
Curso de PostGIS 2.0 - PATHII, Tegucigalpa 2013 Documentation

Versión 1.0

Micho García

05 de julio de 2017

1. Instalación de la máquina virtual	3
1.1. Instalación de VirtualBox	3
1.2. Creación de una máquina virtual	5
1.3. Instalación de Ubuntu/Linux	13
2. Introducción a Linux	17
3. Teoría de base de datos	25
3.1. Bases de datos, el enfoque general	25
3.2. Tablas, columnas, registros	27
3.3. Modelización de base de datos	27
3.4. Referencias	34
4. Conceptos básicos de SQL	35
4.1. Introducción	35
4.2. Componentes del SQL	36
4.3. Consultas	40
4.4. Manejo de varias tablas	44
4.5. Vistas	46
4.6. Referencias	46
5. Ejemplos	47
6. <i>PostGIS</i>, características espaciales	51
6.1. Introducción	51
6.2. Instalación y configuración de <i>PostGIS</i>	51
6.3. Índices espaciales	57
6.4. Funciones espaciales	57
6.5. Otros módulos	58
6.6. Prácticas	58
7. Simple Feature Model	59
7.1. OGC y el Simple Feature Model	59
7.2. WKT y WKB	61
7.3. Tipos de datos espaciales	61
7.4. Definición de geometrías básicas	62
7.5. Práctica	63

7.6. Referencias	63
8. Ejemplos SFA	65
9. Importación y exportación de datos	69
9.1. Importación ESRI shapes mediante shp2pgsql	69
9.2. Exportación desde PostGIS a archivos de tipo ESRI Shapefile	71
9.3. GDAL/OGR	72
9.4. Importación datos OSM a PostGIS	74
9.5. Consulta mediante visores web y SIG escritorio	77
9.6. Referencias	77
10. Indexación espacial	79
10.1. Como funcionan los índices espaciales	79
10.2. Creación de índices espaciales	80
10.3. Ejemplo	81
10.4. ANALYZE y VACUUM	81
10.5. Planificador	81
10.6. Operador embebido	82
11. Relaciones espaciales	85
11.1. Introducción	85
11.2. Matriz DE-9IM	85
11.3. Predicados espaciales	87
11.4. JOINS espaciales	93
12. Validación	95
12.1. Validar geometrías	95
12.2. Diferencias entre el modelo ESRI y el modelo propuesto por OGC	97
13. Análisis espacial	103
13.1. Operadores espaciales	103
13.2. Transformación y edición de coordenadas	107
13.3. Simplificación de geometrías	108
14. GeoNetwork como catálogo de Metadatos	111
14.1. Introducción a los metadatos	111
14.2. CSW el estandar OGC para servir los metadatos	114
14.3. GeoNetwork	114
15. Indices and tables	121

Contents:

CAPÍTULO 1

Instalación de la máquina virtual

Nota:	Fecha	Autores
	24 Junio 2013	■ Fernando González (fernando.gonzalez@fao.org)

©2013 FAO Forestry

Excepto donde quede reflejado de otra manera, la presente documentación se halla bajo licencia : Creative Commons (Creative Commons - Attribution - Share Alike: <http://creativecommons.org/licenses/by-sa/3.0/deed.es>)

La formación se va a realizar en una máquina virtual. Para ello se utilizará un software de virtualización, que se encargará de hospedar la máquina virtual. Para el caso que nos ocupa crearemos una máquina ubuntu/linux dentro del software de virtualización VirtualBox.

Los pasos necesarios para esto son:

- Descarga e instalación de VirtualBox.
- Creación de una máquina máquina virtual
- Instalación de ubuntu/linux

En la terminología de los software de virtualización, la máquina real es la anfitriona, *host* en inglés; mientras que la máquina virtual es la huésped, o *guest* en inglés.

Instalación de VirtualBox

El primero de los pasos es descargar VirtualBox del epígrafe “VirtualBox platform packages” de la página de descargas*⁰ y proceder a su instalación.

⁰ <https://www.virtualbox.org/wiki/Downloads>

Más adelante será necesario instalar Ubuntu/Linux, por lo que es recomendable realizar la descarga del programa de instalación mientras se prepara la máquina virtual. Para ello es necesario ir a la página de descargas de Ubuntu⁰ y descargar Ubuntu Desktop, preferiblemente el paquete para 32 bits de la versión 12.04 LTS (Long Term Support, soporte a largo plazo).

La página de Ubuntu pide una pequeña contribución para la descarga, pero no es obligatorio hacerla. Es posible continuar hacia la descarga pinchando en el enlace “not now, take me to the download”:

Home Ubuntu Business Devices Cloud Download Support Project Community Partners Shop

Desktop Server Cloud

Type to search

Pay what you think it's worth

Millions agree that Ubuntu is a great piece of software. But how much do you really think it's worth?

Make the desktop more amazing	\$ 2
Performance optimisation for games and apps	\$ 2
Improve hardware support on more PCs	\$ 2
Phone and tablet versions of Ubuntu	\$ 2
Community participation in Ubuntu development	\$ 2
Better coordination with Debian and upstreams	\$ 2
Better support for flavours like Kubuntu, Xubuntu, Lubuntu	\$ 2
Tip to Canonical – they help make it happen	\$ 2

The same price as
Peace, Love and Linux
t-shirt
\$20

Contribution
\$ 16

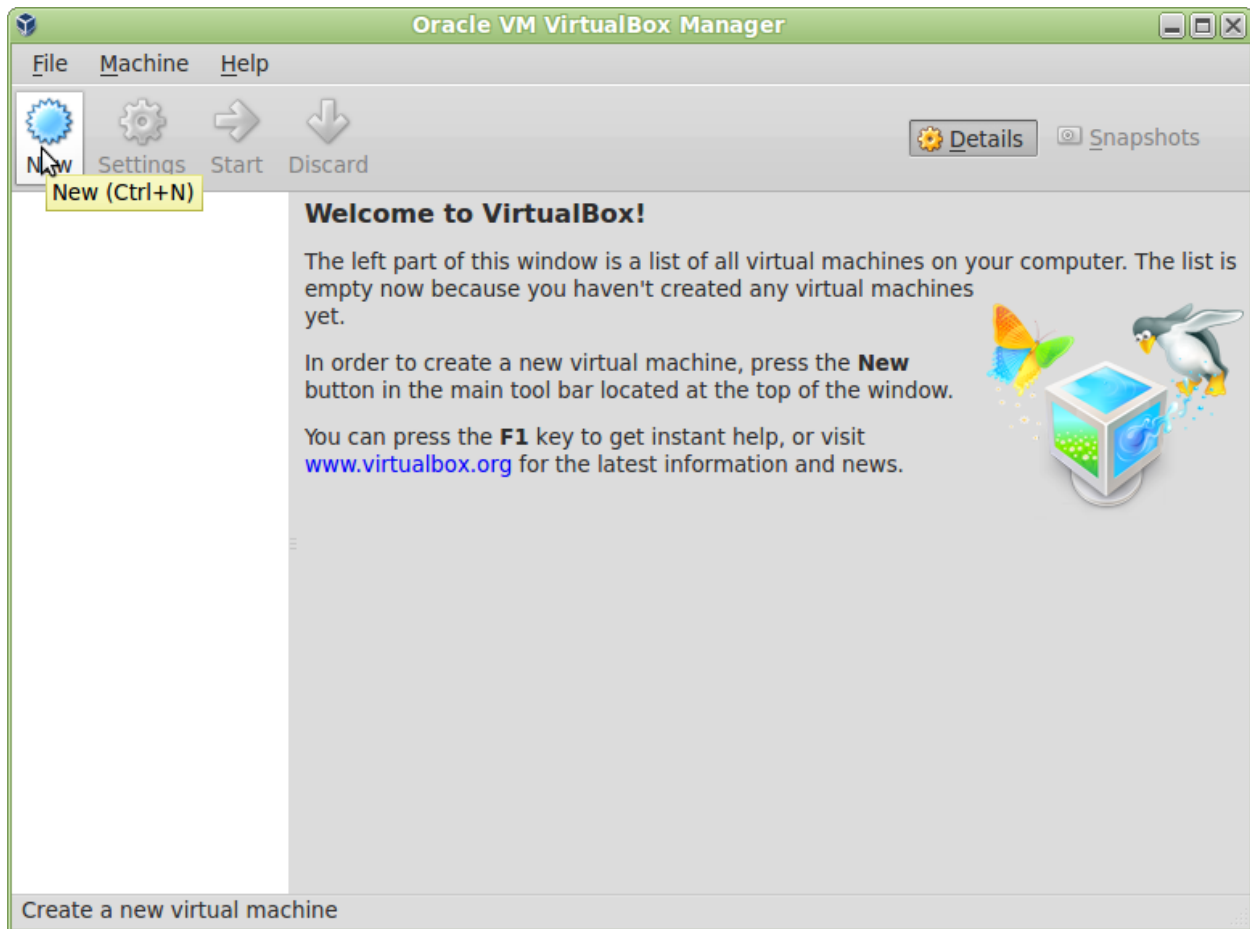
[Not now, take me to the download >](#) [Pay with PayPal](#)

El resultado de esta descarga debe ser un fichero con un nombre parecido a “ubuntu-12.04.1-desktop-i386.iso”.

