen entre dos objetos A y B con geometrías sobre el plano.

Las relaciones espaciales se basan en las definiciones del interior, frontera y exterior de una geometría.

- Contiene: A contiene B si B está incluida en A.
- Contiene propiamente: A contiene propiamente a B si y solo si todos los puntos de B están en el interior de A.
- Cubre: A cubre a B si y sólo si no se encuentran puntos de B en el exterior de A.
- Cruza: A se cruza con B si tiene algún punto en común y la dimensión de la intersección es menor que el máximo de las dimensiones de A y B.

- Disjuntos: A es disjunta de B si las dos geometrías no tienen ningún punto en común en el plano.
- Igual: A es espacialmente igual a B si su geometría es igual.
- Interseca: A interseca con B si las dos geometrías tienen algún punto en común en el plano.
- Solapadas: A se solapa con B si se intersecan y la dimensión de A, B y su intersección son iguales.
- Toca: A se toca con B si el interior de las geometrías no tiene ningún punto en común sobre el plano pero las fronteras de las geometrías sí que tienen puntos es común.

Para extraer las relaciones espaciales se utiliza la matriz DE-9IM (Dimensionally Extended-9 Intersection Model). Es un método matemático para comparar dos geometrías en la que se calcula la dimensión de las intersecciones entre los interiores, fronteras y exteriores de las dos geometrías. Este método fue desarrollado por Clementini et al., que extiende el Modelo de Intersección de Egenhofer y Herring.

En la siguiente tabla se expone que patrón de la matriz DE-9IM se corresponde con los diferentes operadores espaciales disponibles en Jaspa.

Tabla 3 Operadores espaciales disponibles en Jaspa y su correspondiente patrón DE-9IM

Operadores espaciales	Patrón DE-9IM
ST_Contains	**[T** *** FF*]**
ST_ContainsProperly	**[T** FF* FF*]**
ST_Covers	**[T** *** FF*] or [*T* *** FF*] or [*** T** FF*] or [*** *T* FF*]**
ST_CoveredBy	**[T*F **F ***] or [*TF **F ***] or [**F T*F ***] or [**F *TF ***]**
ST_Crosses	**[T*T *** ***] (para Puntos/Líneas, Puntos/Polígonos y Líneas/ Polígonos)** **[T** *** T**] (para Líneas/Puntos, Líneas/Polígonos y Polígonos/Líneas)** **[0** *** ***] (para Líneas / Líneas)**
ST_Disjoint	**[FF* FF* ***]**
ST_Equals	**[T*F **F FF*]**
ST_Intersects	**[T** *** ***] or [*T* *** ***] or [*** T** ***] or [*** *T* ***]**
ST_Overlaps	**[T*T *** T**] (para Puntos/Puntos y Polígonos/Polígonos)** **[1*T *** T**] (para Líneas / Líneas)**
ST_Touches	**[FT* *** ***] or [F** T** ***] or [F** *T* ***]**
ST_Within	**[T*F **F ***]**

1.12. Funciones espaciales

La funciones son métodos que actúan sobre las geometrías y devuelven un resultado, pero en este caso el resultado es una nueva geometría fruto del cálculo.

- Centroide: El centro de A es punto situado en su centro de gravedad.

- Buffer: El buffer de un objeto A nos devuelve una nueva geometría que cubre todos los puntos desde la frontera de A hasta una cierta distancia D de A.

- Intersección (AND): La intersección de A y B devuelve un objeto geométrico que incluye a todos los puntos del plano incluidos tanto en A como en B.

- Unión (OR): La unión de A y B devuelve un objeto geométrico que incluye todos los puntos del plano incluidos en A o en B.

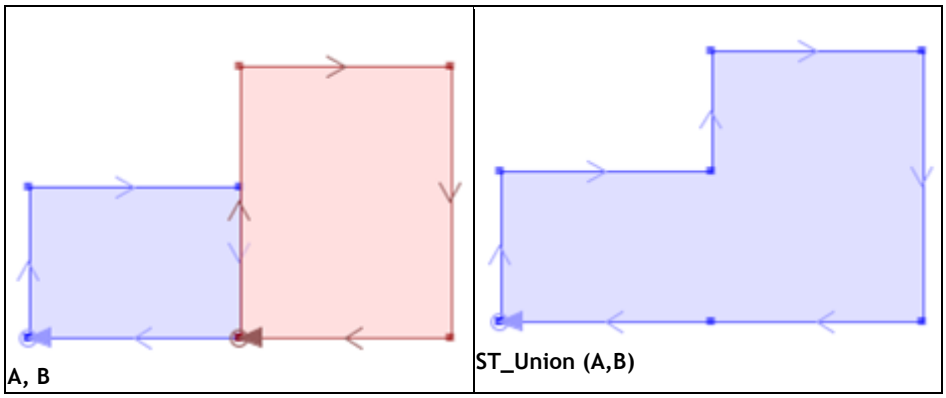


Fig. 4 Ejemplo función unión

- Diferencia: La diferencia de A y B devuelve un objeto geométrico que incluye a todos los puntos del plano incluidos en A y no en B.

- Diferencia simétrica (XOR): La diferencia simétrica de A y B devuelve un objeto geométrico que incluye todos los puntos del plano incluidos en A y no en B o viceversa.

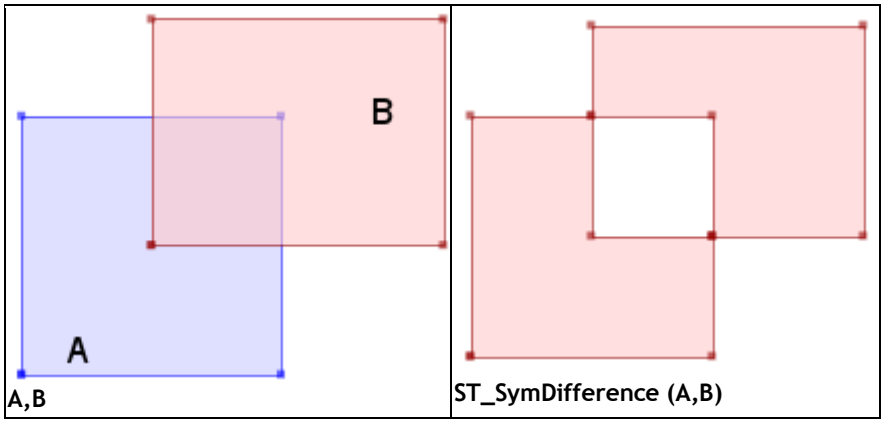


Fig. 5 Ejemplo diferencia simétrica

- Distancia: La distancia entre A y B nos devuelve la longitud de la línea recta más corta entre A y B.

2 EXTENSIÓN ESPACIAL JASPA

2.1. Introducción

Jaspa (Java Spatial) es un proyecto reciente que empezó a desarrollarse a mediados del 2009. La primera versión 0.1 se publicó con licencia GPL en Julio del 2010. La segunda versión 0.2 apareció en el verano de 2011 e incluye un sistema de reglas topológicas para PostgreSQL.

El objetivo principal de Jaspa es ofrecer una alternativa sólida de Java para PostGIS que implemente no sólo la especificación del OGC, sino otras funcionalidades y funciones como agregados no considerados por el OGC.

Jaspa es un proyecto que se ha desarrollado en un ambiente académico. Al ser un proyecto de software libre cualquier usuario puede acceder libremente, estudiar cómo funciona e incluso modificarlo.

Jaspa implementa la especificación OpenGIS Simple Features for SQL y parcialmente el estándar SQL/MM. Se basa en las librerías JTS (Davis, 2006) y GeoTools4. Java Topology Suite (JTS) es una librería muy utilizada en Jaspa y por muchos otros proyectos de software libre, ofreciendo la posibilidad de realizar el análisis espacial.

2.2. Arquitectura

Jaspa se basa en la arquitectura integrada. Desde el punto de vista de la portabilidad, Jaspa es compatible con las bases de datos PostgreSQL y H2. En el presente documento estudiaremos como integrarlo con Oracle 11g.

PostgreSQL es una base de datos objeto-relacional a la que se puede acceder y extender por varios lenguajes de programación. En concreto, Jaspa extiende la funcionalidad de PostgreSQL utilizando el lenguaje procedural PL/Java.

H2 es un SGBDR ligero y de alto rendimiento, de código abierto y completamente escrito en Java. Como pudimos ver en la tabla 1, H2 es hoy en día la principal base de datos de soporte para las extensiones espaciales programadas en Java.

[4] http://geotools.org/

2.3. Almacenamiento de Geometrías

Jaspa ofrece soporte para los formatos estándares WKT (well-known text) y WKB (well-known binary) y para los formatos de PostGIS EWKT (extended well-known text) y EWKB (extended well-known binary). También ofrece soporte para SHP (ESRI Shapefile), KML (Keyhole Markup Language) y GML (Geography Markup Language).

Las geometrías se almacenan en las bases de datos como objetos binarios grandes (BLOB). En concreto H2 utiliza el tipo de dato Varbinary, y PostgreSQL usa el tipo de dato Bytea.

2.4. Conformidad SQL para implementar Jaspa

El estándar ISO 13249-3 define las características que debe tener un SGBDR para la conformidad de SQL. Algunas de estas características son esenciales para implementar la funcionalidad Jaspa:

- Feature T322, "Overloading of SQL-invoked functions and procedures". La sobrecarga de funciones y procedimientos SQL es totalmente compatible con PostgreSQL, y parcialmente por H2. Oracle también lo soporta pero siempre que las funciones y procedimientos estén dentro de un paquete.

- Feature S201, "SQL-invoked routines on arrays". Funciones que tomen como entrada Arrays (por ejemplo, MakeLine, Accum)

- Feature S202, "SQL-invoked routines on multisets". Invocación de rutinas SQL sobre conjuntos múltiples, este aspecto es soportado por Oracle y será tratado en el capítulo de los agregados espaciales.

2.5. Funcionalidad

Jaspa ofrece 179 funciones principales para manejar datos espaciales que se pueden clasificar en funciones de gestión, constructores de geometrías, acceso a geometrías, editores de geometrías, salida de geometrías, relaciones espaciales y mediciones, geoprocesamiento, referencia lineal, y miscelánea.

Tabla 4 Funcionalidad de Jaspa

Tipo de funciones	Ejemplos
Management Functions	7 funciones (AddGeometryColumn, DropGeometryTable, Jaspa_Version …)
Geometry Constructors	34 funciones (ST_GeomFromText , ST_MakeLine, ST_MakePolygon, ST_MakeGeomColl)
Geometry Accessors	29 funciones (ST_CoordDim, ST_IsClosed, ST_IsValid …)
Geometry Editors	27 funciones (ST_Affine, ST_Force_2D, ST_Multi, ST_NodeLine, ST_Transform …)
Geometry Outputs	7 funciones (ST_AsBinary , ST_AsGML, ST_AsHEXEWKB, ST_AsText …)
Spatial Relationships and Measurements	30 funciones (ST_Centroid, ST_Contains, ST_Disjoint, ST_Distance, ST_Within …)
Geometry Processing Functions	19 funciones (ST_Buffer, ST_Collect, ST_Simplify, ST_Union …)
Linear Referencing	8 funciones (ST_Line_Interpolate_Point, ST_Line_Substring, ST_Locate_Along_Measure)
Miscellaneous Functions	17 funciones (ST_Accum, ST_Box2D, ST_Expand, ST_Extent, ST_Xmax, ST_XMin …)

Muchas de estas funciones se basan en las librerías JTS y GeoTools. Jaspa utiliza JTS en la mayoría de las funciones y operadores espaciales.

Por el contrario, la librería GeoTools se utiliza secundariamente. En concreto se emplea en el conversor de Shapefile (SHP) a SQL, reproyecciones entre sistemas espaciales de referencia y conversores de KML y GML.

Los agregados SQL como SUM o AVG son muy comunes, mientras que las funciones de agregados de datos espaciales se limitan a un número finito de operaciones (Benedikt and Libkin, 2002). De hecho, las funciones de los agregados no están definidas en la especificación OGC SFS. Jaspa extiende la especificación para incorporar algunos agregados espaciales como Union o Extent.

2.6. Ventajas del uso de procedimientos almacenados en Java y Jaspa

Para conseguir la funcionalidad espacial Jaspa utiliza la librería JTS directamente. En cambio PostGIS utiliza GEOS que es una portabilidad de JTS en C. Esta característica permite que las actualizaciones de JTS se puedan incorporarán antes que en los sistemas basados en C. A su vez, el desarrollo de nuevos procedimientos almacenados en Jaspa junto con JTS y GeoTools es más sencillo que en PostGIS.

Jaspa utiliza métodos de la clase JP, que trabajan directamente con geometrías JTS, lo que ofrece todo el potencial de la biblioteca JTS y otras bibliotecas compatibles, lo que incrementa su potencial.

Otra ventaja es que es posible utilizar un entorno de desarrollo integrado (IDE) como Eclipse o NetBeans en la fase de desarrollo. Estos sistemas son mucho más avanzados que los actuales entornos de desarrollo de PL/SQL, lo que acelera enormemente el tiempo empleado en el desarrollo y reduce costos.

La comunidad de usuarios es reducida, pero el hecho de estar escrito en Java puede ser un incentivo para su progreso. Por ejemplo, Jaspa representa una oportunidad para la integración de las tecnologías basadas en Java como GeoTools con la base de datos.

Al ser una extensión espacial programada en Java, se puede migrar su funcionalidad a otras bases de datos. En esa misma línea, surge este proyecto para poder dotar a la base de datos Oracle de capacidades espaciales, utilizando Jaspa en lugar de Oracle Spatial.

3 ORACLE

3.1. Oracle y sus extensiones espaciales (Spatial y Locator)

Oracle Locator, también llamado Locator, es una característica de la base de datos Oracle 11g. Está disponible en cada una de las diferentes ediciones (*Express Edition, Standard Edition, Standard Edition One y Enterprise Edition*). Proporciona funciones y servicios básicos de los disponibles en Oracle Spatial.

Locator no está diseñado para ser una solución para Sistemas de Información Geográfica que requieren una gestión compleja de datos espaciales. Para ese propósito se debería utilizar Oracle Spatial en lugar de Locator.

Por el contrario, Oracle Spatial es una opción para la que debe pagarse una licencia extra y disponible sólo con Oracle Database 11g Enterprise Edition. Oracle Spatial incluye todas las características de localización, así como otras características que no están disponibles con Locator.

En general, Locator incluye los tipos de datos, operadores, la capacidad de indexación de Oracle Spatial, y una cantidad limitada de funciones y procedimientos de Oracle Spatial. Locator incluye las siguientes características:

- Tipo de objeto SDO_GEOMETRY que describe y soporta los diferentes tipos de geometrías.

- Capacidad de indexación espacial que permite crear índices espaciales en los datos de geometrías.

- Operadores espaciales que utilizan índices espaciales para las consultas espaciales.

- Algunas funciones geométricas y el agregado espacial SDO_AGGR_MBR.

- Soporte para el sistema de coordenadas para la transformación de coordenadas de geometrías y de metadatos de capas (función SDO_CS.TRANSFORM y procedimiento SDO_CS.TRANSFORM_LAYER).

- Funciones y procedimientos de optimización (paquete SDO_TUNE).

- Funciones de utilidades espaciales (paquete SDO_UTIL).

- Integración con Oracle Application Server.

3.2. Licencias

Las licencias Oracle son de tipo “Procesador” o “Usuario Designado Plus”. El programa de licencias de procesador de Oracle está diseñado para entornos grandes, con un elevado número de usuarios o en los que no se puede identificar ni contar a éstos con facilidad, como es el caso de las aplicaciones basadas en Web. La métrica de procesador también se utiliza cuando resulta más rentable que las licencias de Usuario Nombrado Plus. Todos los procesadores en los que se instalen o se ejecuten los programas de Oracle deben contar con una licencia.

Fig. 6 Oracle Spatial en la tienda online de Oracle

En Oracle Store (https://shop.oracle.com/pls/ostore/) podemos adquirir los diferentes productos de Oracle. Asimismo podemos informarnos sobre sus características y precios.
En la siguiente tabla se indican los costes de licencias de Oracle. Se omiten otros costes como los de soporte.

	Tipo de licencia	
Producto	Procesador	Usuario Designado Plus
Oracle Database Enterprise Edition	37,492 €	750€ (pedido mínimo 25)
Oracle Database Standard Edition	2,763 €	55€ (pedido mínimo 25)
Oracle Spatial	13,813 €	276€ (pedido mínimo 25)

Tabla 5 Costes por unidad de licencia Oracle con duración perpetua (datos en enero 2012)

3.3. Geometrías en Oracle. SDO_GEOMETRY

Los objetos espaciales se almacenan en una columna con tipo de objetos SDO_GEOMETRY. Cualquier tabla que tenga una columna de tipo SDO_GEOMETRY debe tener otra columna, o conjunto de columnas, que defina una clave primaria única para la tabla. Las tablas de este tipo se denominan tablas espaciales.

Oracle Spatial define el tipo de objeto SDO_GEOMETRY como:

```
CREATE TYPE sdo_geometry AS OBJECT (
        SDO_GTYPE NUMBER,
        SDO_SRID NUMBER,
        SDO_POINT SDO_POINT_TYPE,
        SDO_ELEM_INFO SDO_ELEM_INFO_ARRAY,
SDO_ORDINATES SDO_ORDINATE_ARRAY);
```

El atributo **SDO_GTYPE** indica el tipo de geometría. Los tipos de geometrías válidos se corresponden a los especificados en la especificación *Geometry Object Model for the OpenGIS Simple Features for SQL* (con la excepción de los polígonos). Los valores numéricos son diferentes de los de la especificación de OpenGIS, y tampoco hay una correspondencia directa entre los nombres y la semántica en su caso.

El atributo SDO_GTYPE está compuesto por 4 dígitos con el formato DLTT, donde:

- D indica el número de dimensiones (2, 3, or 4).
- L identifica que dimensión (3 o 4) contiene el valor de las medidas (coordenada M). Para aceptar el valor por defecto se indica 0.
- TT identifica el tipo de geometría.

El atributo **SDO_SRID** se utiliza para identificar un sistema espacial de coordenadas asociado con las geometrías.

El atributo **SDO_POINT** se utiliza para el tipo de objetos SDO_POINT_TYPE.

El atributo **SDO_ELEM_INFO** sirve para interpretar los valores del atributo SDO_ORDINATES.

El atributo **SDO_ORDINATES** almacena las coordenadas de cada objeto espacial. Por ejemplo, un polígono con cuatro puntos se almacena como {X1, Y1, X2, Y2, X3, Y3, X4, Y4, X1, Y1}.

3.4. Geometría: Rectángulo

En el capítulo de indexación espacial necesitaremos crear rectángulos con el tipo SDO_GEOMETRY para definir las cajas delimitadoras mínimas (Minimum Bounding Box) de las geometrías a indexar. Por ello explicaremos a continuación los valores de los atributos SDO_GEOMETRY correspondientes a la geometría rectángulo:

- SDO_GTYPE = 2003. El 2 indica que la geometría es de dos dimensiones, y el 3 indica que se trata de un polígono.

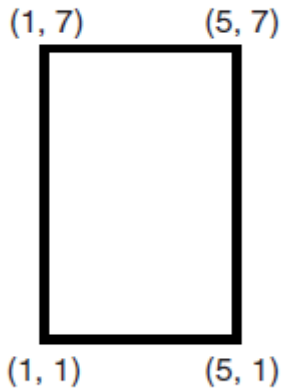
- SDO_SRID = NULL.

- SDO_POINT = NULL.

- SDO_ELEM_INFO = (1, 1003, 3). El 3 de la tupla (1,1003,3) indica que se trata de un rectángulo. Al ser un rectángulo, sólo se especifican dos coordenadas en el parámetro SDO_ORDINATES (inferior izquierda y superior derecha).

- SDO_ORDINATES = (Xmin,Ymin,Xmax,Ymax). Identifican las coordenadas inferior izquierda y superior derecha del rectángulo.

El siguiente ejemplo muestra una sentencia SQL de inserción de un rectángulo SDO_GEOMETRY de coordenadas inferior izquierda x=1, y=1, y superior derecha x=5, y=7.