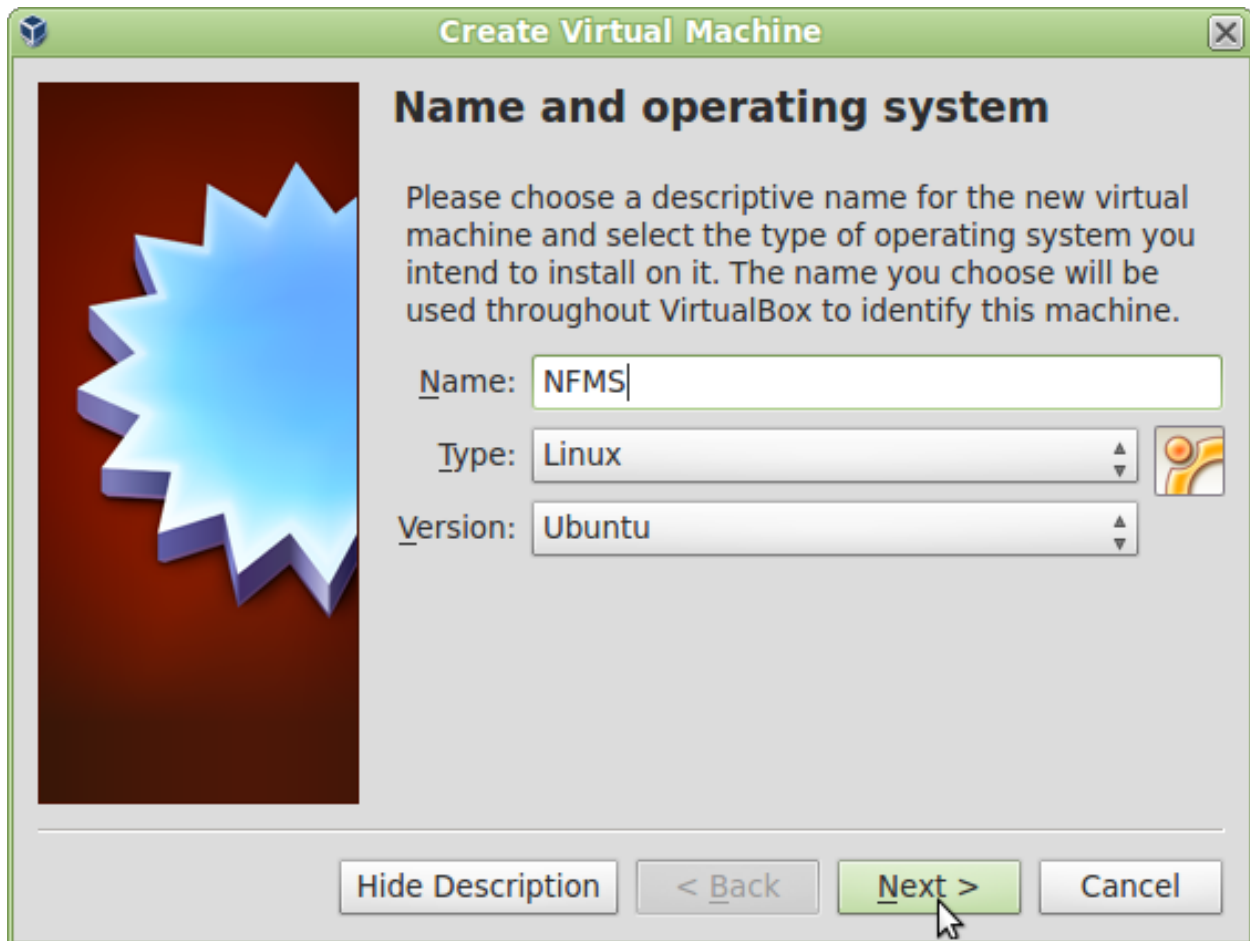
⁰ <http://www.ubuntu.com/download>

Creación de una máquina virtual

Una vez VirtualBox está instalado, se deberá arrancar y crear una nueva máquina virtual:

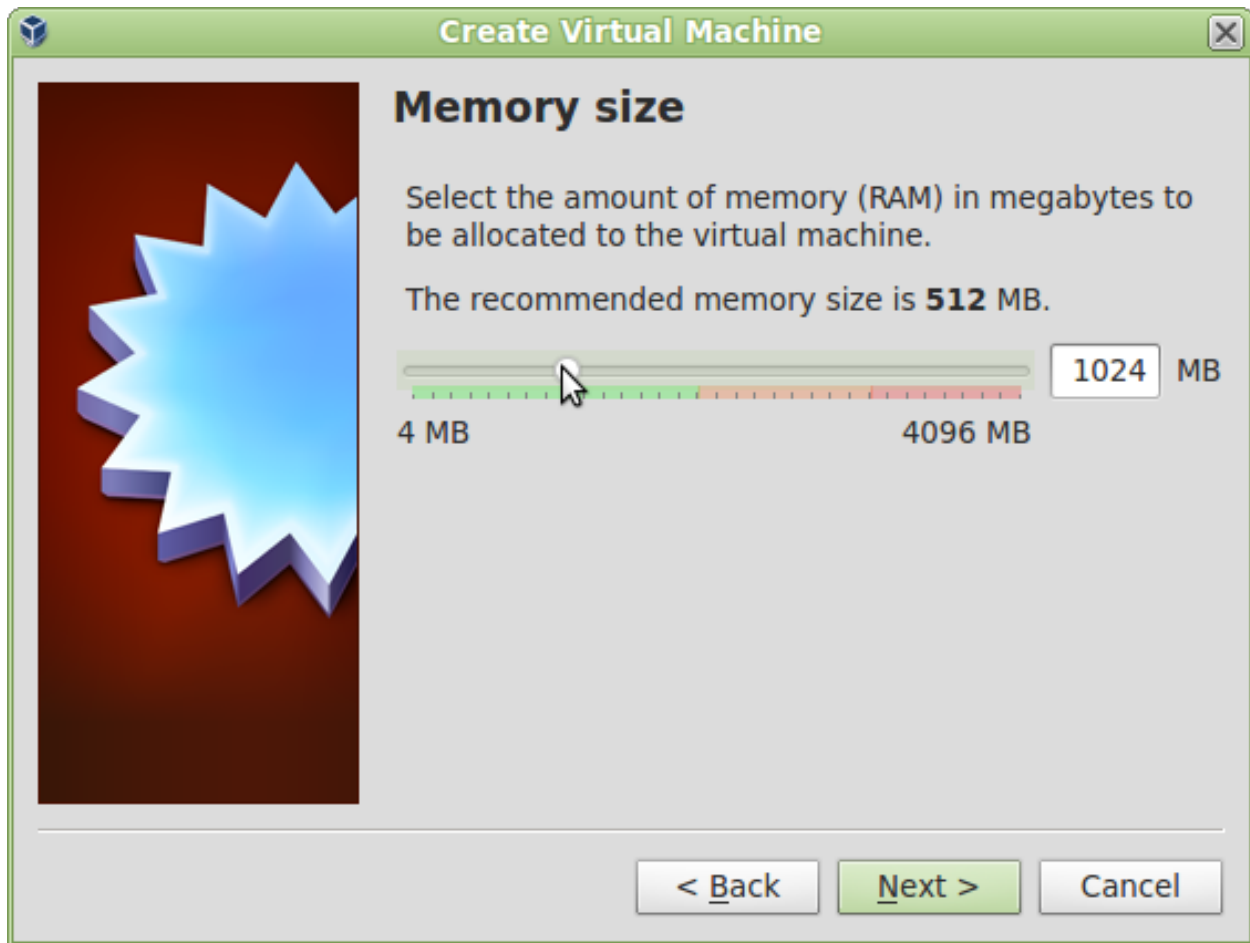


A continuación le damos un nombre a la máquina virtual y especificamos el sistema operativo “Linux”, “Ubuntu”.



Especificamos 1024Mb de memoria para la máquina virtual. Hay que tener en cuenta que esta memoria se toma de la máquina anfitriona por lo que si la máquina anfitriona tiene menos de 2048Mb, dar 1024Mb a la máquina virtual puede ser demasiado, ya que la anfitriona se puede quedar sin memoria.

Como regla general, lo deseable es 1024Mb pero en ningún caso debe sobrepasarse el 50 % de la memoria total de la máquina anfitriona.

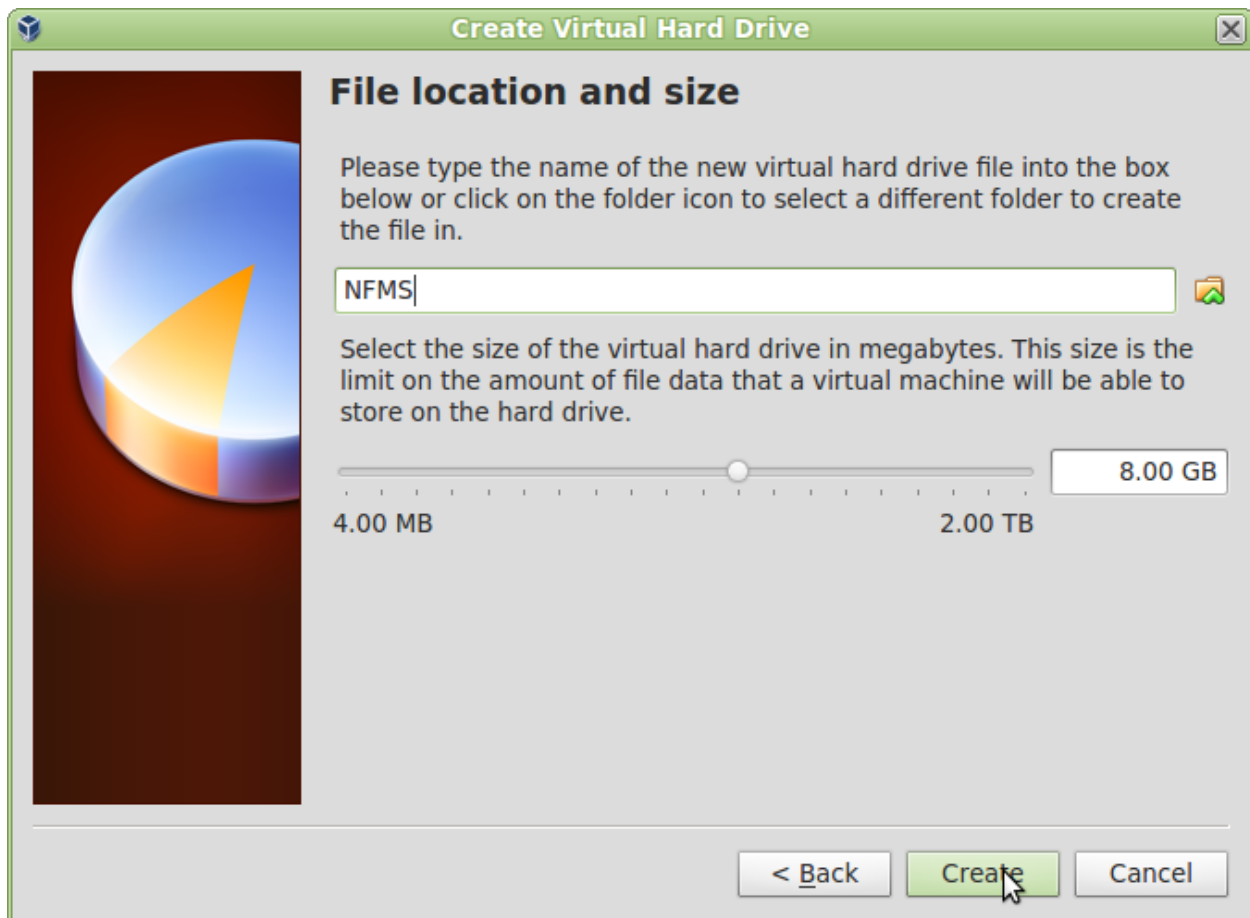


Por último sólo queda especificar el tamaño y tipo del disco, en el que dejaremos las opciones que vienen por defecto.