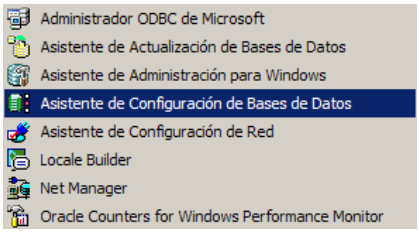

```
INSERT INTO cola_markets VALUES(
        1,
        'cola_a',
        SDO_GEOMETRY(
        2003, -- two-dimensional polygon
        NULL,
        NULL,
        SDO_ELEM_INFO_ARRAY(1,1003,3),
        -- one rectangle (1003 = exterior)
        SDO_ORDINATE_ARRAY(1,1, 5,7)
        -- only 2 points needed to
        -- define rectangle (lower left and
        --upper right) with Cartesian
        ---coordinate data
        )
);
```

Fig. 7 Rectángulo de ejemplo y sentencia SQL de inserción

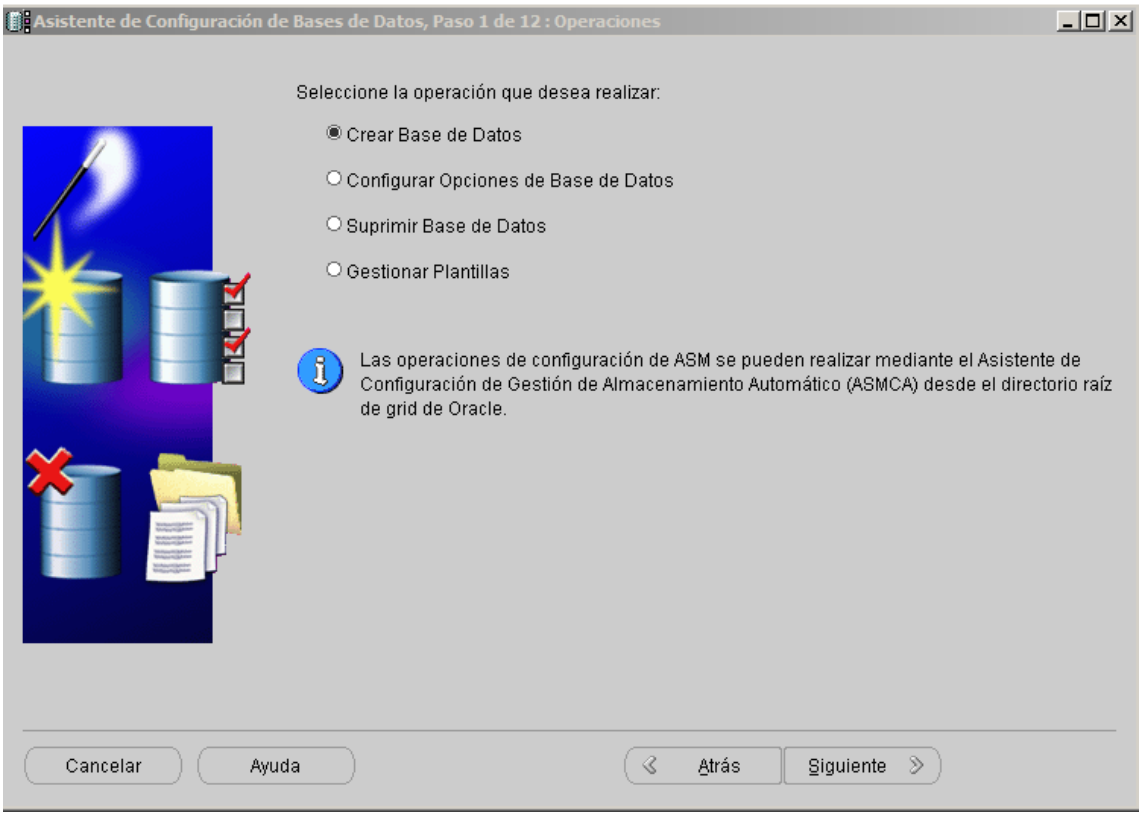
3.5. Creación Base de Datos

En este paso explicaremos los pasos básicos para crear una base de datos Oracle 11g que a la que posteriormente añadiremos la funcionalidad de Jaspa. Cabe notar que en esta base de datos se van a establecer muchos valores por defecto. Sería labor del Administrador de Base de Datos (DBA) establecer las optimizaciones correspondientes a cada base de datos concreta.

Podemos crear una base de datos utilizando el asistente de configuración de bases de datos que incorpora. Para ello entre todas las herramientas de Oracle instaladas, seleccionamos la herramienta “Asistente de Configuración de Bases de Datos”.



Se iniciará el Asistente de Configuración de Bases de Datos, pulsaremos "Siguiente". A continuación marcaremos la opción "Crear una base de datos" y pulsaremos "Siguiente":



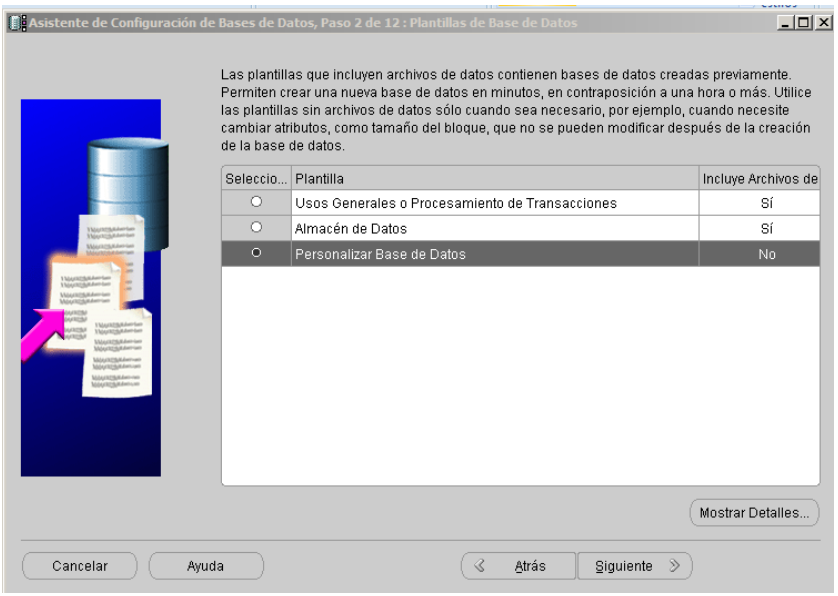
En el siguiente paso, dependiendo de las opciones que queramos establecer, podemos seleccionar plantillas existentes o crear nuestra propia plantilla:

- **Usos Generales o Procesamiento de Transacciones**: plantilla existente con unos datos de configuración estándar para crear una base de datos de propósito general que podrá ser usada por aplicaciones habituales de contabilidad, facturación, gestión, nóminas, recursos humanos, etc. Está pensada para una base de datos con muchos accesos, muchas consultas y para guardar una cantidad de datos "normal".

- **Almacén de Datos**: plantilla existente con unos datos de configuración estándar para crear una base de datos para almacenamiento de información que pueden ser usadas como archivo histórico. Es una base de datos configurada para pocos accesos y guardar gran cantidad de datos.

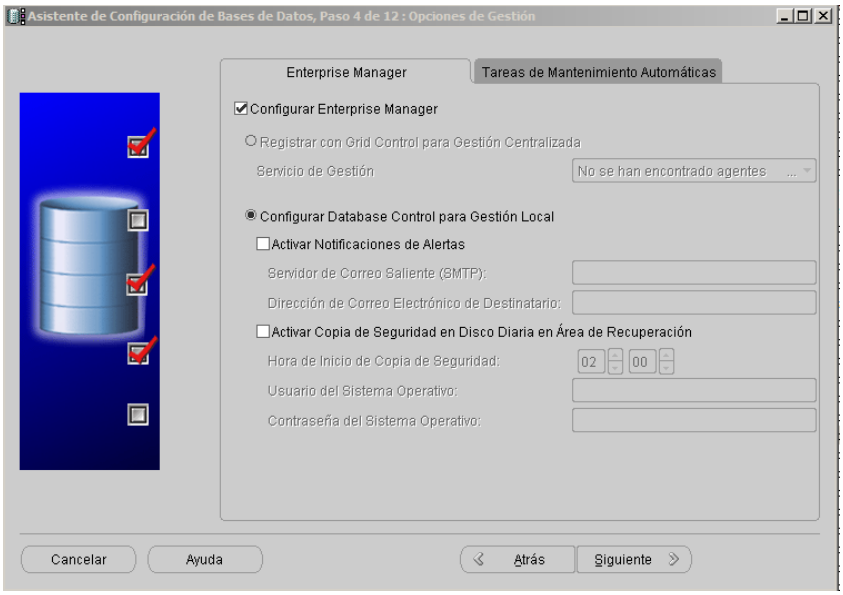
- **Personalizar Base de Datos**: con esta opción podremos crear nuestra propia plantilla. Seleccionando esta opción podremos configurar y establecer todos los parámetros de la base de datos. Está recomendada para usuarios avanzados.

En nuestro caso elegimos la opción “Personalizar Base de Datos”.



En el siguiente paso introduciremos el nombre de la Base de Datos Global y el SID (pueden ser el mismo), en nuestro caso "generaldb".

A continuación elegimos en “Configurar Enterprise Manager” las opciones proporcionadas por defecto.



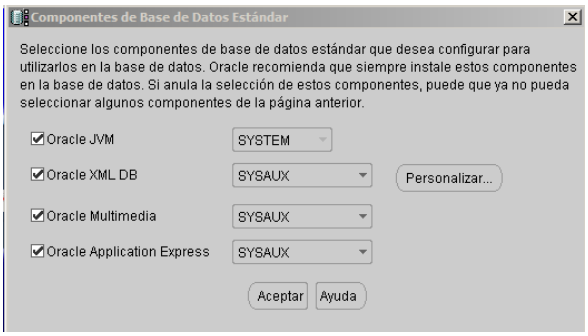
Al darle a "Siguiente" podremos indicar la contraseña para los usuarios que el Asistente de Configuración de Base de Datos creará: SYS, SYSTEM, DBNSMP y SYSMAN. Podremos indicar una contraseña para cada usuario o la misma contraseña para todos los usuarios

En la opción de "tipo de almacenamiento y ubicaciones para los archivos de base de datos" dejamos la configuración por defecto.

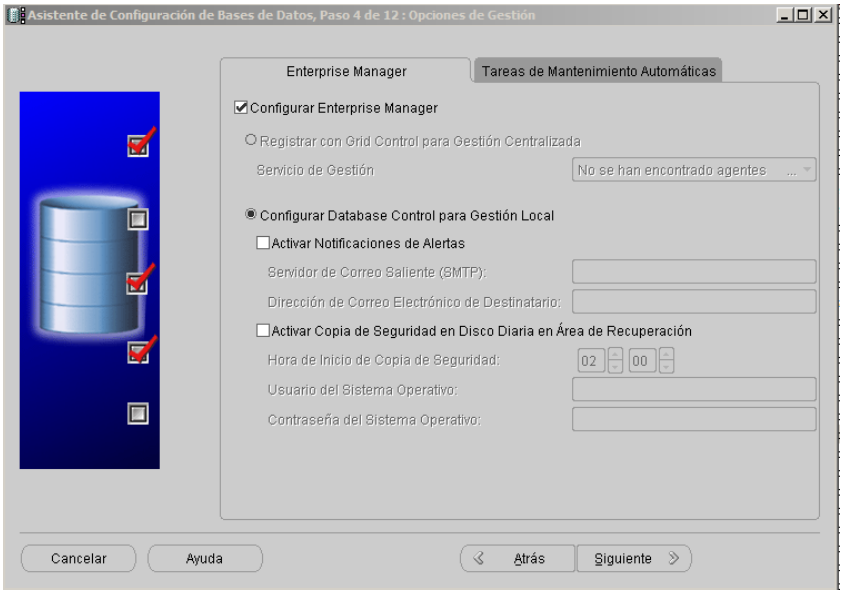
Igualmente en "Opciones de recuperación para la base de datos" dejamos los valores por defecto.

En la siguiente ventana marcaremos las funciones que queramos implementar, nos aseguramos de marcar que NO queremos instalar Oracle Spatial.

En “Componentes de Base de Datos Estándar”, es fundamental marcar que vamos a utilizar JVM (Java Virtual Machine).

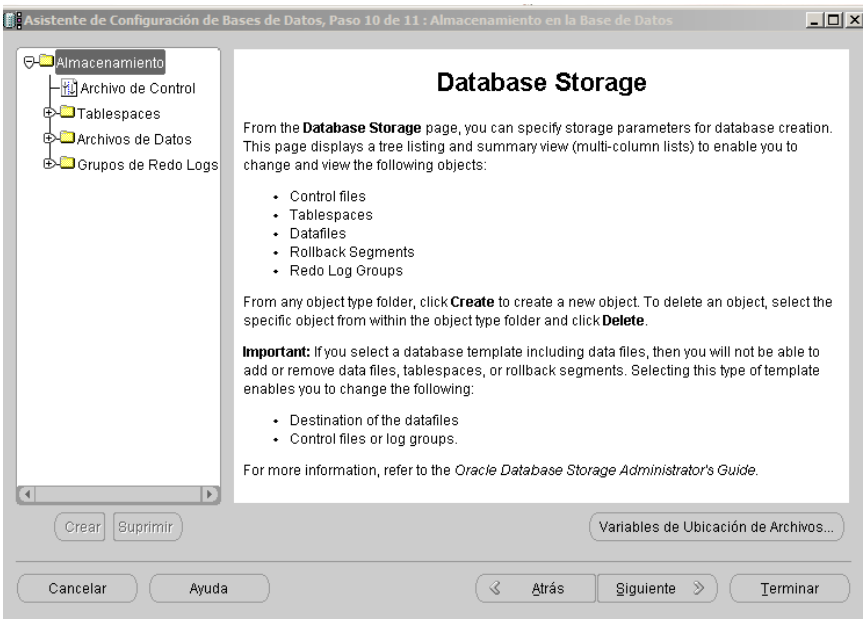


A continuación se pueden introducir diferentes opciones de memoria, tamaño, juego de caracteres y modo de conexión. Podemos modificarlas o seleccionar las opciones disponibles por defecto.



Seguidamente podremos indicar las opciones de almacenamiento. Desde esta página especificaremos los parámetros de almacenamiento para la base de datos. Esta página muestra un listado en árbol y una vista de resumen (lista de varias columnas) que permite cambiar y ver los siguientes objetos:

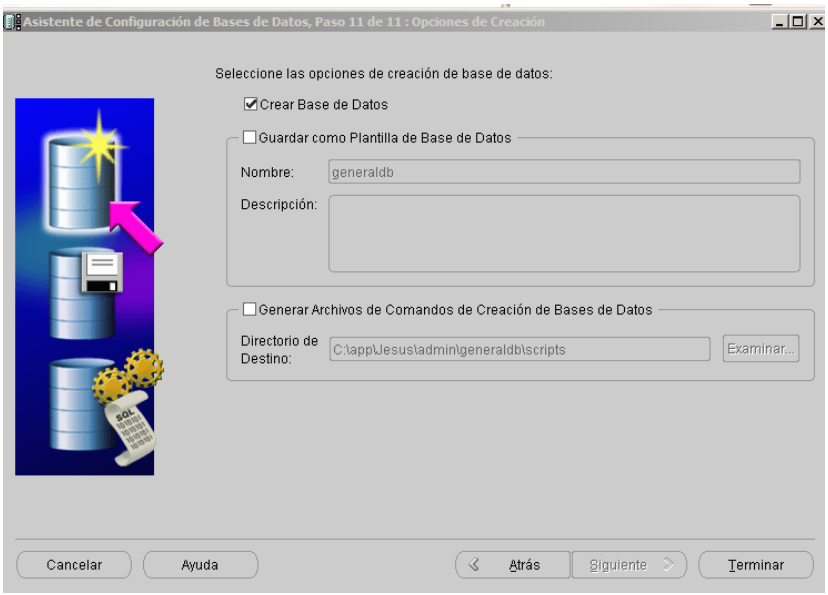
- Archivos de Control.
- Tablespaces.
- Archivos de Datos.
- Segmentos de Rollback.
- Grupos de Redo Logs.



Por último, antes de crear la base de datos, el asistente nos da las siguientes posibilidades:

- **Crear Base de Datos**: para crear la base de datos ahora.
- **Guardar como Plantilla de Base de Datos**: para guardar los parámetros de creación de la base de datos como plantilla. Esta plantilla se agregará automáticamente a la lista de plantillas de base de datos disponibles.
    - Nombre: es el título que le damos a la plantilla y que aparecerá posteriormente en la lista de plantillas preconfiguradas.
    - Descripción: descripción breve del tipo de plantilla.
- **Generar Archivos de Comandos de Creación de Bases de Datos**: esta opción permite generar archivos de comandos de creación de bases de datos para la plantilla de base de datos seleccionada. Activando esta casilla se accede a todos los archivos de comandos utilizados para crear la base de datos. Los archivos de comandos se generan a partir de los parámetros de base de datos proporcionados en las páginas anteriores. Los archivos de comandos sirven como lista de control o para crear la base de datos sin utilizar el Asistente de Creación de Bases de Datos Oracle.

En nuestro caso marcaremos tan solo la primera opción:



Tras pulsar en "Terminar" en la ventana anterior podremos ver todas las opciones seleccionadas:

Resumen

Resumen de Configuración de Base de Datos

Nombre de la Base de Datos Global:	generaldb
Tipo de Configuración de Base de Datos:	Instancia Única
SID:	generaldb
Tipo de Opción de Gestión:	Database Control
Tipo de Almacenamiento:	Sistema de Archivos
Tipo de Configuración de Memoria:	Gestión Automática de Memoria

Detalles de Configuración de Base de Datos

Componentes de la Base de Datos

Componente	Seleccionado
Oracle JVM	true
Oracle Text	true
Oracle XML DB	true
Oracle Multimedia	true
Oracle OLAP	false
Oracle Spatial	false
Oracle Label Security	false
Esquemas de Ejemplo	false
Repositorio de Enterprise Manager	true

Oracle Application Express	true
Oracle Warehouse Builder	false
Oracle Database Vault	false
Oracle Database Extensions for .NET	false

Parámetros de Inicialización

Nombre	Valor
audit_file_dest	{ORACLE_BASE}\admin\{DB_UNIQUE_NAME}\adump
audit_trail	db
compatible	11.2.0.0.0
control_files	("{ORACLE_BASE}\oradata\{DB_UNIQUE_NAME}\control01.ctl", "{ORACLE_BASE}\flash_recovery_area\{DB_UNIQUE_NAME}\control02.ctl")
db_block_size	8KB
db_domain	
db_name	generald
db_recovery_file_dest	{ORACLE_BASE}\flash_recovery_area
db_recovery_file_dest_size	4977MB
db_unique_name	generaldb
diagnostic_dest	{ORACLE_BASE}
dispatchers	(PROTOCOL=TCP) (SERVICE={SID}XDB)
memory_target	818MB
nls_language	SPANISH
nls_territory	SPAIN
open_cursors	300
processes	150
remote_login_passwordfile	EXCLUSIVE
undo_tablespace	UNDOTBS1

Juegos de Caracteres

Nombre	Valor
Juego de Caracteres de la Base de Datos	WE8MSWIN1252
Juego de Caracteres Nacional	AL16UTF16

Archivos de Control

Nombre
{ORACLE_BASE}\oradata\{DB_UNIQUE_NAME}\control01.ctl

{ORACLE_BASE}\flash_recovery_area\{DB_UNIQUE_NAME}\control02.ctl

Tablespaces

Nombre	Tipo	Gestión de Extensiones
SYSAUX	PERMANENT	LOCAL
SYSTEM	PERMANENT	LOCAL
TEMP	TEMPORARY	LOCAL
UNDOTBS1	PERMANENT , UNDO	LOCAL
USERS	PERMANENT	LOCAL

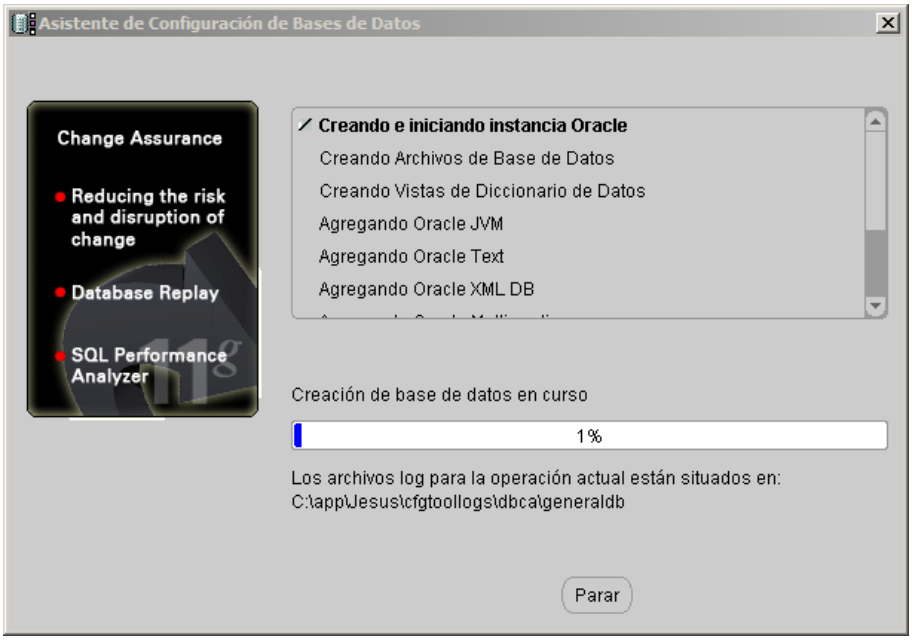
Archivos de Datos

Nombre	Tablespace	Tamaño (M)
{ORACLE_BASE}\oradata\{DB_UNIQUE_NAME}\sysaux01.dbf	SYSAUX	600
{ORACLE_BASE}\oradata\{DB_UNIQUE_NAME}\system01.dbf	SYSTEM	700
{ORACLE_BASE}\oradata\{DB_UNIQUE_NAME}\temp01.dbf	TEMP	20
{ORACLE_BASE}\oradata\{DB_UNIQUE_NAME}\undotbs01.dbf	UNDOTBS1	200
{ORACLE_BASE}\oradata\{DB_UNIQUE_NAME}\users01.dbf	USERS	5

Grupos de Redo Logs

Grupo	Tamaño (K)
1	51200
2	51200
3	51200

Finalmente se iniciará la creación definitiva de la base de datos Oracle:



3.6. Creación de un nuevo usuario

Un usuario es una cuenta a través de la cual se puede acceder a la base de datos. La sentencia SQL CREATE USER crea un usuario. Para crear un usuario se debe tener el privilegio de sistema CREATE USER.