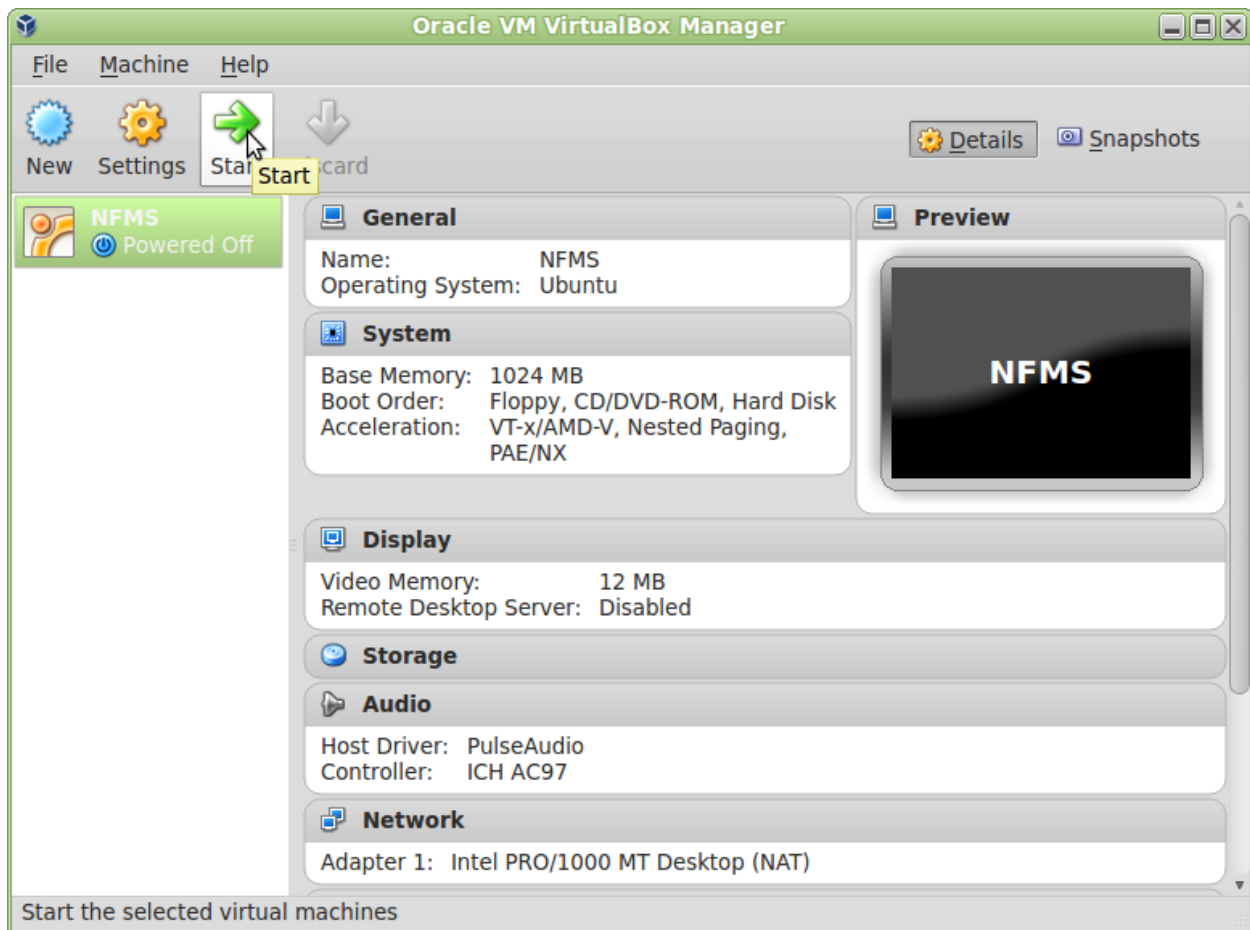




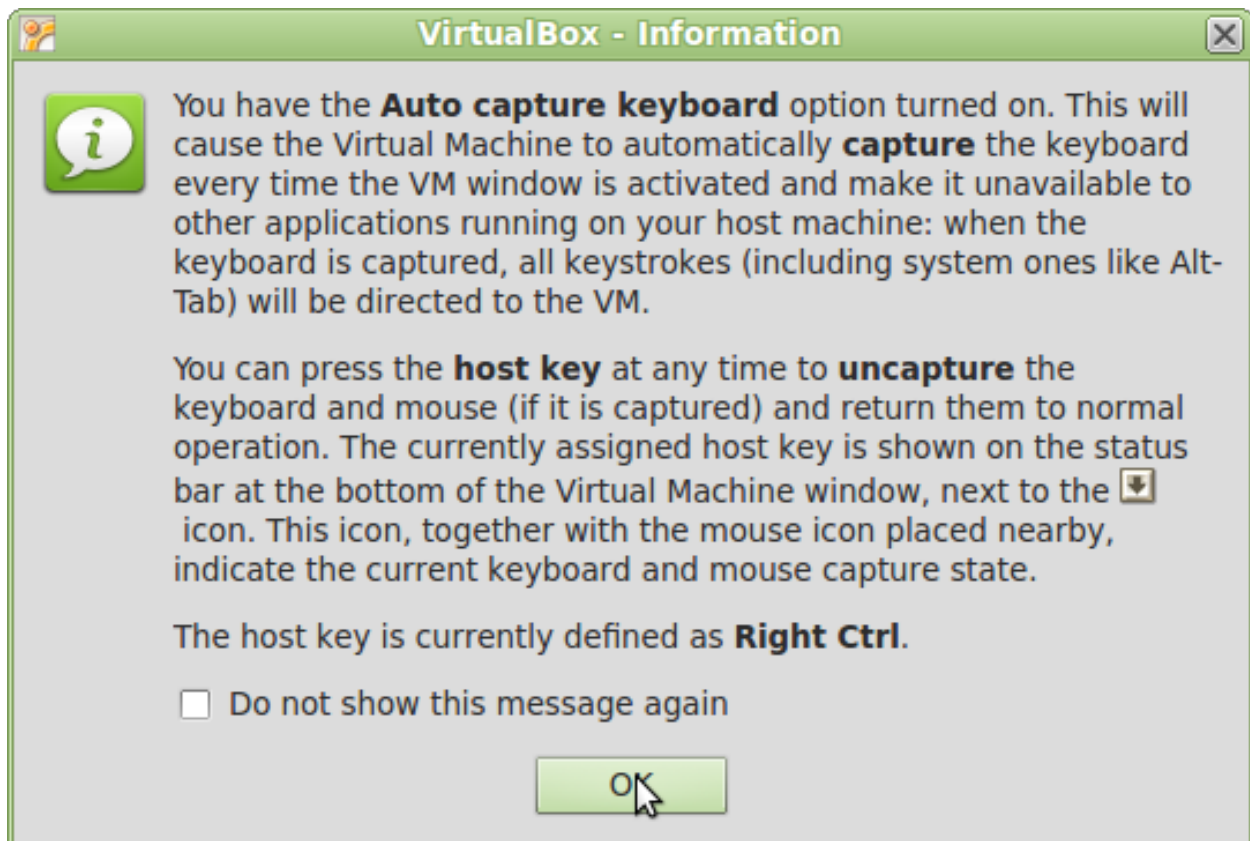




Ahora la máquina está creada y puede ser arrancada seleccionándola y pulsando el botón “Start”.



Al arrancar se ofrecen varios mensajes informativos que no son muy importantes. Uno de ellos informa sobre la “tecla anfitriona”. Cuando se está trabajando en la máquina virtual y se pulsa dicha tecla, el software de virtualización quita el foco al sistema operativo y lo devuelve a la maquina anfitriona. La tecla por defecto es el “Control” derecho.



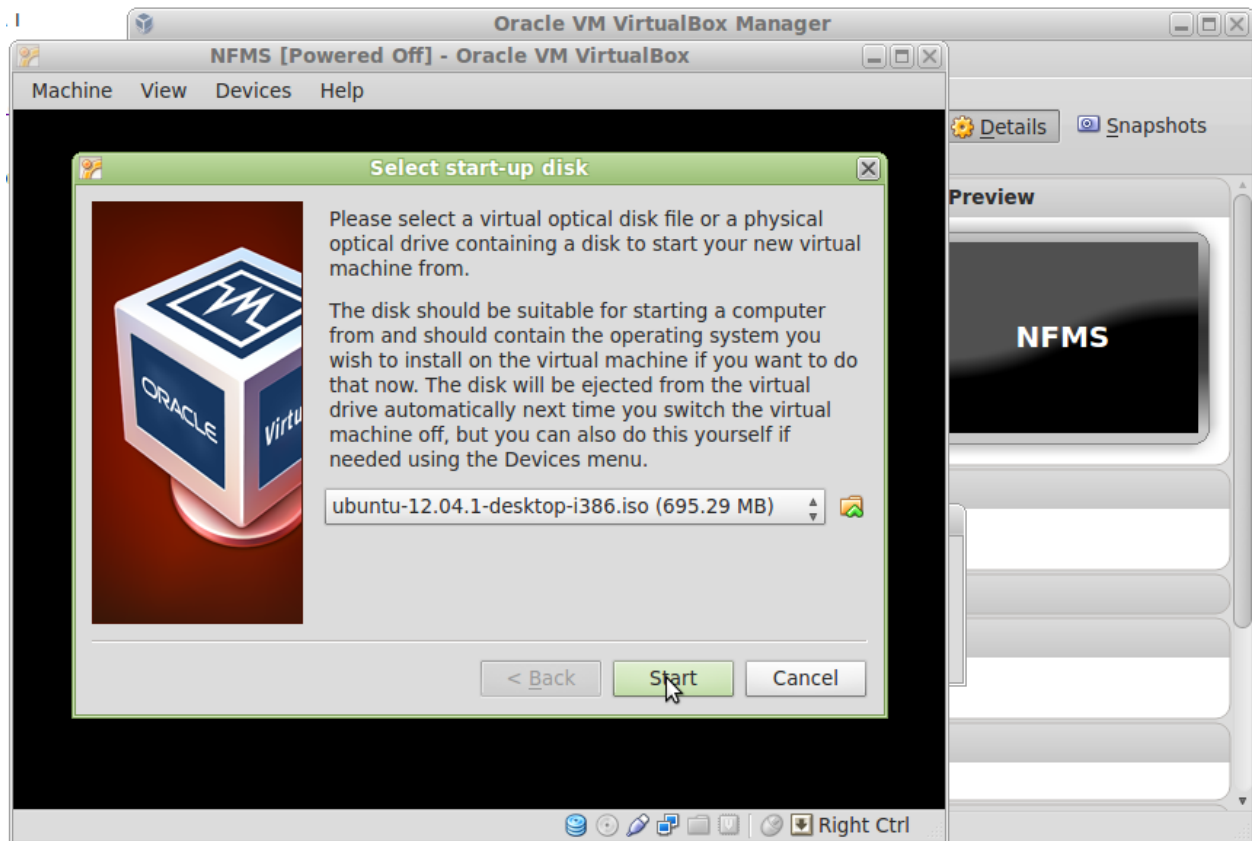
Instalación de Ubuntu/Linux

Lo siguiente que hay que hacer es instalar una versión de Ubuntu/Linux. El propio proceso de arranque de la máquina virtual pregunta a continuación dónde puede encontrar una imagen del sistema operativo.

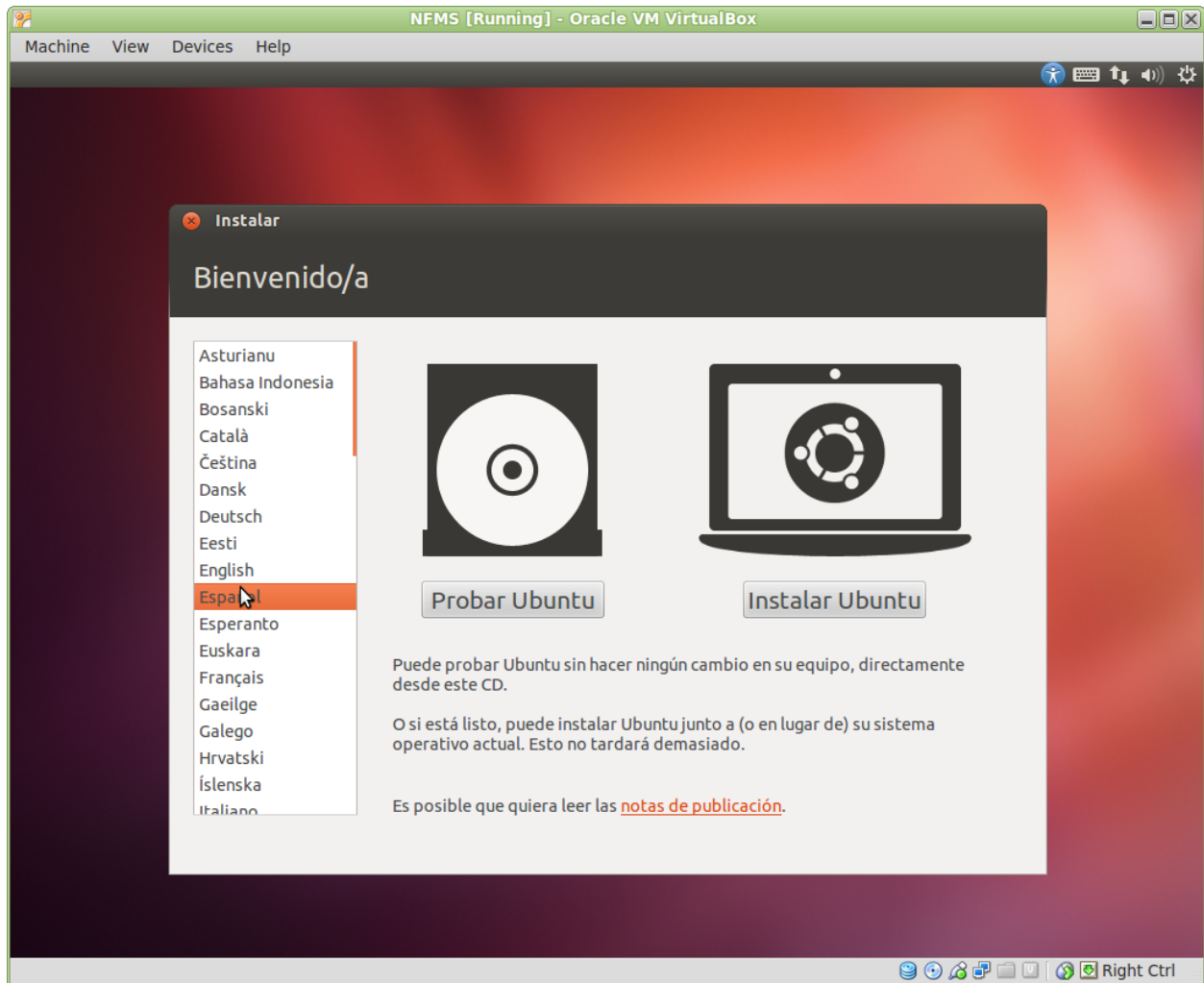
A continuación hay que pulsar el botón de la carpetita para seleccionar la imagen de Ubuntu que descargamos en el primer punto: ubuntu-12.04.1-desktop-i386.iso.



Y por último sólo queda pulsar el botón *Start* para comenzar la instalación.



Un aspecto importante es que la instalación se puede seguir en Español, seleccionándolo desde la lista de la izquierda.



CAPÍTULO 2

Introducción a Linux

Nota: ©2012 Fernando González Cortés y Miguel García Coya

Excepto donde quede reflejado de otra manera, la presente documentación se halla bajo licencia : Creative Commons (Creative Commons - Attribution - Share Alike: <http://creativecommons.org/licenses/by-sa/3.0/deed.es>)

En el símbolo de sistema presentado es posible hacer uso de una serie de comandos que la mayor parte de sistemas linux tienen. Pero antes de ver los comandos es importante tener claro cómo se organiza el sistema de ficheros y cómo se referencian estos mediante rutas relativas y absolutas.

El sistema de ficheros linux se organiza jerárquicamente a partir de un directorio llamado “raíz” y que se denota por la barra inclinada hacia adelante (/). En linux los ficheros se referencian mediante rutas. Estas rutas pueden ser relativas o absolutas. Las absolutas comienzan por /, mientras que las relativas empiezan por el nombre de un subdirectorio, por . (directorio actual) o por .. (directorio padre).

Así pues, podemos tener rutas absolutas como:

```
/tmp
/home/geo
/home/geo/Escritorio
etc.
```

Nota: En la documentación antepondremos el símbolo \$ a toda aquella instrucción que se puede ejecutar en la línea de comandos de un sistema Linux. ¡¡Pero dicho símbolo no forma parte de la instrucción!!

Las rutas absolutas se pueden utilizar desde cualquier directorio. Podemos listar los directorios anteriores con los siguientes comandos, independientemente del directorio en el que se esté:

```
$ ls /tmp
$ ls /home/geo
$ ls /home/geo/Escritorio
```

Las rutas relativas en cambio, parten del directorio actual. Si por ejemplo estamos en `/home/geo`, podemos listar los directorios anteriores con los siguientes comandos:

```
$ ls ../../tmp
$ ls .
$ ls Escritorio
```

o “navegando” de forma más caprichosa:

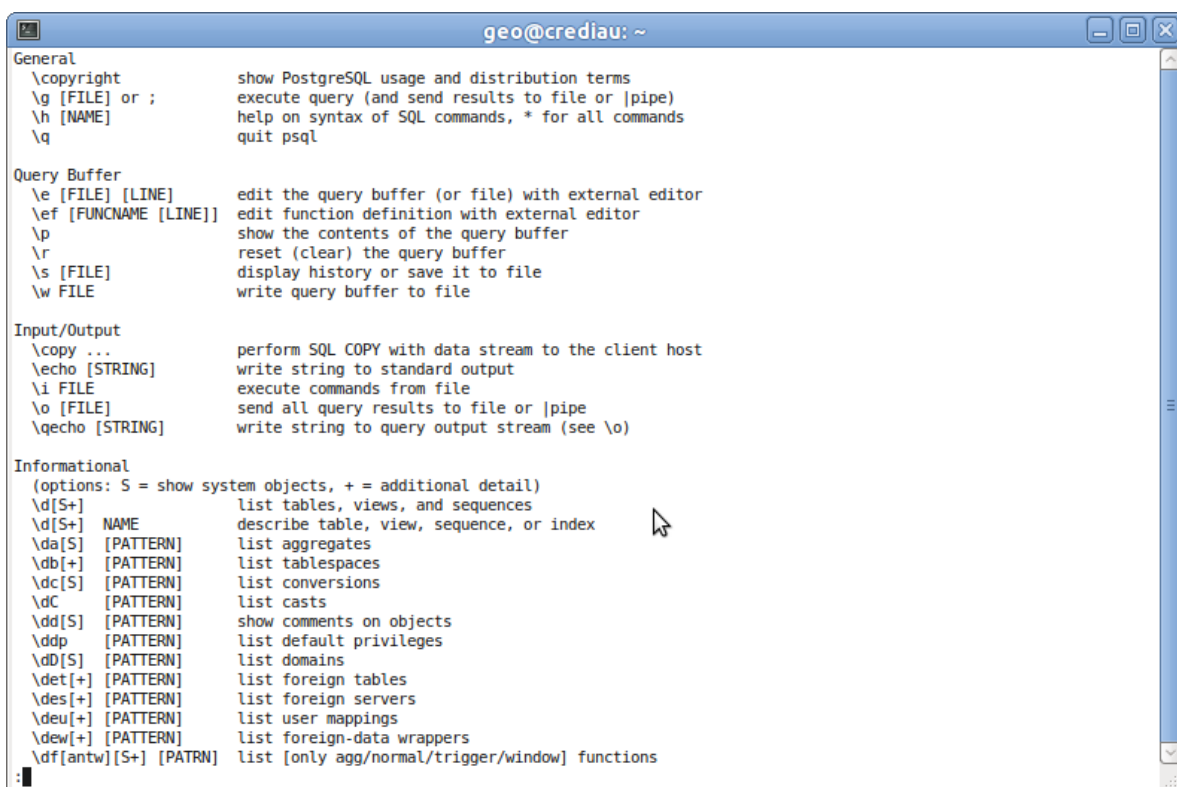
```
$ ls Escritorio/../../../../tmp
$ ls ../../../../../../../../../../geo
$ ls ../geo/Escritorio
```

A continuación mostramos algunos comandos útiles en linux:

- **less:** Visualiza un fichero de texto. La interacción es la misma que la descrita en el apartado “Ayuda de psql” anterior:

```
$ less ruta_fichero_a_visualizar
```

El fichero a visualizar se presenta de una manera muy común en los sistemas UNIX y que podemos identificar porque en la esquina inferior izquierda tenemos el signo de los dos puntos (:) seguido del cursor.



Podemos navegar por el contenido pulsando los cursores arriba y abajo, así como las teclas de página anterior y posterior.