Los usuarios sys y system, que se crean automáticamente al crear la base de datos, tienen privilegios DBA para crear usuarios por lo que podemos iniciar sesión con cualquiera de ellos para crear los usuarios necesarios.

Lo que puede hacer un usuario una vez ha accedido a la base de datos depende de los permisos que tenga asignados. La sentencia GRANT sirve para dar permisos (o privilegios) a un usuario o a un rol.

También se debe otorgar al usuario una cierta cuota de espacio en los tablespaces, en nuestro caso se lo otorgaremos en el tablespace users.

Con las siguientes sentencias vamos a crear desde SQL Developer un usuario de nombre "jaspadb" con todos los derechos necesarios para desplegar Jaspa en Oracle:

```
-- USER SQL
CREATE USER jaspadb IDENTIFIED BY password;

-- ROLES
GRANT "JAVA_DEPLOY" TO jaspadb ;
GRANT "JAVAUSERPRIV" TO jaspadb ;
GRANT "JAVA_ADMIN" TO jaspadb ;
GRANT "JMXSERVER" TO jaspadb ;
GRANT "JAVASYSPRIV" TO jaspadb ;
GRANT "CONNECT" TO jaspadb ;
GRANT "JAVADEBUGPRIV" TO jaspadb ;
GRANT "JAVAIDPRIV" TO jaspadb ;

-- SYSTEM PRIVILEGES
GRANT CREATE TRIGGER TO jaspadb ;
GRANT EXECUTE ANY OPERATOR TO jaspadb ;
GRANT EXECUTE ANY TYPE TO jaspadb ;
GRANT CREATE TABLE TO jaspadb ;
GRANT EXECUTE ANY CLASS TO jaspadb ;
GRANT CREATE PROCEDURE TO jaspadb ;
GRANT EXECUTE ANY RULE TO jaspadb ;
GRANT EXECUTE ANY PROCEDURE TO jaspadb ;
GRANT EXECUTE ANY PROGRAM TO jaspadb ;
GRANT EXECUTE ANY RULE SET TO jaspadb ;
GRANT CREATE SYNONYM TO jaspadb ;
GRANT CREATE PUBLIC SYNONYM TO jaspadb ;
GRANT CREATE VIEW TO JASPADB ;
GRANT CREATE TYPE TO JASPADB ;
GRANT CREATE SEQUENCE TO JASPADB ;

-- QUOTAS
ALTER USER jaspadb QUOTA UNLIMITED ON USERS;
```

Para cargar los sistemas de referencia (proporcionados por un plugin de GeoTools) desde PL/SQL vamos a llamar a un procedimiento almacenado de Java que manipula archivos del sistema operativo. Por ello, debemos darle al usuario JASPADB permiso para que acceda en tiempo de ejecución. Con el usuario administrador ejecutamos el siguiente comando:

```
call dbms_java.grant_permission ('JASPADB',
'SYS:java.lang.RuntimePermission', 'accessDeclaredMembers', '' );
commit;
```

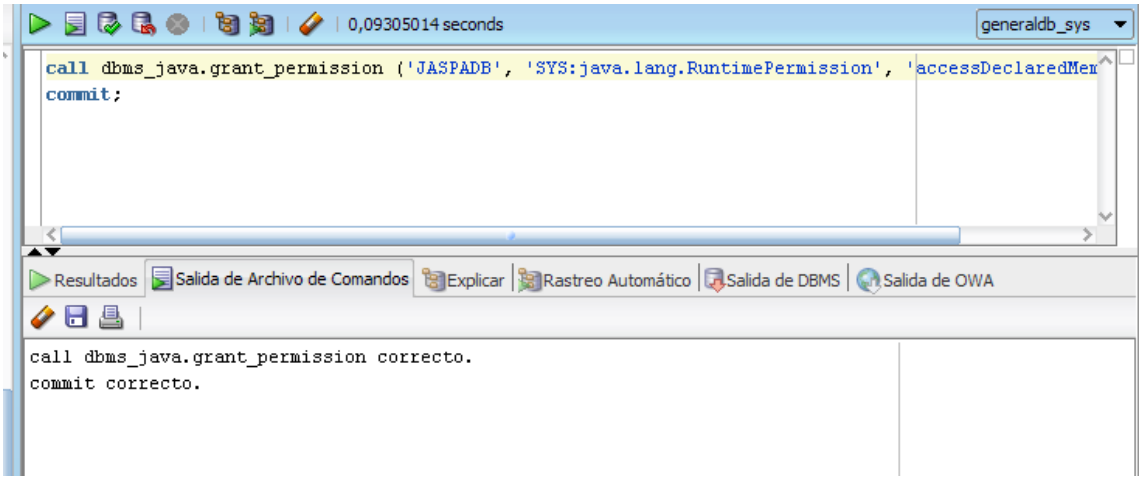


Fig. 8 Otorgamiento de permisos desde SQL Developer

4. DISEÑO Y DESARROLLO

4.1. Carga de librerías necesarias. Loadjava

Oracle permite importar clases de Java creadas de forma convencional que se encuentran fuera de la base de datos, este es el caso en el que interviene la utilidad **loadjava**. A través de esta utilidad podemos cargar código fuente Java y clases dentro de nuestras bases de datos ORACLE.

Para ejecutar esta utilidad se requiere tener privilegios CREATE PROCEDURE y CREATE TABLE dentro del esquema en el que vayamos a cargar las clases java).

En caso de que vayamos a cargar clases dentro de un esquema que no sea el nuestro los privilegios que debemos tener son CREATE ANY PROCEDURE y CREATE ANY TABLE.

Si queremos cargar una clase Java "clasejava.java" dentro de nuestra base de datos, escribimos en la línea de comandos el siguiente ejemplo:

```
loadjava -user scott/tiger clasejava.java
```

La herramienta loadjava es una utilidad de línea de comandos del sistema operativo que carga los archivos de Java en la base de datos. Esta herramienta crea objetos de esquema desde archivos y los carga en un esquema. Tal y como muestra en la siguiente figura, los objetos de esquema pueden ser creados a partir de código fuente de Java, clases y archivos de datos.

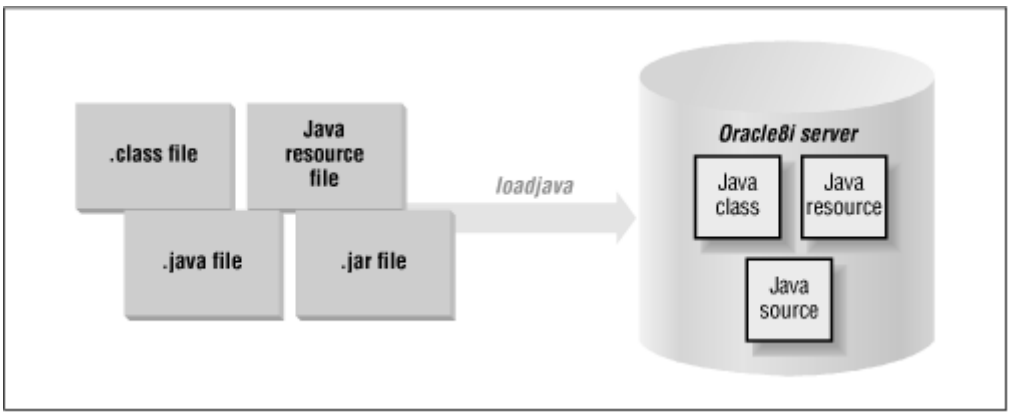


Fig. 9 Carga de elementos Java en Oracle [4]

Para ver todas las posibilidades que tenemos con la utilidad loadjava podemos escribir en línea de comandos la instrucción “loadjava”:

```
C:\>loadjava

loadjava: Usage: loadjava [-definer] [-encoding encoding] [-force] [-genmissing] [-genmissingjar jar] [-grant grants] [-help] [-nousage] [-noverify] [-oci8] [-order] [-resolve] [-nativecompile] [-resolver resolver] [-schema schema] [-synony m] [-thin] [-tableschema schema] [-user user/password@database] [-verbose] [-edition edition] classes..jars..resources..properties...
```

Algunos de los parámetros más destacables de loadjava son:

Tabla 6 Opciones de loadjava

Parámetro	Descripción
-debug	Genera y visualiza información de depuración
-force	Carga las clases aunque ya hayan sido previamente cargadas
-help	Visualiza la ayuda
-resolve	Resuelve las referencias externas
-resolver	Permite indicar paquetes de otros esquemas con los que se debe realizar la resolución de referencias externas
-schema	Carga los ficheros en el esquema indicado. Por defecto es el de –user
-user	Se conecta a la base de datos con el usuario y clave indicados
-verbose	Visualiza información sobre la ejecución de loadjava

Al igual que podemos cargar estos objetos dentro de nuestra base de datos podemos también eliminarlos a través de la utilidad **dropjava**. Esta herramienta puede eliminar clases de Java o ficheros .jar.

Carga del conjunto de librerías de Jaspa

Como ya se ha comentado, el ejecutable loadjava también nos permite cargar archivos .jar. Esta característica nos va a permitir importar todas las librerías dependientes del proyecto Jaspa. Todas ellas serán suministradas en el repositorio Joinup cuando esté disponible la versión de Jaspa para Oracle.

En un solo paso se pueden cargar múltiples librerías. En concreto ejecutamos el siguiente comando para cargar diferentes librerías:

```
loadjava -u jaspadb/password jts4jaspa.jar, jaspa-0.2.0.jar, jaspa4oracle.jar,junit-3.8.1.jar, gt-data-2.7.2.jar,gt-main-2.7.2.jar, gt-epsg-wkt-2.7.2.jar, gt-graph-2.7.2.jar, gt-api-2.7.2.jar, gt-metadata-2.7.2.jar , jsr-275-1.0-beta-2.jar ,gt-referencing-2.7.2.jar ,gt-shapefile-2.7.2.jar ,common-2.6.0.jar ,commons-collections-3.1.jar ,commons-jxpath-1.3.jar ,commons-logging-1.1.1.jar , commons-pool-1.5.4.jar ,ecore-2.6.1.jar ,gt-xsd-core-2.7.2.jar ,jai_core-1.1.3.jar , vecmath-1.3.2.jar , xercesImpl-2.7.1.jar,xml-apis-1.0.b2.jar , xml-apis-xerces-2.7.1.jar , xsd-2.6.0.jar , gt-opengis-2.7.2.jar, jdom-1.0.jar, log4j-1.2.16.jar, picocontainer-1.2.jar
```

Cargamos ciertas librerías con la opción -resolve de modo que se fuerce la compilación por Oracle:

```
loadjava -u jaspa/ya -resolve gt-epsg-wkt-2.7.2.jar, jai_core-1.1.3.jar, gt-metadata-2.7.2.jar
```

De esta manera hemos cargado y resuelto las referencias externas de las librerías relacionadas con GeoTools y la carga de los SRID (identificadores de sistemas de referencias). Esto es necesario porque en tiempo de ejecución se carga un fichero externo con los SRID válidos.

4.2. Tipos intercambio SQL-JAVA

Conocer qué tipos de Java pueden ser válidamente asignados a determinados tipos de datos SQL es básico a la hora de construir en Oracle funciones que se basen en código Java. Por ello a continuación se incluye una tabla con los tipos de intercambio entre SQL y Java a la que más tarde haremos referencia.

Tabla 7 Tipos de intercambio entre SQL y Java

SQL	Java
CHAR, VARCHAR2, LONG	oracle.sql.CHAR java.lang.String java.sql.Date java.sql.Time java.sql.Timestamp java.lang.Byte java.lang.Short java.lang.Integer java.lang.Long java.lang.Float java.lang.Double java.math.BigDecimal byte, short, int, long, float, double
DATE	oracle.sql.DATE java.sql.Date java.sql.Time java.sql.Timestamp java.lang.String
NUMBER	oracle.sql.NUMBER java.lang.Byte java.lang.Short java.lang.Integer java.lang.Long java.lang.Float java.lang.Double java.math.BigDecimal byte, short, int, long, float, double
OPAQUE	oracle.sql.OPAQUE
RAW, LONG RAW	oracle.sql.RAW byte[]
ROWID	oracle.sql.CHAR oracle.sql.ROWID java.lang.String
BFILE	oracle.sql.BFILE

SQL	Java
BLOB	oracle.sql.BLOB java.sql.Blob
CLOB	oracle.sql.CLOB java.sql.Clob
TIMESTAMP	java.sql.Date oracle.sql.DATE java.sql.Time java.sql.Timestamp oracle.sql.TIMESTAMP java.lang.String byte[]
TIMESTAMP WITH TIME ZONE	java.sql.Date oracle.sql.DATE java.sql.Time java.sql.Timestamp oracle.sql.TIMESTAMPTZ java.lang.String byte[]
TIMESTAMP WITH LOCAL TIME ZONE	java.sql.Date oracle.sql.DATE java.sql.Time java.sql.Timestamp oracle.sql.TIMESTAMPLTZ java.lang.String byte[]
Object types	oracle.sql.STRUCT java.sql.Struct java.sql.SqlData oracle.sql.ORAData
Reference types	oracle.sql.REF java.sql.Ref oracle.sql.ORAData
Nested table types and VARRAY types	oracle.sql.ARRAY java.sql.Array oracle.sql.ORAData

4.3. Indexación Espacial

Una base de datos común proporciona índices para permitir el acceso rápido a subconjuntos de datos. La indexación de tipo estándar (números, textos, fechas…) se construye sobre árboles B, B+, funciones de cálculo u otros métodos.

El orden natural de números, textos y fechas es fácil de determinar. Sin embargo los datos espaciales necesitan otro tipo de indexación para un acceso eficiente a los datos.

Los índices espaciales se basan en "Bounding Box". Un Bounding Box es el rectángulo mínimo capaz de contener una geometría dada.

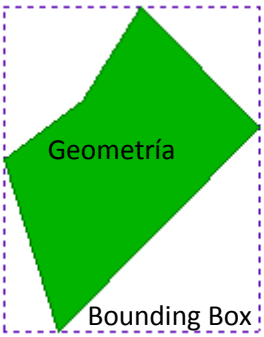


Fig. 10 Bounding Box de una figura geométrica

En el análisis de datos espaciales es habitual hacerse preguntas del tipo ¿Qué líneas coinciden con cierto polígono?

Con los índices espaciales se almacenan los rectángulos límites para poder contestar rápidamente a preguntas como ¿Qué líneas tienen rectángulos límites coincidentes con el rectángulo límite del polígono? Sobre la respuesta de los índices ya se puede comprobar las geometrías exactas y no solo sus rectángulos límites. Por ello, se dice que los índices espaciales son aproximados.

4.4. Índices R-tree

Los índices que va a utilizar Jaspa en Oracle son del tipo R-tree. Por ello, en esta sección explicaremos en detalle en qué consiste este tipo de índices.

Los índices R-Trees, o Rectangle-Tree, son estructuras de datos en forma de árbol que se utilizan para acceder a datos espaciales indexando información multidimensional. El árbol divide el espacio en cajas contenedoras o MBR (Minimum Bounding Rectangle) que representan a las geometrías que se quieren ordenar, estas MBR se organizan de forma jerárquica y con posibilidad de solapamiento.

Cada nodo del árbol contiene un número variable de entradas, hasta un máximo definido. Cada entrada que no se encuentra en un nodo hoja, contiene dos datos: un apuntador al hijo y un MBR de la unión de todas las figuras pertenecientes a sus hijos.

Los algoritmos de inserción y borrado utilizan los conjuntos límite de los nodos para asegurar que elementos cercanos están localizados en la misma hoja (en particular, un nuevo elemento será insertado en la hoja que requiera el menor aumento del conjunto límite). Cada entrada de una hoja contiene dos datos: una forma de identificar el elemento actual (que, alternativamente, podría estar directamente en el nodo) y el conjunto límite de ese elemento.

De forma similar, los algoritmos de búsqueda utilizan los conjuntos límite para decidir en qué nodo buscar. De este modo, la mayoría de los nodos del árbol nunca son examinados durante una búsqueda. Esto hace que este tipo de árboles (como los B-Tree) sean idóneos para el trabajo con bases de datos.

Se pueden utilizar distintos algoritmos para dividir nodos cuando estos crecen demasiado, resultando subtipos de R-Tree cuadráticos y lineales.

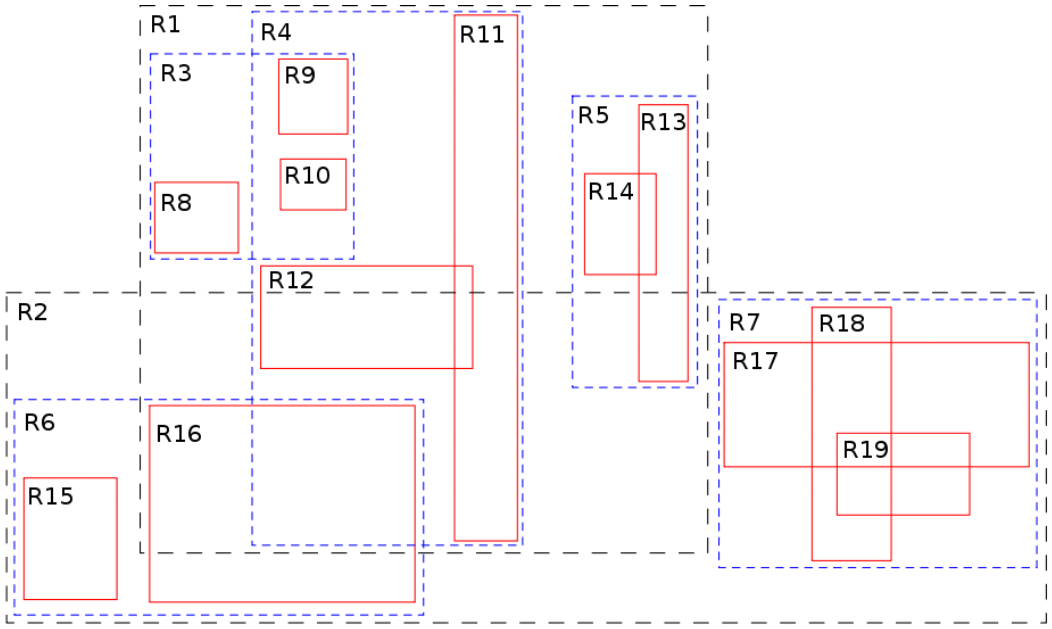


Fig. 11Representación de las cajas de un índice R-Tree

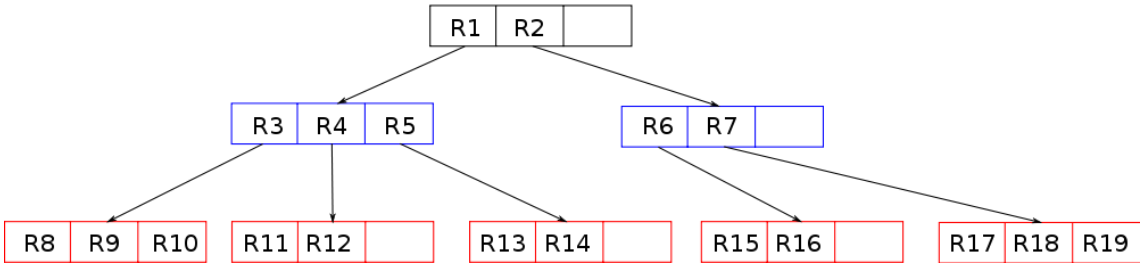


Fig. 12 Representación del árbol de un índice R-Tree

Los R-Trees no garantizan un buen rendimiento en el peor caso, pero en general se comportan bien con datos del mundo real.

4.5. Creación de índices espacial en Jaspa para Oracle

La creación del índice espacial se realiza en cuatro pasos,

1- Creación de la función orcl_box,
2- Creación de la tabla espacial, inserción de datos y prueba de la función orcl_box (Paso opcional)
3- Actualización de la vista USER_SDO_GEOM_METADATA,
4- Creación del índice espacial.

La creación de la tabla espacial e inserción de datos se debe hacer previamente a la creación de los índices. Los pasos 3 y 4 deben realizarse para cada tabla en la que queramos habilitar la indexación espacial.

1. Función orcl_box

Creamos la función orcl_box que devuelve un objeto SDO_GEOMETRY. Como vamos a crear índices espaciales basados en ella, marcamos que es de tipo “Deterministic”.

Esta función toma como argumento de entrada una geometría de tipo BLOB. Utiliza las funciones de Jaspa ST_XMin, ST_YMin, ST_XMax, ST_XMin, que se han creado anteriormente, para obtener las coordenadas X e Y mínimas y máximas de geometría de entrada. Con las cuatro coordenadas extremas construye un rectángulo de tipo SDO_Geometry de Oracle.

```
CREATE OR REPLACE FUNCTION orcl_box (geom in blob)
      return Sdo_GEOMETRY deterministic is
            v_box BLOB;
            v_xmin NUMBER;
            v_xmax NUMBER;
            v_ymin NUMBER;
            v_ymax NUMBER;
      begin
            SELECT st_box2d(geom) into v_box from dual;
            SELECT st_xmin(v_box) into v_xmin from dual;
            SELECT st_xmax(v_box) into v_xmax from dual;
            SELECT st_ymin(v_box) into v_ymin from dual;
            SELECT st_ymax(v_box) into v_ymax from dual;
            return SDO_GEOMETRY(
                  2003, -- two-dimensional polygon
                  NULL,
                  NULL,
                  SDO_ELEM_INFO_ARRAY(1,1003,3), -- one rectangle (1003
= exterior)
                  SDO_ORDINATE_ARRAY(v_xmin,v_ymin, v_xmax, v_ymax)
-- only 2 points needed to
-- define rectangle (lower left and upper right) with
-- Cartesian-coordinate data
            );
end orcl_box;
```

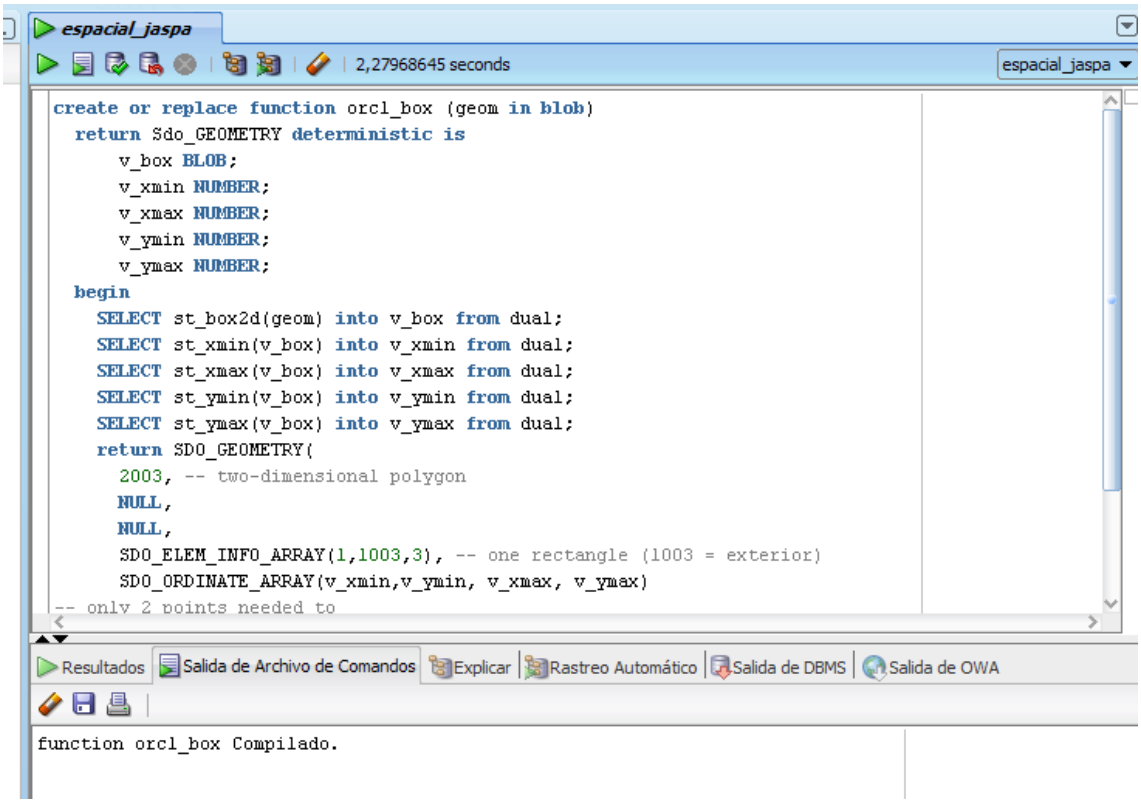
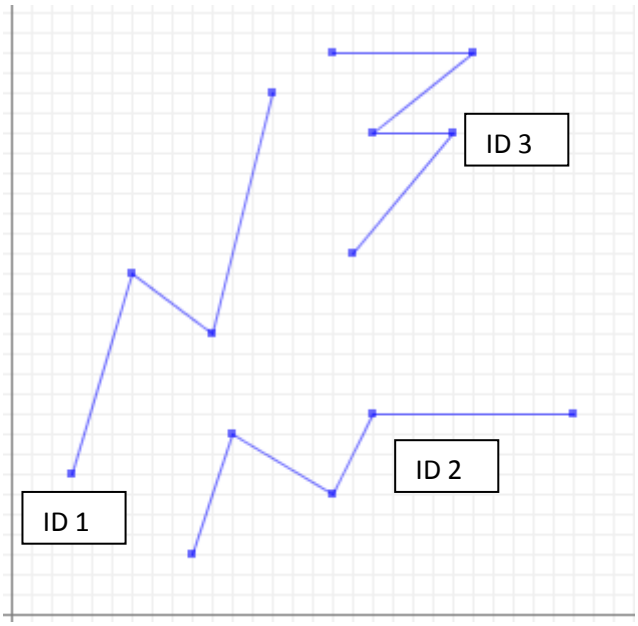


Fig. 13 Compilación de la function orcl_box

2. **Creación tabla espacial e insertar datos y prueba de la función orcl_box (Paso opcional)**

Si la tabla de datos espaciales no existe, la crearemos e insertaremos datos en ella.




```
SQL>
CREATE TABLE lineas (id NUMBER primary key, geom BLOB);
INSERT INTO lineas(id,geom) SELECT 1, ST_geomfromtext('LINESTRING(30
70, 60 170, 100 140, 130 260)') from dual;

INSERT INTO lineas(id,geom) SELECT 2, ST_geomfromtext('LINESTRING(90
30, 110 90, 160 60, 180 100, 280 100)') from dual;

INSERT INTO lineas(id,geom) SELECT 3, ST_geomfromtext('LINESTRING(160
280, 230 280, 180 240, 220 240, 170 180)') from dual;
```

Comprobamos que la función del índice orcl_box devuelve bien las coordenadas del Bounding Box de tipo SDO_Geometry. A su vez, aplicamos la función ST_Box de Jaspa para obtener el rectángulo límite de las geometrías.

```
SQL>
SELECT st_astext(st_box(geom)) FROM lineas;
```

--Resultado:

```
ST_ASTEXT(ST_BOX(GEOM))
POLYGON ((30 70, 30 260, 130 260, 130 70, 30 70))
POLYGON ((90 30, 90 100, 280 100, 280 30, 90 30))
POLYGON ((160 180, 160 280, 230 280, 230 180, 160 180))

3 rows selected
```

```
SQL>
SELECT (orcl_box (geom)) FROM lineas;
```

--Resultado:

```
(ORCL_BOX(GEOM))
MDSYS.SDO_GEOMETRY(2003,null,null,MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,3),MDSYS.SDO_ORDINATE_ARRAY(30,70,130,260))
MDSYS.SDO_GEOMETRY(2003,null,null,MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,3),MDSYS.SDO_ORDINATE_ARRAY(90,30,280,100))
MDSYS.SDO_GEOMETRY(2003,null,null,MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,3),MDSYS.SDO_ORDINATE_ARRAY(160,180,230,280))

3 rows selected
```

3. Actualiza la vista USER_SDO_GEOM_METADATA

En la columna TABLE_NAME indicamos el nombre de la tabla, en este caso “lineas”.

En la columna COLUMN_NAME indicamos qué columna contiene las geometrías. Como utilizamos un índice basado en una función, el formato a indicar es [Usuario].orcl_box([campo de geometrías]). En este ejemplo indicamos “jaspadb.orcl_box(geom)”

La columna DIMINFO (información de dimensión) se utiliza para definir por cada dimensión el límite inferior, superior y tolerancia. Se introduce una entrada por cada dimensión.

La columna SRID debe contener el valor SRID para el sistema de coordenadas de las geometrías, o NULL si no existe un sistema de coordenadas específico asociado.

La sentencia SQL con la que actualizamos la vista user_sdo_geom_metadata en este ejemplo es:

```
SQL>
INSERT INTO user_sdo_geom_metadata
      (TABLE_NAME,
      COLUMN_NAME,
      DIMINFO,
      SRID)
VALUES (
      'lineas',
      'jaspadb.orcl_box(geom)',
      SDO_DIM_ARRAY( -- 200X200 grid
      SDO_DIM_ELEMENT('X', 0, 200, 0.005),
      SDO_DIM_ELEMENT('Y', 0, 200, 0.005)
),
NULL -- SRID
);
```

4. Creación del índice espacial

El índice espacial que crearemos será de tipo R-tree, lo creamos con:

SQL>

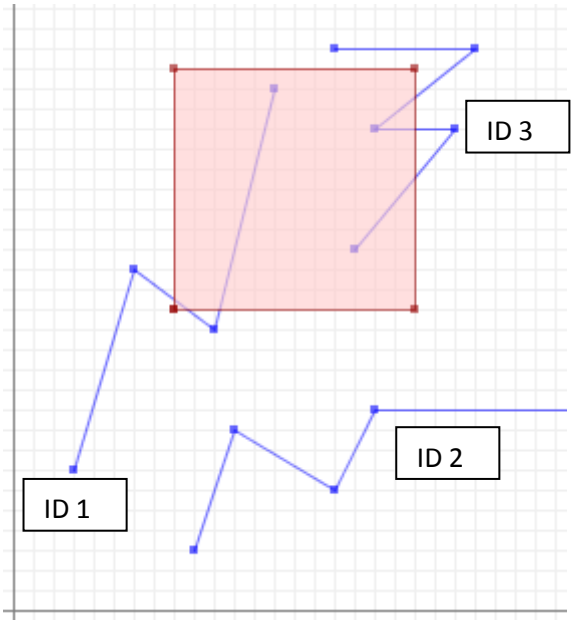
```
   create index lineas_idx on lineas(orcl_box(geom)) indextype is
mdsys.spatial_index;
```

4.6. Utilización de índices espaciales

Para la utilización de la indexación espacial en Jaspa para Oracle haremos uso de la función de Oracle **SDO_Filter**. La función SDO_Filter utiliza el índice espacial para identificar al conjunto de objetos espaciales que pueden interactuar espacialmente con un objeto determinado, o pares de objetos espaciales que pueden interactuar espacialmente entre sí. Se dice que los objetos interactúan espacialmente si no son disjuntos.

Este operador es de tipo filtrado primario, en el que se asegura que las cajas de las geometrías intersecan, aunque puede darse el caso que las propias geometrías sean disjuntas. En Oracle Spatial se utiliza como filtrado secundario el operador SDO_RELATE que sí asegura que los objetos interactúan espacialmente.

El siguiente ejemplo selecciona las geometrías que pueden interactuar con un rectángulo de coordenadas inferior izquierda 80,150 y superior derecha 200,270. Gráficamente vemos que las geometrías que interactúan son la 1 y la 3. Comprobamos como con la consulta SQL a la base de datos obtenemos el resultado previsto:

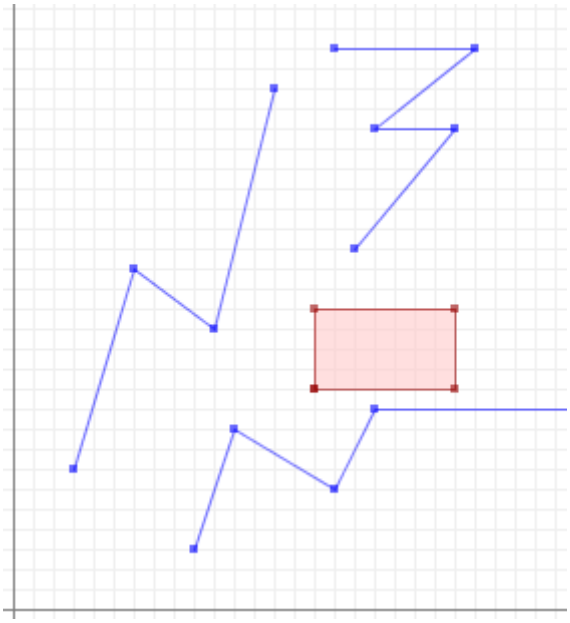



```
SQL> SELECT c.id
       FROM lineas c
       WHERE
SDO_FILTER(orcl_box(c.geom),
             SDO_GEOMETRY(2003,
NULL, NULL,

       SDO_ELEM_INFO_ARRAY(1,1003
,3),

       SDO_ORDINATE_ARRAY(80,150,
200, 270))
       ) = 'TRUE';
```

--Resultado:

```
ID
----------
1
3

2 rows selected
```

Por el contrario, en el siguiente ejemplo el rectángulo no interseca con ninguna de las geometrías. Comprobamos como con la consulta SQL obtenemos el resultado previsto:



```
SQL>  SELECT c.id
      FROM lineas c
      WHERE
SDO_FILTER(orcl_box(c.geom),
            SDO_GEOMETRY(2003,
NULL, NULL,

      SDO_ELEM_INFO_ARRAY(1,1003
,3),

      SDO_ORDINATE_ARRAY(150,110
, 220, 150))
      ) = 'TRUE';
```

--Resultado:

```
ID
----------

0 rows selected
```

Estadísticas

Aunque hemos creado el índice con éxito y utilizamos la función SDO_FILTER para el filtrado, podemos preguntarnos si realmente se está utilizando el índice para ejecutar las consultas.

Para ello debemos primero analizar la tabla sobre la que hemos creado el índice y después analizar las sentencias SQL.

Para analizar la tabla "lineas" junto con sus índices utilizamos la sentencia:

```
SQL> analyze table lineas compute statistics;
--Resultado
analyze table lineas compute statistics correcto
```

Con la sentencia "set autotrace on" se activa que se muestre el plan de ejecución y las estadísticas de las siguientes sentencias SQL que ejecutemos:

```
SQL> set autotrace on
--Resultado
Seguimiento Automático Activado
```

A continuación se muestra el plan de ejecución, así como las estadísticas de las consultas SQL. En el primer ejemplo se emplea la función SDO_FILTER y en segundo no.

- Primer ejemplo:

```
SQL> SELECT c.id
     FROM lineas c
     WHERE SDO_FILTER(orcl_box(c.geom),
           SDO_GEOMETRY(2003, NULL, NULL,
           SDO_ELEM_INFO_ARRAY(1,1003,3),
           SDO_ORDINATE_ARRAY(0,0, 10,10))
     ) = 'TRUE';

--Resultado
Plan hash value: 1199156059
 ----------------------------------------------------------------------------------------
| Id  | Operation                   | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |           |     1 |    90 |     0   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| LINEAS    |     1 |    90 |     0   (0)| 00:00:01 |
|*  2 |   DOMAIN INDEX              | LINEAS_IDX |       |       |     0   (0)| 00:00:01 |
----------------------------------------------------------------------------------------

[…]
```

- Segundo ejemplo:

```
SQL> SELECT c.id
     FROM lineas c;

--Resultado
```

Plan hash value: 4248594370

```
-------------------------------------------------------------------------------
| Id  | Operation        | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |             |     3 |     9 |     1   (0)| 00:00:01 |
|   1 |  INDEX FULL SCAN | SYS_C0010760 |     3 |     9 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

[…]

Comprobamos como consultando la tabla con la función SDO_FILTER se accede a ella por medio del índice (TABLE ACCESS BY INDEX ROWID), mientras que en el segundo caso se recorre toda la tabla (INDEX FULL SCAN).

4.7. Creación de funciones

Una función es un conjunto de instrucciones en PL/SQL, que puede ser llamado usando el nombre con el que se haya creado. Se diferencian de los procedimientos, en que las funciones retornan un valor desde donde fueron llamadas.

Fig. 14 Proceso de creación de una función Java en Oracle

Los pasos para almacenar una función Java en Oracle son: codificación de una clase Java con el método deseado, carga del método Java en el SGBD y resolver las referencias externas, publicación de procedimientos almacenados o funciones y por último invocar al procedimiento o función creados.

Para invocar a las funciones, el usuario debe tener asignados ciertos permisos. En nuestro caso al crear el usuario "jaspadb" en el capítulo "3.6 Creación de un nuevo usuario", ya indicamos los permisos necesarios para crear funciones y procedimientos.

1. **Codificación de una clase Java con el método deseado.**

La funcionalidad descrita en los apartados "1.11 Operadores espaciales" y "1.12 Funciones espaciales" se recopila en el paquete de Java es.upv.jaspa.oracle. En dicha paquete se han definido cuatro clases.

La clase JP contiene los métodos que trabajan directamente con geometrías JTS. Por su parte, en la clase SQL los métodos utilizan datos de tipo binario que son susceptibles de mapearse a la base de datos. En la clase SQLArrays se definen las funciones que toman como argumento arrays de geometrías, y en SQLEx aquellos métodos que se conectan con la base de datos para añadir datos en las tablas de metadatos.

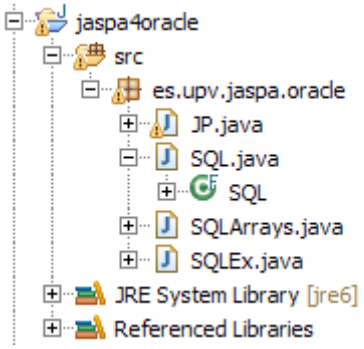


Fig. 15 Proyecto jaspa4oracle en Eclipse

2. Carga de métodos Java en el SGBD y resolución de referencias externas.

Tal y como indicamos en la sección "*4.1. Carga de librerías necesarias. Loadjava*", la utilidad loadjava permite agregar a la base de datos fuentes (.java), clases (.class), ficheros (.jar y .zip), ficheros de recursos Java (.res) y ficheros SQLJ (.sqlj).

Se debe especificar el nombre de usuario y la contraseña. Por defecto las clases o ficheros se cargan en el esquema del usuario indicado.

3. Publicación de procedimientos almacenados o funciones

Antes de poder utilizar directamente un procedimiento almacenado o función Java desde SQL, hay que registrar las referencias a este procedimiento en el diccionario de datos. Esta operación no se realiza de forma automática, ya que el motor de Oracle no puede saber qué métodos serán accesibles desde SQL.

Para cada método Java al que queramos acceder, es necesario crear una función o un procedimiento PL/SQL mediante las instrucciones CREATE FUNCTION y CREATE PROCEDURE.

El cuerpo de estas funciones y procedimientos contendrá la cláusula LANGUAGE JAVA para registrar el nombre completo del método, el valor de retorno y los parámetros.

La correspondencia de tipos entre SQL y Java podemos encontrarla en el capítulo "4.2. Tipos intercambio SQL-JAVA".

La sintaxis para crear una función es la siguiente:

CREATE [OR REPLACE] FUNCTION name [(param [IN] datatype) . . .]
RETURN datatype
[IS|AS] pl/sql_subprogram

Cuando se libere la versión de Jaspa para Oracle, se facilitará un fichero SQL (Jaspa4Oracle.sql) con las sentencias para crear las diferentes funciones, procedimientos, funciones que utilicen arrays de geometrías, agregados y tablas de metadatos en Oracle.