También es posible hacer búsquedas utilizando el comando `/texto`. Una vez pulsamos intro, se resaltarán las coincidencias encontradas, como se puede ver en la siguiente imagen. Para navegar a la siguiente coincidencia es posible pulsar la tecla ‘n’ y para ir a la anterior Mayúsculas + ‘n’

```

geo@crediau: ~
\d[S+]      list tables, views, and sequences
\d[S+] NAME describe table, view, sequence, or index
\da[S] [PATTERN] list aggregates
\db[+] [PATTERN] list tablespaces
\dc[S] [PATTERN] list conversions
\dC [PATTERN] list casts
\dd[S] [PATTERN] show comments on objects
\ddp [PATTERN] list default privileges
\dD[S] [PATTERN] list domains
\det[+] [PATTERN] list foreign tables
\des[+] [PATTERN] list foreign servers
\deu[+] [PATTERN] list user mappings
\dew[+] [PATTERN] list foreign-data wrappers
\df[antw][S+] [PATRN] list [only agg/normal/trigger/window] functions
\df[+] [PATTERN] list text search configurations
\dfd[+] [PATTERN] list text search dictionaries
\dfp[+] [PATTERN] list text search parsers
\dfT[+] [PATTERN] list text search templates
\dg[+] [PATTERN] list roles
\di[S+] [PATTERN] list indexes
\dI [PATTERN] list large objects, same as \lo list
\dL[S+] [PATTERN] list procedural languages
\dN[S+] [PATTERN] list schemas
\do[S] [PATTERN] list operators
\dO[S+] [PATTERN] list collations
\dp [PATTERN] list table, view, and sequence access privileges
\drds [PATRN1 [PATRN2]] list per-database role settings
\ds[S+] [PATTERN] list sequences
\dt[S+] [PATTERN] list tables
\dT[S+] [PATTERN] list data types
\du[+] [PATTERN] list roles
\dv[S+] [PATTERN] list views
\dE[S+] [PATTERN] list foreign tables
\dx[+] [PATTERN] list extensions
\l[+] list all databases
\sF[+] FUNCNAME show a function's definition
\z [PATTERN] same as \dp
/list

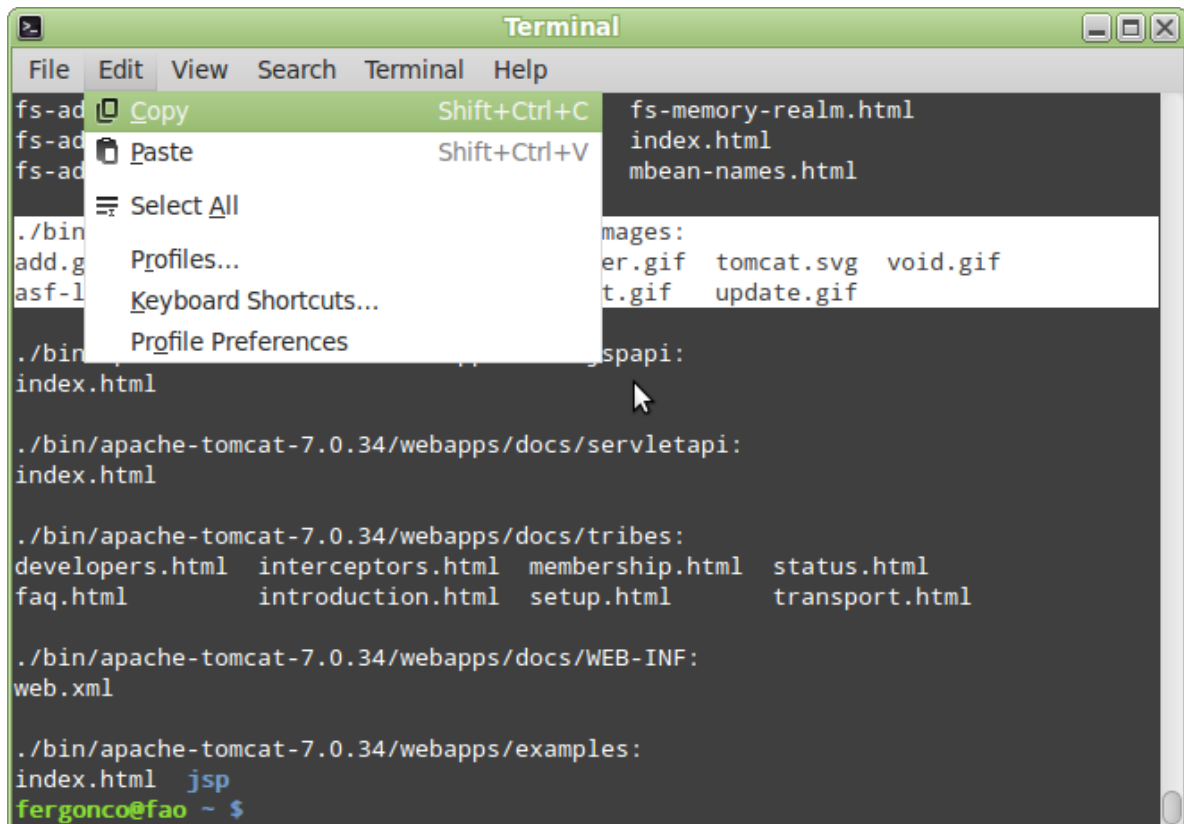
```

Para salir pulsar 'q'.

- nano: Permite editar ficheros. En la parte de abajo se muestran los comandos para salir, guardar el fichero, etc.:

```
$ nano ruta_fichero_a_editar
```

Cuando se trabaja en modo texto en linux, dependiendo de la aplicación terminal utilizada, es posible copiar y pegar texto de la forma habitual: seleccionando con el ratón y presionando una combinación de teclas. Sin embargo, esta combinación de teclas suele ser diferente a las habituales (Ctrl+C, Ctrl+V) ya que Ctrl+C tiene un significado distinto en el terminal: el de interrumpir el proceso que se está ejecutando. La combinación de teclas se puede averiguar si se utiliza un terminal con barra de menús como el siguiente:



Si la aplicación terminal que se utiliza no incorpora menú, como `xterm`, siempre se puede utilizar un método bastante cómodo y siempre disponible en Linux que consiste en seleccionar el texto y pegar directamente con el botón central del ratón. Lo engañoso de este método es que el texto se pega en la posición del cursor y no allí donde se pincha.

Ejercicio: Crear un fichero con tu nombre y que contenga este apartado.

- `locate`: Localiza ficheros en el sistema operativo:

```
$ locate parte_del_nombre_del_fichero
```

Un aspecto a tener en cuenta en el uso de `locate` es que el sistema programa escaneos regulares del disco para construir un índice con los ficheros existentes y permitir así a `locate` responder al usuario sin tener que realizar la búsqueda en el sistema de ficheros, que toma mucho tiempo. Es por esto que `locate` funciona muy rápido pero puede que no encuentre los ficheros creados recientemente. Para estos, habrá que esperar a que se produzca un escaneo programado o lanzar un escaneo de forma manual con `updatedb`.

- `find`: Localiza ficheros en el sistema de archivos:

```
$ find ruta -name nombre_del_fichero
```

A diferencia de `locate`, el comando `find` recorrerá el sistema de archivos cada vez que se lo ejecute, sin emplear índices. Por esa razón, si bien es mucho más lento el resultado, puede hallar ficheros que no se hayan indexado, por ejemplo, los ficheros creados recientemente.

- `id`: Muestra la identidad actual del usuario:

```
$ id
```


- **su:** Permite autenticarse con un usuario distinto. El siguiente comando probablemente no funcionará porque es necesario tener permisos de superusuario para realizar *su*, ver el siguiente caso:

```
$ su postgres
```

- **sudo:** No es un comando en sí, sino que permite ejecutar el comando que le sigue con permisos de superusuario. Por ejemplo, para ejecutar el comando anterior con permisos de superusuario:

```
$ sudo su postgres
```

- **passwd:** Cambia el password de un usuario. Por ejemplo para cambiar el password de root:

```
$ sudo passwd root
```

- **ssh:** Acceso remoto en línea de comandos. Con SSH es posible entrar a un servidor remoto que tenga activado dicho acceso. Para ello es necesario especificar la dirección del servidor:

```
$ ssh 168.202.48.151
The authenticity of host '168.202.48.151 (168.202.48.151)' can't be established.
ECDSA key fingerprint is 9f:7c:a8:9c:8b:66:37:68:8b:7f:95:a4:1b:24:06:39.
Are you sure you want to continue connecting (yes/no)? yes
```

En la salida anterior podemos observar como primeramente el sistema pregunta por la autenticidad de la máquina a la que queremos conectar. Tras responder afirmativamente el sistema nos comunica que el servidor al que vamos a conectarnos se añade a la lista de hosts conocidos, de manera que el mensaje anterior no volverá a aparecer la siguiente vez que se intente una conexión. A continuación el sistema pregunta el password del usuario “usuario”:

```
Warning: Permanently added '168.202.48.151' (ECDSA) to the list of known hosts.
usuario@168.202.48.151's password:
```

En caso de querer conectar con otro usuario es necesario prefijar el nombre de dicho usuario, seguido del carácter “@” antes de la dirección del servidor:

```
$ ssh otro_usuario@168.202.48.151
```

- **scp:** Copia ficheros al servidor:

```
$ scp fichero_origen directorio_destino
```

El directorio puede ser una ruta normal o la cadena de conexión por SSH a un servidor remoto. Veamos varios ejemplos. El siguiente copia ficheros locales en el directorio */tmp* de un servidor remoto:

```
$ scp mi_fichero_local geo@geoportalcredia.org:/tmp
```

El siguiente comando copia el fichero de vuelta:

```
$ scp geo@geoportalcredia.org:/tmp/mi_fichero_local .
```

Se puede observar que el format de la URL remota es parecido al que se usa para conectar por cliente SSH. La única diferencia es que al final, separado por (:), encontramos una ruta en la máquina remota

Ejercicio: Conectar a una máquina linux usando estos comandos.