```
88
89      --=============================================================================
90      --=============================================================================
91      --======                                                               ======
92      --======                                    JASPA GEOMETRY CONSTRUCTORS ======
93      --======                                                               ======
94      --=============================================================================
95      --=============================================================================
96
97    create or replace FUNCTION ST_BdPolyFromText (
98          geom in VARCHAR2,
99          SRID in NUMBER)
100       RETURN RAW AS LANGUAGE JAVA Name
101     'es.upv.jaspa.oracle.SQL.ST_BdPolyFromText(java.lang.String,int) return java.lang.byte[]';
102   /
103   create or replace FUNCTION ST_BdMPolyFromText (
104         geom in VARCHAR2,
105         SRID in NUMBER)
106      RETURN RAW AS LANGUAGE JAVA Name
107    'es.upv.jaspa.oracle.SQL.ST_BdMPolyFromText(java.lang.String,int) return java.lang.byte[]';
108   /
109 create or replace FUNCTION ST_GeomCollFromText (
110         geom in VARCHAR2)
111      RETURN RAW AS LANGUAGE JAVA Name
112    'es.upv.jaspa.oracle.SQL.ST_GeomCollFromText(java.lang.String) return java.lang.byte[]';
113   /
114   create or replace FUNCTION ST_GeomCollFromText2 (
115         geom in VARCHAR2,
116         SRID in NUMBER)
117      RETURN RAW AS LANGUAGE JAVA Name
118    'es.upv.jaspa.oracle.SQL.ST_GeomCollFromText(java.lang.String,int) return java.lang.byte[]';
119   /
```

Fig. 16 Fichero SQL Jaspa4Oracle para añadir la funcionalidad de Jaspa a Oracle

4. Invocar al procedimiento, bien desde SQL o desde PL/SQL.

Una vez cargados y publicados los procedimientos almacenados escritos en Java ya pueden utilizarse, ya sean con sentencias SQL DML, bloques PL/SQL, o con subprogramas PL/SQL.

4.8. Ejemplo. Creación de la función ST_Area.

Esta función tiene como objetivo la devolución del área de un polígono o multipolígono. Seguiremos los pasos explicados en el apartado anterior.

Paso 1. Codificar una clase Java con el método deseado.

Los métodos se deben definir como static.

La función ST_Area de la clase SQL llama a la función homónima de la clase JP. La clase JP finalmente realiza los cálculos. Se definen en el paquete es.upv.jaspa.oracle.

- Método ST_Area en la clase es.upv.jaspa.oracle.SQL

```
public static Double ST_Area(Blob ablob) throws JASPAException,
                SQLException {

        byte[] sgeom = getBytesFromBlob(ablob);
        Geometry geom = Core.getJTSGeometryFromGBLOB(sgeom);
        return JP.ST_Area(geom);
}
```

- Método ST_Area en la clase es.upv.jaspa.oracle.JP

```
public static Double ST_Area(Geometry geom) {

        if (Core.isNullGeometry(geom))
                return null;

        return geom.getArea();
}
```

Una vez codificados los métodos en Java, se exporta el proyecto con el nombre jaspa4Oracle.jar.

Paso 2. Cargar el método Java en el SGBD y resolver las referencias externas.

Para esto utilizamos la utilidad loadjava explicada anteriormente.

Por ejemplo,
Carga el fichero jaspa4Oracle.jar con el usuario jaspadb con contraseña “password”.

```
loadjava -u jaspadb/password -resolve jaspa4Oracle.jar
```

Paso 3. Hacer público el método Java en el SGBD

La publicación construye una función o procedimiento PL/SQL asociándolo con el código Java. El tipo de los parámetros (y resultado) de la especificación Java debe estar calificado por completo.

```
SQL>
create or replace FUNCTION ST_Area (
            geometry in BLOB)
      RETURN NUMBER AS LANGUAGE JAVA Name
'es.upv.jaspa.oracle.SQL.ST_Area(java.sql.Blob) return
java.lang.Double';
```

Paso 4. Invocar al procedimiento, bien desde SQL o desde PL/SQL.

Esta operación puede hacerse aplicando directamente la función o dentro de un procedimiento PL/SQL.

- Directamente la función:

```
--Obención del área de un Poligono 2D
SQL>
SELECT ST_Area(ST_GeomFromText('MULTIPOLYGON (((0 0, 0 10, 10 10, 10
0, 0 0),(2 2, 2 4, 4 4, 4 2, 2 2)))',-1)) from dual;
-- Resultado:
96.0

-- Obención del área de un Polígono 3D
SQL>
SELECT ST_Area(ST_GeomFromEWKT('SRID=-1;MULTIPOLYGON (((0 0 5, 0 10 8,
10 10 9, 10 0 2, 0 0 5),(2 2 1, 2 4 1, 4 4 1, 4 2 1, 2 2 1)))')) from
dual;
-- Resultado:
96.0
```

- Dentro de un procedimiento PL/SQL:

Para activar la salida de los valores por pantalla debemos ejecutar “set serveroutput on”.

```
SQL>
DECLARE
      RET_VAL NUMBER;
      BEGIN
            RET_VAL := ST_Area(ST_GeomFromText('POLYGON ((0 0, 0 10, 10
10, 10 0, 0 0))'));

            dbms_output.put_line(to_char(ret_val));
END;
```

4.9. Tipos de parámetros

Un procedimiento almacenado o función se pueden crear sin ningún parámetro, con parámetros In, parámetros Out o parámetros In/Out. Pueden existir diversos parámetros por procedimiento almacenado o función. Cada parámetro de la función Java se debe corresponder con uno de PL/SQL, estas correspondencias las indicamos en la sección "4.2. Tipos intercambio SQL-JAVA".

Si la función Java no tiene parámetros, ni siquiera es necesario indicar la lista vacía en la declaración PL/SQL.

Parámetro IN: especifica que el parámetro es de entrada y que por tanto dicho parámetro debe tener un valor en el momento de llamar a la función o procedimiento. Si no se especifica nada, los parámetros son por defecto de tipo entrada.

Parámetro OUT: especifica que se trata de un parámetro de salida.

Parámetro IN/OUT: Son parámetros de entrada y salida a la vez.

Boolean

El tipo de datos boolean de Java no tiene correspondiente en PL/SQL. Por ello, las funciones de Jaspa que devuelven este tipo de datos se ha mapeado en Oracle al tipo NUMBER. Esto será una diferencia a tener en cuenta respecto a otras bases de datos soportadas.

Ejemplo:

-Creación de la función ST_Contains

```
SQL>
create or replace FUNCTION ST_Contains (
            geom1 in BLOB,
            geom2 in BLOB)
      RETURN NUMBER AS LANGUAGE JAVA Name
 'es.upv.jaspa.oracle.SQL.ST_Contains(java.sql.Blob,java.sql.Blob)
return boolean';
 /
```

-Consulta si una línea contiene a un punto

```
SQL>
SELECT ST_Contains(Line,Point) from
(SELECT ST_GeomFromText('LINESTRING (0 0, 10 10, 10 5, 15 10)') as
Line, ST_GeomFromText('POINT (5 5)') as Point from dual)
--Resultado:
ST_CONTAINS(LINE,POINT)
----------------------
1
1 rows selected
```

4.10. Tipo de datos Blob

En Jaspa para PostgreSQL o para H2 las funciones que toman como argumento una geometría utilizan en Java el tipo byte[]

```
public static Double ST_Area(byte sgeom[])
                throws JASPAException {

        Geometry geom = Core.getJTSGeometryFromGBLOB(sgeom);
        return JP.ST_Area(geom);
}
```

Posteriormente, en la base de datos PostgreSQL para publicar la función se mapea de byte[] a tipo SQL bytea.

```
SQL>
  CREATE OR REPLACE FUNCTION Area (
          bytea
    )
    RETURNS double precision
    AS 'es.upv.jaspa.SQL.ST_Area'
    LANGUAGE JAVAU IMMUTABLE STRICT;
```

Oracle	Java
RAW, LONG RAW	oracle.sql.RAW byte[]
BLOB	oracle.sql.BLOB java.sql.Blob

Tabla 8 Mappings de Blob y byte[]

En la BD Oracle, se podría mapear el tipo byte[] a Raw. Sin embargo, se desaconseja el uso de este tipo de datos por tener una limitación de tamaño incompatible con el almacenamiento de información geográfica. Además si utilizásemos este tipo de dato, solo podríamos almacenarlo en una columna por tabla, o lo que es lo mismo, una sola geometría por tabla, siendo esto contrario a los estándares y a la arquitectura de Jaspa para otras bases de datos.

Como alternativa a ello, creamos una función en Java (getBytesFromBlob) que obtenga datos Blob a partir de byte[]. Así a la hora de crear las funciones PL/SQL en Oracle se mapea directamente de java.sql.Blob (Java) a BLOB (Java) tal y como muestra la Tabla 8 Mappings de Blob y byte[].

4.11. Alias

En Oracle los seudónimos o alias se crean utilizando la sentencia `CREATE SYNONYM`. Con dicha sentencia, se pueden crear nombres alternativos para tablas, vistas, secuencias, procedimientos, funciones, paquetes, vistas materializadas, clases Java, objetos definidos por el usuario u otros.

Los sinónimos proporcionan independencia de datos y transparencia de ubicación. Permiten que las aplicaciones funcionen sin modificaciones, independientemente de qué usuario o base de datos posea la tabla o vista. Sin embargo, los sinónimos no son un sustituto de los privilegios en los objetos de base de datos. En cualquier caso, se deben otorgar los privilegios apropiados para que un usuario pueda utilizar el sinónimo.

Con la opción `PUBLIC` se especifica que se cree un sinónimo público. La ventaja es que los sinónimos públicos son accesibles a todos los usuarios. En cualquier caso, cada usuario debe tener privilegios adecuados en el objeto subyacente con el fin de utilizar el sinónimo.

En este proyecto se van a utilizar los alias en dos casos. El primero es para crear sinónimos de funciones. En las especificaciones OGC e ISO se definen diferentes alias, por ejemplo ST_MakePoint y ST_Point. Este tipo de funciones sinónimas se tendrán que definir en el proyecto.

Además, en general, definiremos las funciones con y sin los prefijos "ST_", por ejemplo la función Box2d será creada como sinónima de ST_Box2d. La sintaxis SQL para este último ejemplo es:

```
SQL>
CREATE SYNONYM Box2d FOR ST_box2d;
```

El segundo caso de utilización de sinónimos, será para crear un sinónimo PUBLIC del paquete que contiene la funcionalidad de Jaspa y que pueda ser usada por otros usuarios. En este caso la sintaxis empleada es:

```
SQL>
CREATE or replace PUBLIC SYNONYM jaspa FOR jaspa;
```

Podemos ver gráficamente los sinónimos creados en SQL Developer, como se muestra en la siguiente imagen:

Fig. 17 Vista de sinónimos en SQL Developer

4.12. Paquetes

En Jaspa existen varias funciones sobrecargadas, para poder utilizarlas sin necesidad de renombrarlas necesitamos incorporarlas en un paquete de Oracle. Además los paquetes tienen otra serie de ventajas entre las que podemos citar:

Permite modularizar el diseño de nuestra aplicación
El uso de Paquetes permite encapsular elementos relacionados entre sí (tipos, variables, procedimientos, funciones) en un único módulo PL/Sql que llevará un nombre que identifique la funcionalidad del conjunto.

Otorga flexibilidad al momento de diseñar la aplicación
En el momento de diseñar una aplicación todo lo que necesitaremos inicialmente es la información de interfaz en la especificación del paquete. Puede codificarse y compilarse la especificación sin su cuerpo para posibilitar que otros sub-programas que referencian a estos elementos declarados puedan compilarse sin errores.

Permite ocultar los detalles de implementación
Pueden especificarse qué tipos, variables y sub-programas dentro del paquete son públicos (visibles y accesibles por otras aplicaciones y sub-programas fuera del paquete) o privados (ocultos e inaccesibles fuera del paquete).

Por ejemplo, dentro del paquete pueden existir procedimientos y funciones que serán invocados por otros programas, así como también otras rutinas de uso interno del paquete que no tendrán posibilidad de ser accedidas fuera del mismo. Esto asegura que cualquier cambio en la definición de estas rutinas internas afectará sólo al paquete donde se encuentran, simplificando el mantenimiento y protegiendo la integridad del conjunto.

Agrega mayor funcionalidad a nuestro desarrollo
Las definiciones públicas de tipos, variables y cursores hechas en la especificación de un paquete persisten a lo largo de una sesión. Por lo tanto pueden ser compartidas por todos los sub-programas y/o paquetes que se ejecutan en ese entorno durante esa sesión. Puede utilizarse para por ejemplo mantener tipos y variables globales a todo el sistema.

Introduce mejoras al rendimiento
En relación a su ejecución, cuando un procedimiento o función que está definido dentro de un paquete es llamado por primera vez, todo el paquete se integra en memoria. Por lo tanto, posteriores llamadas al mismo u otros sub-programas dentro de ese paquete realizarán un acceso a memoria en lugar de a disco. Esto no sucede con procedimientos y funciones estándares.

Sobrecarga de funciones (Overloading)
PL/Sql nos permite que varios procedimientos o funciones almacenadas, declaradas dentro de un mismo paquete, tengan el mismo nombre. Esto nos es muy útil cuando necesitamos que los sub-programas puedan aceptar parámetros que contengan diferentes tipos de datos en

diferentes instancias. En este caso Oracle ejecutará la "versión" de la función o procedimiento cuyo encabezado se corresponda con la lista de parámetros recibidos.

La propiedad de sobrecarga de funciones nos obliga a definir un paquete que llamaremos "Jaspa" debido a que en el proyecto existen varios métodos sobrecargados. En la siguiente sección veremos cómo crear un paquete y como se definen las funciones sobrecargadas.

Ejemplo: paquete Jaspa

A continuación se va a mostrar el código para crear un paquete de nombre Jaspa con la función ST_MakePoint sobrecargada. En el primer caso se va a definir en un solo paso, y en el segundo definiremos el paquete por separado en cabecera y cuerpo:

- **Caso 1, un solo paso.**

```
SQL>
CREATE OR REPLACE PACKAGE jaspa AS

FUNCTION ST_MakePoint (
            X in NUMBER,
            Y in NUMBER
      )
      RETURN RAW AS LANGUAGE JAVA Name
 'es.upv.jaspa.oracle.SQL.ST_MakePoint(double,double) return
java.lang.byte[]';

FUNCTION ST_MakePoint (
            X in NUMBER,
            Y in NUMBER,
            Z in NUMBER
      )
      RETURN RAW AS LANGUAGE JAVA Name
 'es.upv.jaspa.oracle.SQL.ST_MakePoint(double,double,double) return
java.lang.byte[]';

END;
```

- **Caso 2, cabecera y cuerpo:**

```
SQL>
CREATE OR REPLACE PACKAGE jaspa AS

FUNCTION ST_MakePoint (X NUMBER,Y NUMBER)
      RETURN RAW;

FUNCTION ST_MakePoint ( X NUMBER, Y NUMBER, Z NUMBER)
      RETURN RAW;

END jaspa;

/
```

```
CREATE OR REPLACE PACKAGE BODY jaspa AS
FUNCTION ST_MakePoint (
            X in NUMBER,
            Y in NUMBER
      )
      RETURN RAW AS LANGUAGE JAVA Name
 'es.upv.jaspa.oracle.SQL.ST_MakePoint(double,double) return
java.lang.byte[]';

FUNCTION ST_MakePoint (
            X in NUMBER,
            Y in NUMBER,
            Z in NUMBER
      )
      RETURN RAW AS LANGUAGE JAVA Name
 'es.upv.jaspa.oracle.SQL.ST_MakePoint(double,double,double) return
java.lang.byte[]';
END jaspa;
```

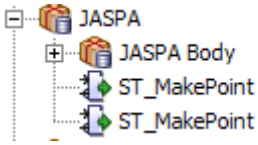


Fig. 18 Vista del paquete creado en SQL Developer

Utilización de las funciones definidas en los paquetes

Para invocar a funciones definidas en paquetes simplemente debemos cualificar el nombre de la función insertando previamente el nombre del paquete.

Ejemplo:

```
SQL>
select jaspa.st_makepoint(2,3,2) from dual;

--Resultado:
JASPA.ST_MAKEPOINT(2,3,2)
--------------------------------------------------------------
092001010000000000000040000000000000084000000000000040

1 rows selected


SQL>
select jaspa.st_makepoint(2,3) from dual

--Resultado:
JASPA.ST_MAKEPOINT(2,3)
----------------------------------------------------------------
09200001000000000000004000000000000840

1 rows selected
```

4.13. Agregados de geometrías

Las funciones agregadas permiten realizar operaciones sobre un conjunto de filas. Oracle, como otras bases de datos, define una serie de agregados como MAX, MIN, SUM que se aplican a datos numéricos.

Oracle permite la definición de nuevas funciones agregadas que se apliquen incluso a datos complejos como LOB. Para ello las funciones agregadas definidas por el usuario implementan la interfaz ODCIAggregate.

La interfaz de ODCIAggregate: Descripción general

Las funciones agregadas definidas por el usuario se definen gracias al modelo Oracle Data Cartridge que emplea tipos de objetos y otras características de extensibilidad de la base de datos.

Existen otras maneras de crear funciones agregadas definidas por el usuario, como utilizando Java o C, pero hemos optado por programarlo en PL/SQL. La creación de un nuevo operador de agregación se hace en dos pasos:

- Creación de un tipo de objeto que implemente la interfaz ODCIAggregate
- Creación de una función PL/SQL, indicando que se empleará para agregados y enlazándola con el tipo de objeto anterior.

El tipo de objeto debe implementar obligatoriamente cuatro funciones llamadas ODCIAggregateInitialize, ODCIAggregateIterate, ODCIAggregateTerminate y ODCIAggregateMerge.

La función de **inicialización** se llama por el motor de ejecución de SQL antes de procesar un grupo de registros. Es donde se hacen las inicializaciones. Por ejemplo si se van a concatenar cadenas de texto, en este paso se prepararía una cadena vacía.

La función de **iteración** se llama para cada registro en el grupo. Tiene el valor real de la columna o expresión que se agrega como parámetro de entrada. El tipo de datos que devuelve la función de terminación debe ser el mismo que el input de la función de iteración.

La función de **terminación** se llama por la base de datos para recuperar el resultado de agregación del conjunto.

La función de **fusión** se llama cuando se han realizado agregaciones en paralelo y se deben combinar los resultados.

La función de PL/SQL en sí es muy simple. Se declara como una función de agregado y se define el tipo de objeto que implementa sus reglas.

La mejor manera de visualizar una función de agregado definida por un usuario es con un ejemplo. En la siguiente sección vamos a ver una función de agregado que construye una geometría de tipo colección a partir de otras simples. La función se llama "ST_Collect".

Ejemplo. ST_Collect

En este ejemplo se crea la función ST_Collect que sirve para devolver una geometría a partir de una colección de otras geometrías.

Creación de un tipo de objeto que implemente la interfaz ODCIAggregate. Primero creamos la definición del tipo `t_ST_Collectagg` y posteriormente establecemos el cuerpo del tipo.

Paso 1. Definición del tipo t_ST_Collectagg

```
CREATE OR REPLACE TYPE t_ST_Collectagg AS OBJECT
(
  L_IN Blobarray ,
  i NUMBER,

  STATIC FUNCTION ODCIAggregateInitialize(
sctx  IN OUT  t_ST_Collectagg)
    RETURN NUMBER,

  MEMBER FUNCTION ODCIAggregateIterate(
self   IN OUT  t_ST_Collectagg,
value  IN  BLOB )
     RETURN NUMBER,

  MEMBER FUNCTION ODCIAggregateTerminate(
self  IN t_ST_Collectagg,
returnValue  OUT  RAW,
                              flags        IN   NUMBER)
    RETURN NUMBER,

  MEMBER FUNCTION ODCIAggregateMerge(
self  IN OUT  t_ST_Collectagg,
                              ctx2  IN      t_ST_Collectagg)
    RETURN NUMBER
);
/
SHOW ERRORS
```

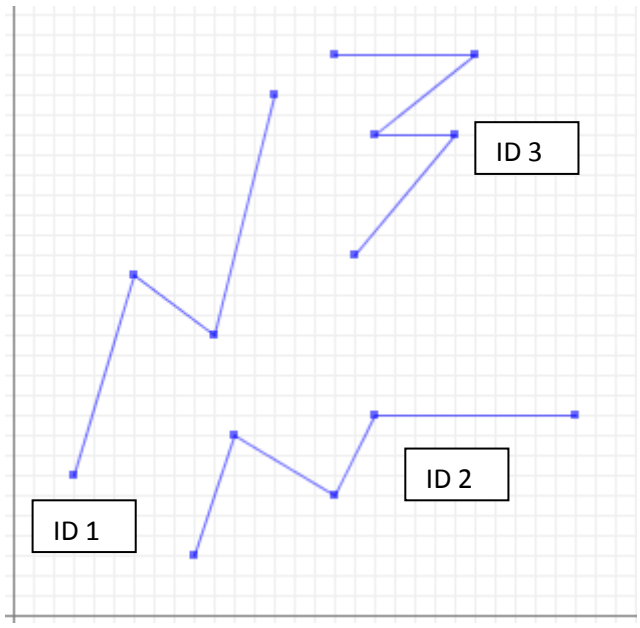
Paso 2. Definición del cuerpo t_ST_Collectagg

```
CREATE OR REPLACE TYPE BODY t_ST_Collectagg IS
  STATIC FUNCTION ODCIAggregateInitialize(sctx  IN OUT
t_ST_Collectagg)
    RETURN NUMBER IS
  BEGIN
    sctx := t_ST_Collectagg(Blobarray (),0);

    RETURN ODCIConst.Success;
  END;

  MEMBER FUNCTION ODCIAggregateIterate(self   IN OUT  t_ST_Collectagg,
                                       value  IN      BLOB )
    RETURN NUMBER IS

  BEGIN
    self.i:=self.i+1;
    SELF.L_IN.EXTEND;
    SELF.L_IN(self.i):=value;
    RETURN ODCIConst.Success;
  END;

  MEMBER FUNCTION ODCIAggregateTerminate(self         IN
t_ST_Collectagg,
                                         returnValue  OUT  RAW,
                                         flags        IN   NUMBER)
    RETURN NUMBER IS
  BEGIN

    returnValue := (st_collect_arr(self.l_in));
    RETURN ODCIConst.Success;
  END;

  MEMBER FUNCTION ODCIAggregateMerge(self  IN OUT  t_ST_Collectagg,
                                     ctx2  IN      t_ST_Collectagg)
    RETURN NUMBER IS
  BEGIN
    RETURN ODCIConst.Success;
  END;
END;
```

Paso 3. Creación de una función PL/SQL, indicando que se empleará para agregados enlazándola con el tipo de objeto anterior.

```
CREATE OR REPLACE FUNCTION ST_Collectagg (p_input BLOB)
RETURN RAW
PARALLEL_ENABLE AGGREGATE USING t_ST_Collectagg;
```

A continuación probaremos la función agregada que acabamos de definir. Para ello utilizaremos una tabla con 3 líneas que ya utilizamos para probar la indexación espacial.



Recordemos previamente los datos que contenía:

```
SQL>
select id,st_astext(geom) from lineas;
```

ID	ST_ASTEXT(GEOM)
1	LINESTRING (30 70, 60 170, 100 140, 130 260)
2	LINESTRING (90 30, 110 90, 160 60, 180 100, 280 100)
3	LINESTRING (160 280, 230 280, 180 240, 220 240, 170 180)

4 rows selected

Aplicamos la función agregada sobre el conjunto de registros de la tabla:

```
SQL>
SELECT st_astext(ST_Collectagg(geom)) AS geom
FROM   lines;

--Resultado:
```

GEOM
MULTILINESTRING ((30 70, 60 170, 100 140, 130 260), (90 30, 110 90, 160 60, 180 100, 280 100), (160 280, 230 280, 180 240, 220 240, 170 180))

1 rows selected

Otra manera de hacer Agregados

Una forma diferente de hacer agregados espaciales es la propuesta por Greener, 2007. En ella se concatenan líneas por medio de la clausula GROUP BY. Este método se muestra de modo anecdótico, ya que la solución adoptada para crear agregados es la explicada en el apartado anterior que utiliza la interfaz ODCIAggregate.

En el siguiente ejemplo se crea una función llamada "concatLines" que va aplicando la función ST_Union(A,B) para concatenar las líneas por pares.

```
Create Or Replace Type GeometrySet Is Table Of raw(2000);
Grant execute on GeometrySet to public;
/
create or replace Function concatLines (p_lines IN geometrySet)
  Return RAW
Is
  v_geometry raw(2000);
Begin
  IF ( p_lines is null ) THEN
      Return NULL;
  END IF;
  IF ( p_lines.COUNT = 1 ) THEN
      Return p_lines(1);
  Else
    v_geometry := p_lines(1);
    FOR i IN (p_lines.FIRST+1) .. p_lines.LAST LOOP
       select  st_union(v_geometry,p_lines(i))  into  v_geometry  from
dual;
    END LOOP;
    Return v_geometry;
  End If;
End concatLines;
/
SHOW ERRORS
```

Paso 1. Ejemplo de aplicación de la función de concatenación:

```
select      id,      st_astext(concatLines(CAST(COLLECT(a.GEOM)      as
geometrySet))) as aggregatedLines
  from (select 1 as id, st_geomfromtext('LINESTRING (0 0 10 8, 0 50 10
18)') as geom from dual
     union all
     select 1 as id, st_geomfromtext('LINESTRING (0 50 10 18, 10 70 10
18)') as geom from dual
    union all
     select 2 as id, st_geomfromtext('LINESTRING (0 0, 5 5)') as geom
from dual) a
group by id;
```

--Resultado:

```
ID                AGGREGATEDLINES
-- ------------------------------------------------------------------
1  MULTILINESTRING ((0 0 10 8, 0 50 10 18), (0 50 10 18, 10 70 10 18))
2  LINESTRING (0 0, 5 5)
```

4.14. Array de geometrías

La creación de funciones que tomen como argumento Arrays de geometrías no es un proceso directo como pudiera parecer.

Básicamente, se necesita crear un proceso que reciba una lista indefinida de valores. Oracle no soporta directamente tipos array y para poder utilizarlo debemos definir un nuevo tipo:

```
SQL>
create or replace TYPE BLOBARRAY AS TABLE OF raw(200);
```

Una vez definido este tipo de datos, podemos utilizarlo para crear una función que lo utilice como entrada. Por ejemplo:

```
SQL>
create or replace FUNCTION ST_MakeGeomColl (P_IN IN BLOBARRAY)
      RETURN RAW AS LANGUAGE JAVA Name
 'original.SQLArrays.ST_MakeGeomColl(oracle.sql.ARRAY)            return
java.lang.byte[]';
```

La base de datos Oracle permite mapear los tipos tabla anidada de SQL a oracle.sql.Array, java.sql.Array u oracle.sql.ORAData.

En este caso definimos un método Java que tome como argumento un Array del tipo oracle.sql.Array.

```
public static byte[] ST_MakeGeomColl(oracle.sql.ARRAY p_in)
                  throws java.sql.SQLException, IOException,
JASPAException {

            Object[] values = (Object[]) p_in.getArray();
            byte[] mybyte = ST_MakeGeomColl(values);

            return mybyte;

      }
```

En el método anterior se extraen los valores del Array y se obtiene un array de objetos (Object[]) que ya se puede pasar a las funciones de Jaspa de Arrays programadas para la base de datos H2.

Con esto ya tendremos definido el tipo de datos, la función en SQL y el código Java que ejecuta las funciones.

Ejemplo 1. MakeGeomColl

La función ST_MakeGeomColl construye un objeto del tipo GeometryCollection a partir de un conjunto de geometrías. En primer lugar se crean los **métodos Java** necesarios:

```
package es.upv.jaspa.oracle;

import es.upv.jaspa.exceptions.JASPAException;


import java.io.IOException;
import java.util.List;

public class SQLArrays {

	public static byte[][] getSgeomArrayFromBlob(Object[] ablob) {

		if (ablob == null)
			return null;

		int size = ablob.length;
		byte[][] res = new byte[size][];

		for (int i = 0; i < size; i++) {
			res[i] = (byte[]) ablob[i];
		}

		return res;

	}

	public static byte[] ST_MakeGeomColl(Object[] sgeomArray)
			throws JASPAException {

		return
SQL.ST_MakeGeomColl(getSgeomArrayFromBlob(sgeomArray));
	}

	public static byte[] ST_MakeGeomColl(oracle.sql.ARRAY p_in)
			throws java.sql.SQLException, IOException,
JASPAException {

		Object[] values = (Object[]) p_in.getArray();
		byte[] mybyte = ST_MakeGeomColl(values);

		return mybyte;

	}

}
```

En segundo lugar creamos la **función en Oracle** (debemos haber importado previamente el fichero que contiene los métodos previos):

SQL>

```
create or replace TYPE BLOBARRAY AS TABLE OF raw(200);
/
create or replace FUNCTION ST_MakeGeomColl (P_IN IN BLOBARRAY)
      RETURN RAW AS LANGUAGE JAVA Name
 'original.SQLArrays.ST_MakeGeomColl(oracle.sql.ARRAY) return
java.lang.byte[]';
```

A continuación, en el siguiente **ejemplo** utilizaremos la función ST_MakeGeomColl que toma un Array de geometrías como argumento de entrada. Creamos un procedimiento almacenado anónimo en el que se generan 5 puntos.

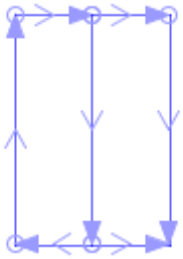
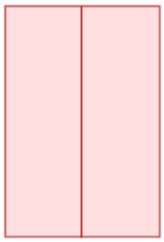
```
SQL>

SET serveroutput on size 1000000;
/
DECLARE
   L_IN Blobarray := Blobarray ();
   v_out blob;
   v_out_text VARCHAR2(200);

BEGIN
   FOR I IN 1 .. 5
   LOOP
      L_IN.EXTEND;
      select st_makepoint(i,i) into l_in(i) from dual;
   END LOOP;

    select st_astext(ST_MakeGeomColl(L_IN)) into v_out_text from dual;
   DBMS_OUTPUT.PUT_LINE (v_out_text);
END;
/
```

--Resultado

```
anonymous block completed
GEOMETRYCOLLECTION (POINT (1 1), POINT (2 2), POINT (3 3), POINT (4
4), POINT (5 5))
```

Ejemplo 2. ST_Polygonize

La función ST_Polygonize crea una nueva geometría de tipo GeometryCollection que contiene los posibles polígonos formado a partir de las líneas que se facilitan como conjunto de geometrías de entrada.

	
GeomA	ST_Polygonize(GeomA)

Ejemplo de utilización:

```
SQL>

SET serveroutput on size 1000000;
/
DECLARE
   L_IN Blobarray := Blobarray ();
   v_out_text VARCHAR2(200);

BEGIN
    L_IN.EXTEND;
   select ST_GeomFromEWKT('LINESTRING (4 11, 4 4, 11 4, 11 11, 4 11)')
into l_in(1) from dual;
   L_IN.EXTEND;
   select ST_GeomFromEWKT('LINESTRING (1 9, 1 5, 5 5, 5 9, 1 9)') into
l_in(2) from dual;
      --select geom into l_in(i) from lines where id =1;

    select st_astext(ST_Polygonize(L_IN)) into v_out_text from dual;
   DBMS_OUTPUT.PUT_LINE (v_out_text);
END;
/
```

--Resultado

```
anonymous block completed
GEOMETRYCOLLECTION (POLYGON ((1 9, 5 9, 5 5, 1 5, 1 9)), POLYGON ((4
11, 11 11, 11 4, 4 4, 4 11)))
```

4.15. Tabla de metadatos SPATIAL_REF_SYS

La tabla SPATIAL_REF_SYS se define es las especificaciones OGC para enumerar los sistemas de referencias espaciales y los detalles necesarios para las transformaciones entre ellos.

La vista SPATIAL_REF_SYS contiene un identificador numérico y una descripción textual del sistema de coordenadas espacial de la base de datos. Esta tabla se carga a partir del plugin de GeoTools epsg-wkt. En Jaspa se utiliza una tabla auxiliar AVAILABLESRIDS para cargar los identificadores numéricos.

Este proceso se lleva a cabo en tres pasos:

- Creación del método Java _getAvailableSRIDs
- Publicación del método Java getAvailableSRIDs en el SGBD
- Creación de la tabla AVAILABLESRIDS, y de la vista SPATIAL_REF_SYS en SQL

Paso 1. Creación del método Java _getAvailableSRIDs

```
public static ARRAY _getAvailableSRIDs() throws JASPAException,
                  SQLException {

      ArrayList<Integer> auxList = new ArrayList<Integer>();

      CRSAuthorityFactory factory = ReferencingFactoryFinder
                        .getCRSAuthorityFactory("EPSG", null);
      int SRID;
      String textSRID = null;

      try {
            Iterator<String> iterator = factory.getAuthorityCodes(

      CoordinateReferenceSystem.class).iterator();

            while (iterator.hasNext()) {
                  // Reading from epsg-wkt plugin
                  String text[] = iterator.next().split(":");

                  if ((text.length == 2) &&
text[0].equalsIgnoreCase("EPSG")) {
                        textSRID = text[1];
                        if (!textSRID.startsWith("_")) {
                              SRID = Integer.parseInt(textSRID);
                              auxList.add(SRID);

                        }
            }
            }
      } catch (NumberFormatException ex) {
            throw new JASPAIllegalArgumentException("Invalid srid
number: "
                              + textSRID);
            } catch (FactoryException e) {
                  throw new JASPAGeoToolsException(e);
            }
```

```
            int auxListSize = auxList.size();
            int[] mySridlist = new int[auxListSize];

            for (int i = 0; i < auxListSize; i++) {

                  mySridlist[i] = auxList.get(i);

            }

            Connection conn = new OracleDriver().defaultConnection();
            ArrayDescriptor descriptor =
ArrayDescriptor.createDescriptor(
                        "NUMARRAY", conn);

            oracle.sql.ARRAY p_out = new ARRAY(descriptor, conn,
mySridlist);

            return p_out;
      }
```

Paso 2. Hacer público el método Java getAvailableSRIDs en el SGBD (SQL)

```
create or replace TYPE NUMARRAY AS TABLE OF NUMBER;
/
create or replace FUNCTION getAvailableSRIDs
      RETURN NUMARRAY AS LANGUAGE JAVA Name
'es.upv.jaspa.oracle.SQLEx._getAvailableSRIDs() return
oracle.sql.ARRAY';
/
```

Paso 3. Creación de la tabla AVAILABLESRIDS, y de la vista SPATIAL_REF_SYS en SQL

```
DECLARE
v_count NUMBER :=0;
BEGIN
SELECT     COUNT(*)     INTO     v_count     FROM     all_tables     WHERE
      table_name='AVAILABLESRIDS';
      IF v_count = 1 THEN
            EXECUTE IMMEDIATE 'DROP TABLE AVAILABLESRIDS';
      END IF;
END;
/
CREATE TABLE availableSRIDs (srid integer primary key);
/
SET SERVEROUTPUT ON;
/
DECLARE
  aux numarray;
BEGIN
  select getAvailableSRIDs() into aux from dual;
  DBMS_OUTPUT.PUT_LINE('There   are   '   ||   aux.COUNT   ||   '   SRID
elements.');
```

```
  FOR i IN 1 .. aux.COUNT
   LOOP
    insert into availableSRIDs select aux(i) from dual;
  END LOOP;

END;
/
DROP VIEW spatial_ref_sys;
/
CREATE VIEW spatial_ref_sys (srid, auth_name, auth_srid, srtext) as
SELECT srid,'EPSG' as auth_name,srid, ST_AsWKTSRS(srid) FROM
availableSRIDs;
```

AVAILABLESRIDS

Columnas | Datos | Restriccio

	SRID
1	23894
2	2390
3	2391
4	2392
5	2393
6	2394
7	23946

Fig. 19 Tabla AvailableSrids

SPATIAL_REF_SYS

Columnas | Datos | Permisos | Dependencias | Detalles | Disparadores | SQL

Ordenar... | Filtro:

	SRID	AUTH_NAME	AUTH_SRID	SRTEXT
1	23894	EPSG	23894	PROJCS["ID74 / UTM zone 54S", GEOGCS["ID74", DATUM["Indo...
2	2390	EPSG	2390	PROJCS["Xian 1980 / 3-degree Gauss-Kruger CM 135E", GEOGCS["...
3	2391	EPSG	2391	PROJCS["KKJ / Finland zone 1", GEOGCS["KKJ", DATUM["Kartast...
4	2392	EPSG	2392	PROJCS["KKJ / Finland zone 2", GEOGCS["KKJ", DATUM["Kartast...
5	2393	EPSG	2393	PROJCS["KKJ / Finland Uniform Coordinate System", GEOGCS["KKJ...
6	2394	EPSG	2394	PROJCS["KKJ / Finland zone 4", GEOGCS["KKJ", DATUM["Kartast...
7	23946	EPSG	23946	PROJCS["Indian 1954 / UTM zone 46N", GEOGCS["Indian 1954", ...
8	23947	EPSG	23947	PROJCS["Indian 1954 / UTM zone 47N", GEOGCS["Indian 1954", ...
9	23948	EPSG	23948	PROJCS["Indian 1954 / UTM zone 48N", GEOGCS["Indian 1954", ...
10	2395	EPSG	2395	PROJCS["South Yemen / Gauss-Kruger zone 8", GEOGCS["South Y...
11	2396	EPSG	2396	PROJCS["South Yemen / Gauss-Kruger zone 9", GEOGCS["South Y...
12	2397	EPSG	2397	PROJCS["Pulkovo 1942(83) / 3-degree Gauss-Kruger zone 3", GEO...
13	2398	EPSG	2398	PROJCS["Pulkovo 1942(83) / 3-degree Gauss-Kruger zone 4", GEO...
14	2399	EPSG	2399	PROJCS["Pulkovo 1942(83) / 3-degree Gauss-Kruger zone 5", GEO...
15	2400	EPSG	2400	PROJCS["RT90 2.5 gon W", GEOGCS["RT90", DATUM["Rikets ko...
16	2401	EPSG	2401	PROJCS["Beijing 1954 / 3-degree Gauss-Kruger zone 25", GEOGCS...

Fig. 20 Vista Spatial_Ref_Sys

4.16. Tabla de metadatos geometry_columns

La tabla geometry_columns contiene información sobre las tablas que contienen alguna columna con geometrías georeferenciadas. Las columnas de esta tabla son las siguientes:

- F_TABLE_CATALOG,F_TABLE_SCHEMA,F_TABLE_NAME. Distingue totalmente la tabla de geometrías que contiene la columna geométrica.

- F_GEOMETRY_COLUMN. Nombre de la columna geométrica en la tabla de características.

- COORD_DIMENSION. Dimensión espacial de la columna geométrica (2, 3 o 4 dimensiones).

- SRID. Es una clave foránea que referencia SPATIAL_REF_SYS.

- TYPE. Tipo de objeto espacial. POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON o GEOMETRYCOLLECTION.

La tabla geometry_columns se crea con la siguiente instrucción SQL:

```
/
DROP TABLE geometry_columns;
/
  CREATE TABLE geometry_columns (
  f_table_catalog varchar(256) not null,
  f_table_schema varchar(256) not null,
  f_table_name varchar(256) not null,
  f_geometry_column varchar(256) not null,
  coord_dimension integer not null,
  srid integer not null,
  type varchar(30) not null,
  CONSTRAINT geometry_columns_pk primary key (
  f_table_catalog,
  f_table_schema,
  f_table_name,
  f_geometry_column )
  );
/
GRANT SELECT ON geometry_columns TO public;
```

5. UTILIZACIÓN DE JASPA PARA ORACLE

5.1. Insertar datos espaciales

La entrada de datos espaciales en Oracle se puede realizar utilizando sentencias SQL del tipo INSERT INTO que permite insertar nuevos registros en una tabla existente. Los datos insertados en las columnas de la geometría deben ser una representación de geometría. La construcción de geometrías puede utilizar tomando como entrada geometrías en formato WKT, Well-Known Text (como ST_GeomFromText, ST_LineFromText o en formato WKB (ST_GeomFromWKB, ST_MPointFromText) o funciones propias de JASPA (ST_MakePolygon, ST_MakePoint…).

En Oracle existe una limitación al insertar sentencias SQL, debido a que los literales de cadena están limitados a 4000 caracteres.

Informe de error:
Error SQL: ORA-01704: literal de cadena demasiado largo
01704. 00000 - "string literal too long"
**Cause: The string literal is longer than 4000 characters.*
**Action: Use a string literal of at most 4000 characters.*
Longer values may only be entered using bind variables.

La solución pasa en esos casos por trabajar con PL/SQL, por ejemplo:

```
declare
     geom blob;
begin
     geom := ST_GeomFromText ('MULTIPOLYGON
(((20.046419143676758 -24.927137374876462, 20.043914794921875 -
26.472183227537684, 20.05790138244629 -27.324111938475227,
20.069581985473633 -28.33503723144409, 20.044054031372074 -
28.427246093748774, 19.83953857421875 -28.570554733275184,
19.594211578369144 -28.70656204223516, 19.332651138305657 -
28.930606842039882, 19.140853881835938 -28.945985794066246,
18.879348754882812 -28.87134361266972, 18.544212341308594 -
28.8524665832508, 18.142627716064453 -28.873849868773256,
17.910261154174805 -28.856100082396328, 17.777345657348636 -
28.76074218749884, 17.752477645874023 -28.52300643920777,
17.73366928100586 -28.368125915526132, 17.646650314331055 -
28.268297195433362, 17.360239028930664 -28.112037658690173,
17.212162017822266 -
[…]
21.14139938354493 -19.898061752317552, 21.121915817260746 -
20.438587188718923, 21.122659683227536 -22.04075241088699,
20.85922813415527 -22.061363220213167, 20.15521621704102 -
22.084537506101825, 20.089704513549805 -22.097640991209246,
```

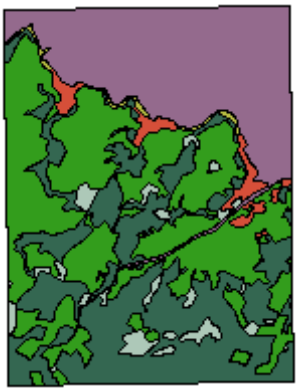
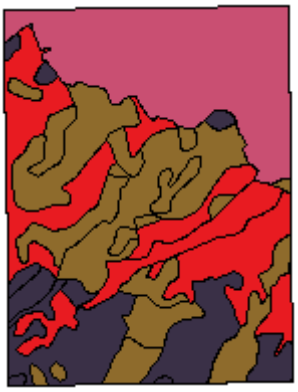
```
20.041919708251953 -23.29318618774251, 20.03795051574707 -
23.833623886106828, 20.043533325195312 -24.564750671385198,
20.046419143676758 -24.927137374876462)))',4326);

INSERT INTO mundo (geom, gid, name) VALUES (geom, 47,
'Namibia');
end;
```

5.2. Carga de datos

A continuación se van a insertar tres capas de información geográfica. La tabla “uses” se corresponde con datos sobre usos del suelo. La tabla de “soils” contiene información sobre diferentes tipos de suelos. Por su parte, en la capa “rivers” se almacenan cursos hidrográficos de la región en la que se distinguen dos tipos.

Los datos estaban almacenados en ficheros shp (formato de archivo informático propietario de datos espaciales desarrollado por la compañía ESRI). Con el conversor shp2jaspapg se convierten los ficheros shp a ficheros sql que contienen sentencias SQL de creación de tablas e inserción de datos. Finalmente ya podemos crear las tablas y poblarlas de datos con los ficheros sql obtenidos.

Tabla 9 Datos del tutorial

		
uses	soils	rivers

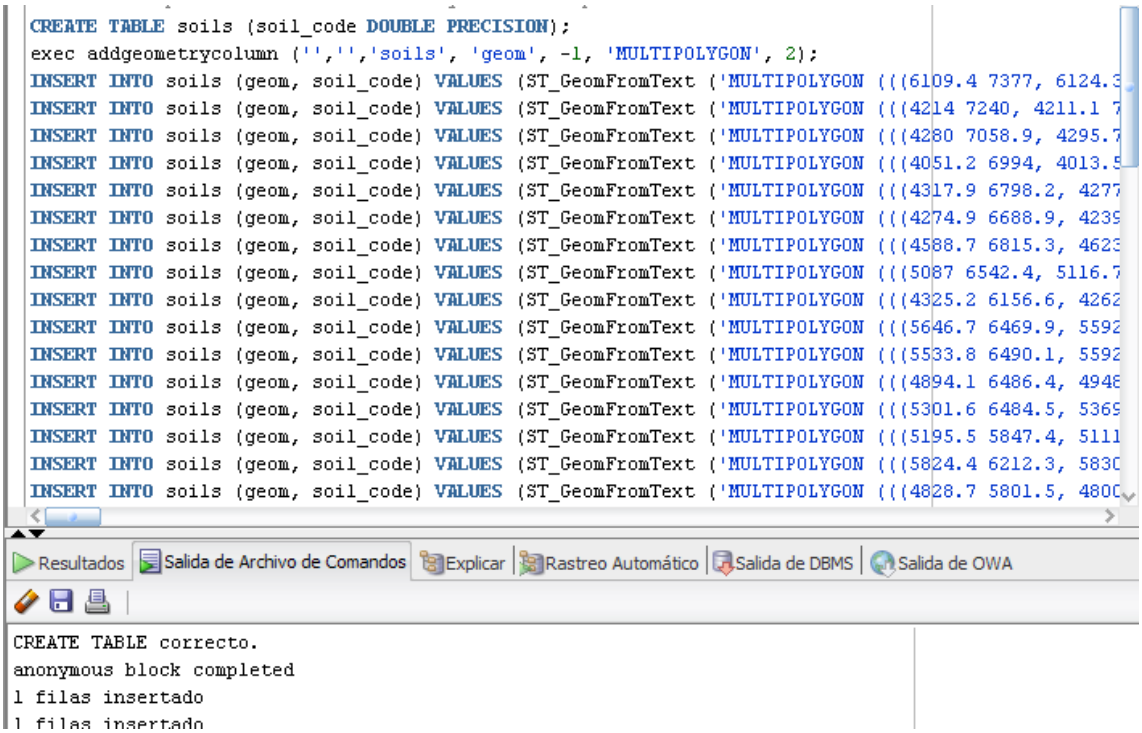


Fig. 21 Ejemplo inserción de datos en la tabla "soils"

- Creamos la función orcl_box.

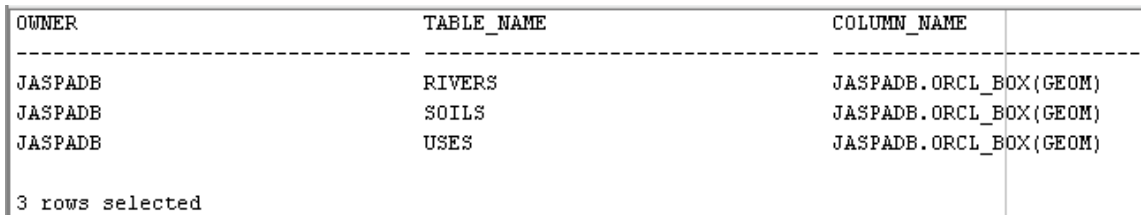
```
SQL>
create or replace function orcl_box (geom in blob)
      return Sdo_GEOMETRY deterministic is
            v_box BLOB;
            v_xmin NUMBER;
            v_xmax NUMBER;
            v_ymin NUMBER;
            v_ymax NUMBER;
      begin
            SELECT st_box2d(geom) into v_box from dual;
            SELECT st_xmin(v_box) into v_xmin from dual;
            SELECT st_xmax(v_box) into v_xmax from dual;
            SELECT st_ymin(v_box) into v_ymin from dual;
            SELECT st_ymax(v_box) into v_ymax from dual;
            return SDO_GEOMETRY(
                  2003,
                  NULL,
                  NULL,
                  SDO_ELEM_INFO_ARRAY(1,1003,3),
                  SDO_ORDINATE_ARRAY(v_xmin,v_ymin, v_xmax, v_ymax)
            );
end orcl_box;
```

- Actualizamos la vista USER_SDO_GEOM_METADATA.

```
SQL>
INSERT INTO user_sdo_geom_metadata(TABLE_NAME,COLUMN_NAME,DIMINFO,SRID)
VALUES ('soils','jaspadb.orcl_box(geom)',SDO_DIM_ARRAY(  SDO_DIM_ELEMENT('X', 0, 20,
0.005), SDO_DIM_ELEMENT('Y', 0, 20, 0.005)),NULL);

INSERT INTO user_sdo_geom_metadata(TABLE_NAME,COLUMN_NAME,DIMINFO,SRID)
VALUES  ('uses','jaspadb.orcl_box(geom)',SDO_DIM_ARRAY(SDO_DIM_ELEMENT('X',  0,  20,
0.005), SDO_DIM_ELEMENT('Y', 0, 20, 0.005)),NULL);

INSERT INTO user_sdo_geom_metadata(TABLE_NAME,COLUMN_NAME,DIMINFO,SRID)
VALUES  ('rivers','jaspadb.orcl_box(geom)',SDO_DIM_ARRAY(SDO_DIM_ELEMENT('X',  0,  20,
0.005),SDO_DIM_ELEMENT('Y', 0, 20, 0.005)),NULL);
```

- Podemos consultar la información de metadatos para todas las tablas espaciales en las que el usuario tiene permiso SELECT con la siguiente sentencia:

```
SQL> SELECT * FROM all_sdo_geom_metadata;
```



OWNER	TABLE_NAME	COLUMN_NAME
JASPADB	RIVERS	JASPADB.ORCL_BOX(GEOM)
JASPADB	SOILS	JASPADB.ORCL_BOX(GEOM)
JASPADB	USES	JASPADB.ORCL_BOX(GEOM)

3 rows selected

Fig. 22 Metadatos de las tablas espaciales

- En el siguiente paso creamos los índices espaciales propiamente dichos:

```
SQL>
CREATE INDEX soils_idx on soils(orcl_box(geom))
indextype is mdsys.spatial_index;

CREATE INDEX uses_idx on uses(orcl_box(geom))
indextype is mdsys.spatial_index;

CREATE INDEX rivers_idx on rivers(orcl_box(geom))
indextype is mdsys.spatial_index;
```

Si todo ha ido bien deberíamos obtener el mensaje “create index correcto.”

5.4. Consultas Básicas

Con los datos ya cargados en la base de datos tutorial se pueden realizar las consultas. A continuación mostramos diferentes ejemplos:

Calcula el área total de la tabla “uses” en hectáreas:

```
SQL> SELECT Sum(ST_Area(geom))/10000 AS has_uses FROM uses;
HAS_USES
----------------------
304,7339

1 rows selected
```

Calcula el área total, en hectáreas, de aquellos polígonos que cumplan "use_code=300":

```
SQL> SELECT Sum(ST_Area(geom))/10000 AS hectare_uses300
       FROM uses
       WHERE use_code=300;

 HECTARE_USES300
----------------------
21,23765

1 rows selected
```

Calcula la longitud total en kilómetros de la capa de "ríos":

```
SQL> SELECT Sum(ST_Length(geom))/1000 AS km_rivers
       FROM rivers;

KM_RIVERS
----------------------
21,59161001221247918

1 rows selected
```

Calcula cuál es el río más largo:

```
SQL> SELECT * FROM (SELECT ST_Length(geom)/1000 AS length_km
       FROM rivers ORDER BY length_km DESC)
       WHERE ROWNUM = 1;

LENGTH_KM
----------------------
0,708044614061041
1 rows selected
```

5.5. Almacenar los resultados en tablas y vistas

En el siguiente ejemplo se van a almacenar los centroides de los suelos de tabla en una nueva tabla.

La opción más recomendable es crear previamente la tabla y utilizar la función *AddGeometryColumn* para añadir la columna de geometrías.

Creamos además una secuencia para simular una columna de tipo serial de PostgreSQL:

```
SQL>
CREATE SEQUENCE serial
    MINVALUE 1
    START WITH 1
    INCREMENT BY 1
    CACHE 20;

CREATE TABLE centroid_uses (id NUMBER PRIMARY KEY,use_code INTEGER);

EXECUTE AddGeometryColumn ('','','centroid_uses','geom',-1,'POINT',2);
```

Una vez que se han creado las tablas hay que poblarlas de datos. En el siguiente ejemplo vamos a almacenar los centroides de la tabla “uses” en la nueva tabla “centroid_uses”.

```
SQL>
INSERT INTO centroid_uses (id,use_code,geom) SELECT serial.nextval,
u.use_code, ST_Centroid(u.geom) from uses u;
```

Consultamos los datos insertados en la tabla centroid_uses:

```
SQL>
SELECT id,use_code,st_astext(geom)
      FROM centroid_uses;
```

ID	USE_CODE	ST_ASTEXT(GEOM)
2	300	POINT (4045.7875894988065 5323.087112171837)
3	300	POINT (5766.636997478119 5286.629135143154)
4	200	POINT (5717.523277506011 5176.678785617555)
5	300	POINT (5378.921983701054 5223.529835022858)
6	300	POINT (4749.424631195094 5227.082877507045)
7	300	POINT (4606.9344078343365 5215.67472542419)
8	200	POINT (4304.030033178014 5016.443935447012)
9	200	POINT (6126.9113333333335 5157.564333333333)
10	300	POINT (4063.2664993887925 5075.0125587081)
11	200	POINT (5025.181662142371 5004.147146895027)
12	200	POINT (6007.784980889811 4898.980755928303)
13	300	POINT (5475.122951545752 4968.043424066411)
14	300	POINT (5966.513612950699 5050.43451066961)
15	300	POINT (5833.340694601737 4994.10731151293995)
16	200	POINT (4498.72704625168 4849.354239282059)
17	400	POINT (4224.891235769255 4785.612864911276)

Fig. 23 Resultado de la creación de centroides de la tabla usos del suelo

5.6. Actualizar una tabla

Vamos a redondear las coordenadas de los puntos de la tabla "centroid_uses" al centímetro. Para ello hay que actualizar el campo geom.

```
SQL>
UPDATE centroid_uses SET geom = ST_SnapToGrid(geom, 0.01);
--Resultado
73 filas actualizado
```

Vuelve a obtener los datos de la tabla “uses” cuyo código de uso sea igual a 100:

```
SQL>
SELECT id, use_code, ST_AsText(geom) FROM centroid_uses where
use_code=100 ORDER BY id;
```

ID	USE_CODE	ST_ASTEXT(GEOM)
29	100	POINT (4460.75 6733.05)
36	100	POINT (5091.43 6489.13)
42	100	POINT (5786.2 6099.61)
50	100	POINT (6084.96 6070.6)
54	100	POINT (6007.74 5978.36)
58	100	POINT (5809.36 5842.22)
64	100	POINT (5905.58 5695.95)

7 rows selected

5.7. Intersecciones

Vamos a obtener la intersección de las dos tablas "uses" and "soils":

- En primer lugar obtenemos el número de valores que se obtienen en el producto cruzado de dos tablas:

```
SQL>
SELECT count(*)
 FROM soils s, uses u;
COUNT(*)
----------------------
3139

1 rows selected
```

- Al realizar la intersección sin hacer uso del filtrado es como si estuviéramos haciendo un producto cruzado entre tablas, la consulta tarda 16.7 segundos en ejecutarse. El número de elementos obtenidos es de 3139.

```
SQL>
SELECT st_astext(Geom) from
(select ST_Intersection(s.geom,u.geom) as Geom
 FROM soils s, uses u );
```

```
16,76099586 seconds
3139 rows selected
```

- En este caso aplicamos la indexación espacial al utilizar la función SDO_FILTER. En este caso el número de elementos obtenidos se reduce a 395, y el tiempo a 3.95 segundos. Los valores nulos se corresponden con geometrías cuyos rectángulos límites intersecan pero las propias geometrías no.

```
3,79087806 seconds
```

```
SQL>
SELECT st_astext(Geom) from
(SELECT ST_Intersection(s.geom,u.geom) as Geom
 FROM soils s, uses u WHERE (SDO_FILTER(orcl_box(s.geom),orcl_box(u.geom)) = 'TRUE')
);
```

```
ST_ASTEXT(GEOM)
--------------------------------------------------------------------------------
POLYGON ((4053.267510694878 5342.701017458666, 4032.5740362359325 5315.131862079974, 4032.0043478260


POLYGON ((4032.5740362359325 5315.131862079974, 4053.267510694878 5342.701017458666, 4070 5324, 4033
POLYGON ((5753.086228218101 5290.526138279933, 5756.8 5291.6, 5761.714831317632 5308.016549968173, 5


POLYGON ((5761.714831317632 5308.016549968173, 5756.8 5291.6, 5753.086228218101 5290.526138279933, 5
MULTIPOLYGON (((4422.4 5285.5, 4467.2 5247.7, 4471.758877284595 5241.705352480418, 4464 5241, 4430 5
POLYGON ((4395 5365.9, 4430.7 5355.1, 4432.104397355213 5355.117302636821, 4445 5350, 4468.631614654
POLYGON ((4390.289200736936 5371.7106346282, 4395 5365.9, 4347.9 5278.1, 4304.3 5225.2, 4213.3 5146.
POLYGON ((4422.4 5285.5, 4468.631614654002 5342.437883310719, 4470 5342, 4505 5353, 4573.14881227920

POLYGON ((4839 5393, 4837 5327, 4788 5298, 4822 5344, 4839 5393))



395 rows selected
```

- En este caso a la función SDO_FILTER añadimos una restricción de geometrías no nulas para asegurarnos que las geometrías realmente intersequen y no solo sus cajas.

```
SQL>
SELECT st_astext(Geom) from
(SELECT ST_Intersection(s.geom,u.geom) as Geom
 FROM soils s, uses u WHERE (SDO_FILTER(orcl_box(s.geom),orcl_box(u.geom)) = 'TRUE')
)
 WHERE Geom IS NOT NULL;
```

```
3,84484744 seconds

ST_ASTEXT(GEOM)
--------------------------------------------------------------------------------
POLYGON ((4053.267510694878 5342.701017458666, 4032.5740362359325 5315.131862079974, 4032.0043478
POLYGON ((4032.5740362359325 5315.131862079974, 4053.267510694878 5342.701017458666, 4070 5324, 4
POLYGON ((5753.086228218101 5290.526138279933, 5756.8 5291.6, 5761.714831317632 5308.016549968173
POLYGON ((5761.714831317632 5308.016549968173, 5756.8 5291.6, 5753.086228218101 5290.526138279933
```

.

```
POLYGON ((4395.892780094923 5369.486992025824, 4395 5365.9, 4390.289200736936 5371.7106346282, 4395.
MULTIPOLYGON (((4241.67381254267 5067.861679508436, 4249.20933726858 5065.594150791521, 4240 5065, 4
MULTIPOLYGON (((4422.4 5285.5, 4467.2 5247.7, 4471.758877284595 5241.705352480418, 4464 5241, 4430 5
POLYGON ((4395 5365.9, 4430.7 5355.1, 4432.104397355213 5355.117302636821, 4445 5350, 4468.631614654
POLYGON ((4390.289200736936 5371.7106346282, 4395 5365.9, 4347.9 5278.1, 4304.3 5225.2, 4213.3 5146.
POLYGON ((4422.4 5285.5, 4468.631614654002 5342.437883310719, 4470 5342, 4505 5353, 4573.14881227920
POLYGON ((4839 5393, 4837 5327, 4788 5298, 4822 5344, 4839 5393))

223 rows selected
```

5.8. Buffer

Creamos una nueva tabla de atributos "riverdistance" para utilizarla posteriormente como distancia para los buffers.