Ejercicio: Copiar el fichero creado en el apartado sobre *nano* en */tmp*

Ejercicio: Conectar al sistema linux desde windows y copiar un fichero cualquiera haciendo uso de *putty.exe* y *scp.exe*.

- zip: Comprime ficheros:

```
$ zip -r ruta_fichero.zip lista_de_ficheros_a_comprimir
```

La opción -r hace que zip incluya los contenidos de los directorios que se encuentre en la lista de ficheros a compartir.

- unzip: Descomprime ficheros:

```
$ unzip ruta_fichero.zip
```

- chgrp: cambia el grupo de usuarios de un archivo o directorio en sistemas tipo UNIX. Cada archivo de Unix tiene un identificador de usuario (UID) y un identificador de grupo (GID) que se corresponden con el usuario y el grupo de quien lo creó.

El usuario root puede cambiar a cualquier archivo el grupo. Los demás usuarios sólo pueden hacerlo con los archivos propios y grupos a los que pertenezca.:

```
$ chgrp nuevogrp archivo1 [ archivo2 archivo3...]
```

Cambia el grupo de archivo1 archivo2, etc. que pasará a ser nuevogrp

```
$ chgrp -R nuevogrp directorio
```

Cambia el grupo para que pase a ser nuevogrp a directorio, todos los archivos y
↳ subdirectorios contenidos en él, cambiándolos también de forma recursiva en
↳ todos archivos de los subdirectorios.

- chown: cambiar el propietario de un archivo o directorio:

```
$ chown nuevouser archivo1 [ archivo2 archivo3...]
```

```
$ chown -R nuevouser directorio
```

- chmod: permite cambiar los permisos de acceso de un archivo o directorio:

```
$ chmod [modificadores] permisos archivo/directorio
```

Ejercicio: Quitarse el permiso de lectura sobre el fichero creado en el apartado de nano.

- wget: Utilizado para descargar ficheros de distintos servidores HTTP, HTTPS y FTP. Basta con teclear wget seguido de la dirección del fichero en internet:

```
wget http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf
```

Entre las muchas opciones que soporta, la más frecuente es -O <nombre_fichero>, que permite dar un nombre distinto al fichero descargado:

```
wget http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf -O especificacion_  
↳ shapefile.pdf
```

Ejercicio: Descargar el logo del portal de FAO (<http://fao.org>) con wget

Aprovecharemos para explicar una funcionalidad de Linux que nos vendrán bien más adelante:

- Redireccionamiento: En linux es posible mediante el uso del caracter mayor que > redireccionar la salida de un proceso hacia otro proceso. Por ejemplo, podremos escribir en un archivo mediante:

```
Creamos el archivo vacio mediante el comando ``touch``  
$ touch nombre_del_archivo
```

```
`echo` nos permite escribir texto en la consola
$ echo "Estoy escribiendo en el archivo" > nombre_del_archivo
$ nano nombre_del_archivo
```

y podremos ver el texto “Estoy escribiendo en el archivo” en el archivo que hemos creado. Esta función será usada muy habitualmente para la generación de archivos sql con la salida de las operaciones.

La instalación de aplicaciones en Linux se hace mediante el uso de herramientas de gestión de paquetes. En función de la distribución que estemos utilizando utilizaremos una herramienta u otra dependiendo de la versión en la que se base nuestra distribución Debian, Red Hat... En nuestro caso, como nuestra versión se basa en la distribución Debian, utilizaremos la herramienta apt:

```
- Buscar aplicaciones::
```

```
$ apt-cache search nombre_aplicación
```

- Ver información sobre la aplicación:

```
$ apt-cache show nombre_aplicación
```

- Instalar aplicación:

```
$ apt-get install nombre_aplicación (con permisos de superusuario)
```

- Eliminar aplicación (con permisos de superusuario):

```
$ apt-get remove nombre_aplicación
$ apt-get purge nombre_aplicación
```

en este último caso eliminará también los archivos de configuración que haya instalado la aplicación.

La herramienta apt conoce los repositorios desde los que descargarse los archivos gracias a que se lo indicamos en un archivo, denominado `sources.list` que se encuentra en la carpeta `/etc/apt` de nuestra distribución. Si queremos añadirle un repositorio más podremos:

```
$ sudo echo "ruta al repositorio en la red" > /etc/apt/sources.list
```

después debemos actualizar los repositorios disponibles mediante:

```
$ apt-get update
```

Para una descripción completa de las operaciones que son posibles mediante apt-get se recomienda el uso de:

```
$ apt-get --help
```

Realizar el siguiente ejercicio:

1. Crear un fichero llamado `/tmp/copy-contents.sh` con las siguientes líneas (sustituyendo `<servidor>` y `<nombre>` por valores adecuados):

```
wget http://www.diva-gis.org/data/rrd/ARG_rrd.zip -O rails.zip
unzip rails.zip
scp * nfms@<servidor>:/tmp/<nombre>
```

2. Dar permisos de ejecución
3. Ejecutar

Ejercicio: Crear un fichero vacío en `/var/lib/postgresql`

De cuantas maneras es posible realizar esto?

1. Usando `sudo` para crear el fichero
2. Creando el fichero como `postgres`
3. Cambiando los permisos al directorio. ¡NO!

Teoría de **base de datos**

Nota:	Fecha	Autores
	1 Noviembre 2012	■ Micho García (micho.garcia@geomati.co)
	15 Octubre 2013	■ Jorge Arévalo(jorge.arevalo@geomati.co)

©2012 Micho García

Excepto donde quede reflejado de otra manera, la presente documentación se halla bajo licencia : Creative Commons (Creative Commons - Attribution - Share Alike: <http://creativecommons.org/licenses/by-sa/3.0/deed.es>)

Bases de datos, el enfoque general

La utilización de **base de datos** se ha extendido dando solución a problemas como

- Manejo de grandes volúmenes de datos
- Complejidad en la extracción de estos datos
- Concurrencia en el acceso a datos, accesos simultáneos por varios usuarios

Antes el almacenamiento y manejo de la información se realizaba mediante el uso de archivos, formatos tipo texto o archivos con estructuras internas (.dbf) permitían el manejo de esta información. Tenían limitaciones como

- Limitaciones en la cantidad de datos que era posible almacenar
- Rendimiento de lectura de estos archivos
- Bloqueo de los archivos con el acceso por usuario
- Imposibilidad de gestionar el versionado de manera sencilla

Gracias al desarrollo de la tecnología se democratiza el uso de ordenadores potentes que permiten poner a disposición de las organizaciones equipos potentes que gestionen de manera eficiente las **base de datos** mediante Sistemas gestores de bases de datos (SGBD).

Una base de datos es

- Una gran masa de datos relacionados entre si pertenecientes a un mismo contexto
- Colección estructurada almacenada en un sistema informático

Objetivo

- Aportar a la organización a la que sirve la información necesaria

Funciones

- Recogida
- Almacenamiento
- Procesamiento
- Recuperación

Propiedades

- Estructuradas de manera independiente de las aplicaciones y del soporte de almacenamiento que las contiene (SQL)
- Presentan la menor redundancia posible
- Son compartidas por todos los usuarios de una red

Así de esta manera podremos definir unos **Objetivos generales de la base de datos**

- Abstracción de la información
- Independencia
- Redundancia mínima
- Consistencia
- Seguridad
- Integridad
- Respaldo y recuperación
- Control de la concurrencia, versionado
- Tiempo de respuesta

Debemos diferenciar entre **base de datos** y **SGBD**. La primera se encarga del almacenamiento propiamente dicho y el **SGBD** de la manipulación de la información contenida en la base de datos. Una **base de datos** asimismo contendrá no solo los datos propios, sino que puede almacenar consultas sobre estos datos, vistas, informes...