```
SQL>
CREATE TABLE riversdistance (river_code integer, dist real);

INSERT INTO riversdistance values (1,40);

INSERT INTO riversdistance values (2,20);
```

RIVER_CODE	DIST
1	40
2	20

Ahora creamos un buffer de la capa “rivers” utilizando como parámetro de distancia el valor de la tabla "riverdistance". Almacenamos los resultados en la tabla "buf2":

```
SQL>
CREATE TABLE buf2 (id number PRIMARY KEY);

EXECUTE addgeometrycolumn ('','','buf2','geom',-1,'MULTIPOLYGON',2);

INSERT INTO buf2(id,geom) SELECT serial.nextval,
st_multi(st_buffer(r.geom, d.dist, 16)) FROM rivers r, riversdistance
d where r.river_code = d.river_code;
```

```
--Resultado
106 filas insertado
```

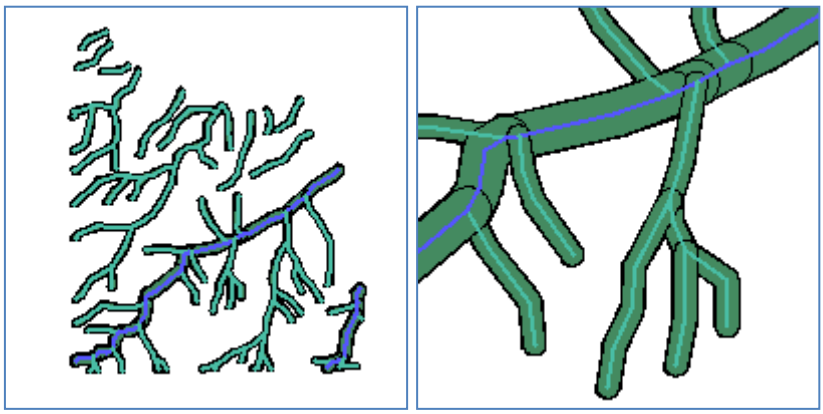
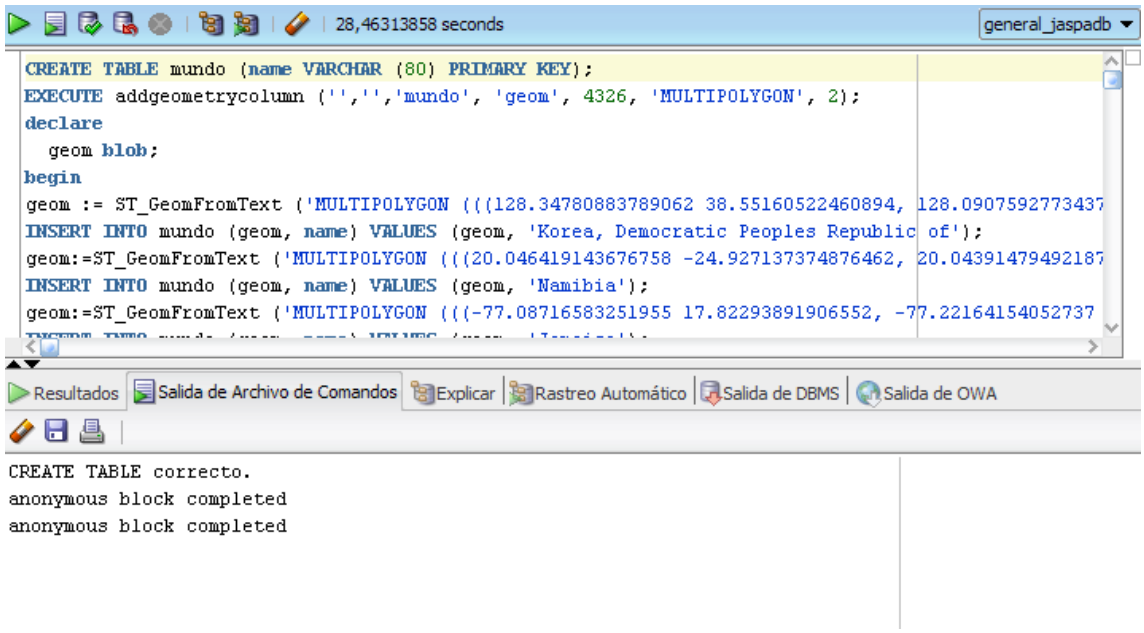


Fig. 24 Tabla buf2 y detalle

5.9. Reproyección de coordenadas

Para este ejercicio vamos a utilizar una capa de países, con sistema de referencia EPSG 4326 (Spherical Mercator latitud y longitud).

Creamos la tabla “mundo” e insertamos los diferentes registros:



	NAME	GEOM
1	Oman	(BLOB)
2	Kuwait	(BLOB)
3	Algeria	(BLOB)
4	Cuba	(BLOB)
5	Brazil	(BLOB)
6	Puerto Rico	(BLOB)
7	Montenegro	(BLOB)
8	Zambia	(BLOB)
9	Belgium	(BLOB)

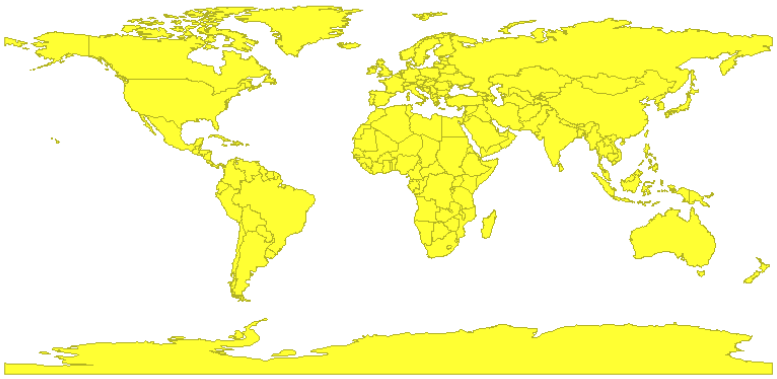


Fig. 25 Capa mundo con SR EPSG 4326

Reproyectamos al sistema de referencia EPSG=3395:

SQL>

```
SELECT UpdateGeometrySrid('mundo','geom',3395);
UPDATE mundo SET geom=(ST_Transform(ST_SetSRID(geom,4326),3395));
SELECT UpdateGeometrySrid('mundo','geom',-1);
```

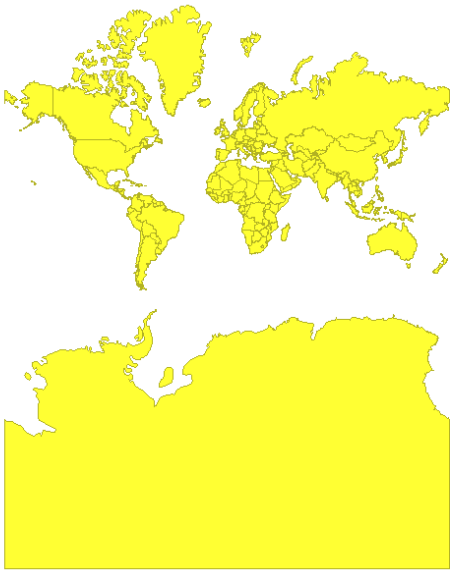


Fig. 26 Capa mundo con SR EPSG 3395

6. CONCLUSIONES

Durante los últimos años han aparecido numerosas iniciativas orientadas al desarrollo de aplicaciones informáticas de Sistemas de Información Geográfica bajo la filosofía del software libre. Los sistemas gestores de bases de datos relacionales son herramientas especializadas en el almacenamiento de datos, pero no incorporan de forma nativa soporte para los datos espaciales. Una de esas iniciativas es la extensión espacial de bases de datos Jaspa.

Los proyectos de software libre crecen con la interacción de la comunidad. Al ser Jaspa un proyecto bastante reciente no tiene actualmente una gran comunidad de usuarios. A pesar de ello, Jaspa puede evolucionar rápidamente gracias a la accesibilidad del software libre y a la utilización de Java. De hecho, se están llevando a cabo algunos desarrollos como la indexación espacial para la base de datos H2. Debido a su extensibilidad Jaspa está abierto a un amplio rango de profesionales que, pese a no poseer grandes habilidades en cuanto a programación, trabajen diariamente con datos espaciales.

La portabilidad es otra fortaleza destacable. Hasta la fecha, Jaspa podía ser desplegado en PostgreSQL o H2. Con este trabajo se ha desarrollado su utilización en combinación con la base de datos Oracle. Oracle Spatial se utiliza fundamentalmente en los Sistemas de Información Geográfica para implementar las referencias geográficas y ejecutar consultas espaciales.

Se han estudiado los estándares que existen sobre datos geográficos para que el producto sea compatible con otras herramientas y pueda integrarse en nuevos proyectos de diferente índole. En este proyecto se ha reutilizado software existente, factor que conduce a minimizar los esfuerzos y así concentrar la investigación en características totalmente innovadoras (Fichman y Kemerer, 2001). En este sentido, Jaspa se basa principalmente en las librerías JTS y GeoTools, ambos son productos de probada fiabilidad que soportan estándares abiertos para garantizar la interoperabilidad entre los componentes.

Asimismo se han mostrado las diferentes soluciones que existen para integrar la información geográfica. En la arquitectura integrada se deben incorporar tipos de datos espaciales, funciones e indexación espacial.

Para el desarrollo es necesario importar a la base de datos Oracle las librerías del proyecto. Estas librerías son el propio distribuible de Jaspa, las de JTS y las de GeoTools. Estas librerías se facilitarán cuando se publique la versión final de Jaspa para Oracle.

La funcionalidad disponible es muy variada. Incluye, por ejemplo, operadores espaciales para obtener las relaciones espaciales entre geometrías, creación de entidades espaciales, consultas sobre las geometrías como áreas, vértices, coordenadas límites, etc. Se posibilita el tratamiento de geometrías asociadas a un sistema de coordenadas de referencia, lo que es imprescindible para los cálculos sobre superficies de referencia elipsoídicas o reproyecciones cartográficas.

Uno de los problemas detectados es que en ciertas ocasiones la documentación existente para extender Oracle era escasa. Esto fue especialmente remarcable a la hora de construir las funciones de agregados o de crear Arrays de geometrías.

Otra de las dificultades encontradas es el manejo de datos espaciales que se tratan como datos binarios. Este tipo de datos requiere un tratamiento más especializado que los tipos habituales numéricos o textuales. A esto hay que sumar que en muchas ocasiones al manejar Oracle se obtienen errores genéricos del tipo ORA-XX que no aportan información de qué es lo que está fallando, por lo que la resolución de errores se ralentiza.

Debido a la naturaleza voluminosa de los datos espaciales, la indexación es un elemento imprescindible en cualquier base de datos espacial que aspire a ser eficiente. Para implementar la indexación se han utilizado las herramientas Locator que vienen incorporadas en cualquier versión de Oracle 11g. Para ello se deben obtener rectángulos delimitadores de las geometrías en formato SDO_Geom de Oracle y utilizar las tablas de metadatos propias de Oracle. Por tanto, para implementar la indexación de Jaspa para Oracle ha sido necesario estudiar cómo trabaja el propio sistema de Oracle Spatial.

Se pretende que la versión de Jaspa para Oracle forme parte del núcleo de Jaspa y que se publique en una versión próxima. Así la extensión espacial ofrecerá soporte para las bases de datos PostgreSQL, H2 y Oracle. Esta nueva versión puede ofrecer la posibilidad de atraer a nuevos usuarios de Jaspa. Deberían desarrollarse conectores de bases de datos a otras aplicaciones como Sistemas de Información Geográfica de escritorio, que permiten la visualización y edición de datos espaciales.

Otros desarrollos futuros pueden consistir en ampliar su funcionalidad, por ejemplo, incorporando diferentes desarrollos de GeoTools. Además se puede añadir el sistema de reglas topológicas de Jaspa 0.2 para PostgreSQL con el objetivo de controlar las relaciones entre las entidades de las diferentes tablas.

El hecho de añadir una extensión libre a la base de datos propietaria predominante abre la posibilidad de utilizar datos geográficos a muchos usuarios con su sistema ya establecido, que no desean adquirir la licencia Oracle Spatial ni migrar a bases de datos libres. Además de que Oracle Spatial es en muchos casos costoso en cuanto a aprendizaje e implementación. En cualquier caso, la solución aportada puede ser utilizada por sí misma o en combinación con las capacidades espaciales de Oracle Spatial.

7. BIBLIOGRAFÍA

Publicaciones Previas al Trabajo Fin de Máster

Martinez-Llario, J., Gonzalez-Alcaide, M., Design of a Java spatial extension for relational databases. *Journal of Systems and Software. Volume 84 Issue 12, December, 2011 Elsevier Science Inc. New York, NY, USA*
doi:10.1016/j.jss.2011.06.072

Martinez-Llario, J., Gonzalez-Alcaide, M., JASPA, la mejor alternativa libre a PostGIS. *V Jornadas SIG libre. Servicio de Sistemas de Información Geográfica y Teledetección de la Universidad de Girona.* Girona, 2011

Martinez-Llario, J., Gonzalez-Alcaide, M. Póster: "JASPA, an alternative to PostGIS". *FOSS4G. Free and Open Source Software for Geomatics Conference. Open Source Geospatial Foundation (OSGeo).* Barcelona, 2010

Bibliografía

Arens, C., Stoter, J., van Oosterom, P., 2005. Modelling 3D spatial objects in a geo-DBMS using a 3D primitive. Computers & Geosciences 31 (2), 165–177.

Baca Moreno-Torres, I., de Diego Alarcón, J., Pérez Navarro, A., 2011. Desarrollo de un índice espacial para la extensión JASPA sobre H2. Universidad Oberta Cataluña.

Benedikt, M., Libkin, L., 2002. Aggregate operators in constraint query languages.Journal of Computer and System Sciences 64 (3), 628–654.

Brown, D.G., Riolo, R., Robinson, D.T., North, M., Rand, W., 2005. Spatial process and data models: toward integration of agent-based models and GIS. Journal of Geographical Systems 7 (1), 25–47.

Chen, R., Xie, J., 2008. Open source databases and their spatial extensions. In: Hall, G.B., Leahy, M.G. (Eds.), Open Source Approaches in Spatial Data Handling. Springer, Berlin Heidelberg, pp. 105–129, http://dx.doi.org/10.1007/978-3-540-74831-1 6.

Davis, M., 2006. JTS topology suite. http://www. vividsolutions.com/jts/JTSHome.htm.

Dell, 2011. Programa de licencias de procesador de Oracle.
http://content.dell.com/es/es/grandes-corporaciones/vsl-oracle-processor

Deoliveira, J., 2010. GeoDB. http://github.com/jdeolive/geodb.

Douglas, K., 2006. PostgreSQL: A comprehensive guide to building, programming and administering PostgreSQL databases. Sams. ISBN 0735712573.

Earp, R. W. Advanced SQL functions in Oracle 10g. Wordware, 2006. ISBN 9781598220216

Fichman, R.G., Kemerer, C.F., 2001. Incentive compatibility and systematic software reuse. Journal of Systems and Software 57 (1), 45.

Goetz, B., 2004. Java Theory and Practice: The Exceptions Debate, http://www.ibm.com/developerworks/java/library/j-jtp05254.html.

Feuerstein, Steven, 1999. Oracle PL/SQL Programming Guide to Oracle8i Features. ISBN 1-56592-675-7E. First Edition, published 1999-10-01. http://docstore.mik.ua/orelly/oracle/guide8i/ch09_04.htm

Gabillaud, Jérôme. 2005. Oracle 10g: SQL, PL/SQL, SQL*Plus. Ediciones ENI, 496 páginas

Greener, Simon. 2007; Oracle Locator vs Oracle Spatial: A Reflection on Oracle Licensing of the SDO_GEOM Package. SpatialDB Advisor. 2007
Disponible en: http://www.spatialdbadvisor.com/

Groot, R., McLaughlin, J., 2000. Geospatial Data Infrastructure. Concepts, Cases and Good Practice. Oxford University Press, Oxford.

Hwang, S.; Kwon, K.; Cha, S. K.; Lee, B. S. (2003). Performance Evaluation of Main-Memory R-tree Variants". Advances in Spatial and Temporal Databases. Lecture Notes in Computer Science. 2750. pp. 10. doi:10.1007/978-3-540-45072-6_2. ISBN 978-3-540-40535-1.

Infor Consult Soluciones, 2004. Índices Basados en Funciones.
http://www.inforconsult.es/oracle/ficheros/indices_basados_en_funciones.pdf

ISO/IEC 13249-3:2006, 2006. Information technology – database languages – SQL multimedia and application packages – part 3: spatial.

Jellema, Lucas, 2005. Oracle Quiz on SQL and PL/SQL – See water burning. AMIS Technology Weblog.

Ki Sun Song, 2010. 20 Minutes to Understanding Spatial Database.
http://www.cubrid.org/blog/dev-platform/20-minutes-to-understanding-spatial-database/

Kralidis, A.T., 2008. Geospatial open source and open standards convergences. Advances in Geographic Information Science 2, 1–20, http://dx.doi.org/10.1007/978-3-540-74831-1 1.

Manso Callejo, M. A., 2006. Aplicaciones informáticas “Open Source” en el contexto de las IDEs.

Martínez Llario J.C, 2008. Talleres prácticos de iniciacion a Postgis (Linux Y PostgreSQL). Universidad Politécnica De Valencia. ISBN 9788483632550.

Martinez-Llario, J., Gonzalez-Alcaide, M., Design of a Java spatial extension for relational databases. *Journal of Systems and Software* (July 2011) doi:10.1016/j.jss.2011.06.072

Martinez-Llario, J., Weber-Jahnke, J.H., Coll, E., 2009. Improving dissolve spatial operations in a simple feature model. Advances in Engineering Software 40 (3), 170–175.

Matthew, N., Stones, R., 2005. Beginning Databases with postgreSQL: From Novice to Professional. Apress, New York.

Montesinos, M. Y Sanz, J.G, 2007. Panorama actual del ecosistema de software libre para SIG. I Jornadas SIG Libre.

OpenGIS Consortium, 1999. OpenGIS Simple Features Specification for SQL, Revision 1.1. OpenGIS Project Document 99-049.

OpenGIS Consortium, 2006. OpenGIS implementation specification for geographic information – simple feature access – part 2: SQL option (revision 1.2.1 ed.).
http://www.opengeospatial.org/standards/sfs.

OpenGeo. Spatial Database Tips and Tricks Workshop.
Disponible en: http://workshops.opengeo.org/postgis-spatialdbtips/

Oracle® Database SQL Reference 10g Release 1 (10.1). December 2003.
http://docs.oracle.com/cd/B14117_01/server.101/b10759.pdf

Oracle9i, 2002. Data Cartridge Developer's Guide Release 2 (9.2). March 2002. Part No. A96595-01 http://www.stanford.edu/dept/itss/docs/oracle/9i/appdev.920/a96595.pdf

Oracle® Spatial Developer's Guide 11*g* Release 1 (11.1). 2005. Chapter 9 Extending Spatial Indexing Capabilities.
http://docs.oracle.com/cd/B28359_01/appdev.111/b28400/sdo_exten.htm

Oracle Spatial 11g, 2007. Administración Avanzada de Datos Espaciales para la Empresa
http://www.oracle.com/technetwork/es/documentation/317501-esa.pdf

Peterson, M. P. International perspectives on maps and the Internet. ISBN 9783540720287.

PostgreSQL, 2009. PostgreSQL 8.4.5 documentation.
http://www.postgresql.org/docs/8.4/interactive/index.html.

Ramsey, P., 2007. The state of open source GIS. In: Paper Presented at the Vancouver, BC, Canada.
http://www.foss4g2007.org/ presentations/viewattachment.php?attachment id=8.

Soler, T., Hothem, L.D., 1988. Coordinate systems used in geodesy: basic definitions and concepts. Journal of Surveying Engineering 114 (2),
84–97.

Steiniger, S., Hay, G.J., 2009. Free and open source geographic information tools for landscape ecology. Ecological Informatics 4 (4), 183–195.

Stolze, K., 2003. Sql/mm spatial: the standard to manage spatial data in relational database systems. In: Paper Presented at the Proceedings of the BTW, LNI P-26, pp. 247–264.

Stonebraker, M., 2007. PostGIS Case Studies. What is it, who is using it, and why? Free and Open Source software for geospatial. Canada, 2007.

www.foss4g2007.org/presentations/viewattachment.php?attachment_id=7

Van Oosterom, P., Stoter, J., Quak, W., Zlatanova, S., 2002. The balance between geometry and topology. In: Paper Presented at the Advances in Spatial Data Handling , 10th International Symposium on Spatial Data Handling, pp. 209–224.

Vijlbrief, T., van Oosterom, P., 1992. The GEO++ system: an extensible GIS. In: Proceedings of the 5th International Symposium on Spatial Data Handling , Charleston, South Carolina, pp. 40–50.

Zakhour, S., Hommel, S., Royal, J., Rabinovitch, I., Risser, T., Hoeber, M., 2006. Exceptions.

The Java Tutorial: A Short Course on the Basics, 4th edition java series (4th ed.). Prentice Hall PTR, Upper Saddle River, NJ.

Zlatanova, S., 2006. 3D geometries in spatial DBMS. In: Cartwright, W. (Ed.), Innovations in 3D Geo Information Systems. Springer, Berlin, Heidelberg, pp. 1–14,
http://dx.doi.org/10.1007/978-3-540-36998-1 1.

Zlatanova, S., Stoter, J., 2006. The role of DBMS in the new generation GIS architecture. In: Rana, S., Sharma, J. (Eds.), Frontiers of Geographic Information Technology. Springer, Berlin Heidelberg, pp. 155–180, http://dx.doi.org/10.1007/3-540-31305-2 8.