El **modelo de datos** es el encargado de reflejar mediante un conjunto de REGLAS y CONCEPTOS la estructura de datos y operaciones aplicables sobre estos datos. Se trata de una abstracción de la realidad. Permite definir el tipo de datos que hay en la **base de datos** y la forma en que se relacionan. Además aplica restricciones entre estos datos, condiciones que deben cumplir estos para reflejar la realidad. Por último se definen en ellos las operaciones de manipulación de los datos de la **base de datos**

Existen modelos de datos jerárquicos, de red, orientados a objetos... Nosotros estudiaremos el Modelo de datos relacional, por ser el más ampliamente utilizado para el modelado de la realidad. Desarrollado en 1970 por Edgar Frank Codd se ha consolidado como el paradigma de los modelos de datos. Una base de datos relacional es un conjunto de una o más tablas estructuradas en registros (líneas) y campos (columnas), que se vinculan entre sí por un campo en

común, en ambos casos posee las mismas características como por ejemplo el nombre de campo, tipo y longitud; a este campo generalmente se le denomina ID, identificador o clave. A esta manera de construir bases de datos se le denomina modelo relacional y está implementado en los **SGBD** relacionales, como por ejemplo PostgreSQL.

Tablas, columnas, registros

Dentro del modelo de datos relacional los conceptos básicos con las que comenzar serán

- Tablas
- Columnas
- Registros
- Relaciones

Para llegar a comprender la necesidad de estos debemos partir del deseo de almacenar una información determinada, unos datos. Los **datos** serían la información que deseamos almacenar. Un dato puede ser

- El área de un parque natural
- El nombre de un parque natural
- La dirección de una oficina de correos
- El número de empleados de la oficina de correos
- El nombre de un accidente geográfico
- Las coordenadas de un accidente geográfico
- ...

Cualquier hecho conocido que pueda registrarse y que tenga un significado implícito. Una **entidad** es todo aquello de lo cual nos interesa guardar **datos**, por ejemplo

- Parques naturales
- Oficinas de correos
- Accidentes geográficos
- ...

Práctica 1

Defina la estructura de una tabla para los Parques Naturales de su país. Para ello detecte la información necesaria susceptible de ser almacenada y estrúcturela en una tabla definiendo el nombre de los campos.

Modelización de base de datos

Para seguir adelante con el **modelo relacional** antes necesitamos definir algunos conceptos más

Entidad

Por entidad entendemos un objeto del mundo real que podemos distinguir del resto de objetos y del que nos interesan algunas propiedades.

En el **modelo relacional**, se puede observar que estas entidades se formarán por atributos o campos referidos a un mismo tema que interesa almacenar. Una entidad debe definir cualquier objeto real o abstracto (que pueda ser pensado) y acerca del cual queremos guardar información. Se representan mediante rectángulos en el **modelo relacional**

Una entidad se correspondería en el **modelo relacional** con una tabla. La tabla a su vez estará formada por filas y columnas que serán

- FILAS serían cada unidad necesaria de almacenamiento, que se corresponden con los REGISTROS de la tabla
- COLUMNAS que se corresponden con los CAMPOS, unidad mínima de información, donde podríamos almacenar cada dato referente a una propiedad del REGISTRO

Mediante este sencillo esquema podremos definir en nuestro sistema las entidades mínimas necesarias para almacenar información.

Ejemplo de tablas:

```
TABLA -> ENTIDAD -> PARQUE NATURAL
FILAS -> REGISTRO -> Parque Nacional de
COLUMNAS -> CAMPO -> 8° 33' N 83° 35' O
```

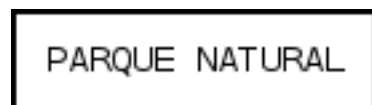
Ejemplos de entidad

Algunos ejemplos de entidad son un empleado, un producto o un despacho. También son entidades otros elementos del mundo real de interés, menos tangibles pero igualmente diferenciados del resto de objetos; por ejemplo, una asignatura impartida en una universidad, un préstamo bancario, un pedido de un cliente, etc.

El término entidad se utiliza tanto para denominar objetos individuales como para hacer referencia a conjuntos de objetos similares de los que nos interesan los mismos atributos; es decir, que, por ejemplo, se utiliza para designar tanto a un empleado concreto de una empresa como al conjunto de todos los empleados de la empresa. Más concretamente, el término entidad se puede referir a instancias u ocurrencias concretas (empleados concretos) o a tipos o clases de entidades (el conjunto de todos los empleados).

El modelo **ER** proporciona una notación diagramática para representar gráficamente las entidades y sus atributos:

- Las entidades se representan con un rectángulo. El nombre de la entidad se escribe en mayúsculas dentro del rectángulo.



Ejemplo de Entidad:

```
PARQUE NATURAL -> Entidad
OFICINA CORREO -> Entidad
ACCIDENTE GEOGRÁFICO -> Entidad
```

Entidad débil

Una entidad débil es una entidad cuyos atributos no la identifican completamente, sino que sólo la identifican de forma parcial. Esta entidad debe participar en una interrelación que ayuda a identificarla.

Una entidad débil se representa con un rectángulo doble, y la interrelación que ayuda a identificarla se representa con una doble línea.

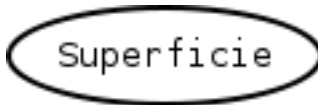


Ejemplo entidad débil:

```
Curso -> Profesor
Localidad -> Provincia
```

Dominio y valor

El conjunto de posibles valores que puede tomar una cierta característica se denomina **dominio**

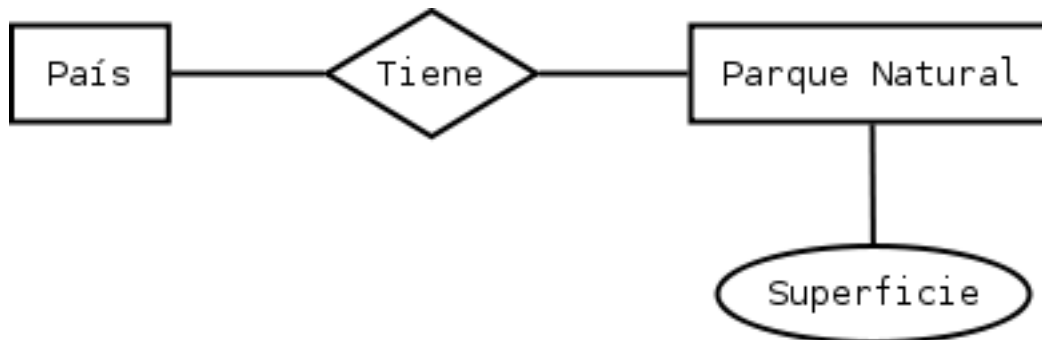


Ejemplo de dominio:

```
Inglés pertenece al dominio de Idiomas
33000ha pertenece al dominio de unidades de medida de superficie
```

Atributo

Cada una de las propiedades o características que tiene un tipo de entidad o un tipo de relación se denomina **atributo**; estos toman valores de uno o varios dominios.



Dentro del **modelo relacional** podremos encontrar atributos multivaluados y también opcionales.

- Atributo multivaluado: atributos de una entidad que pueden tener más de un valor.
- Atributo optativo: aquel que puede admitir valores nulos
- Atributo identificador: Uno o más campos cuyos valores son únicos en cada ejemplar de una entidad
 1. Deben distinguir a cada ejemplar teniendo en cuenta las entidades que utiliza el modelo
 2. Todos los ejemplares de un entidad deben tener el mismo identificador
 3. Cuando un atributo es importante aun cuando no tenga entidad concreta asociada, entonces se trata de una entidad y no de un atributo

Ejemplo de atributo:

```
Parque Natural -> Superficie
Parque Natural -> Nombre
Parque Natural -> Teléfono
```

Ejemplo de atributo multivaluado:

```
Idiomas de un curso -> Inglés, francés...
```

Restricciones

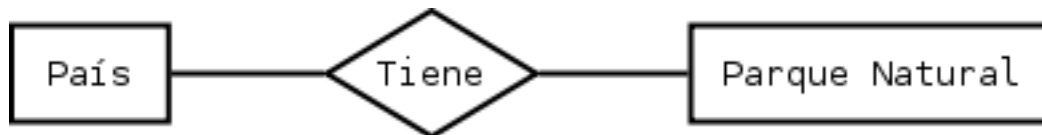
Se trata de limitaciones en las estructuras y en los datos impuestas por el propio modelo o por el desarrollador del modelo. Estas solo deben darse entre las entidades del modelo, nunca entre las relaciones. El modelo obliga a que las entidades tengan un identificador. El uso de dominios se puede considerar una restricción sobre los valores. Además existen restricciones estructurales.

Ejemplo restricción:

```
* Restricción de dominio::
    * Un trabajador de Correos no puede tener un sueldo menor a 75000 colones
* Integridad referencial::
    * Si cierra Correos no puede quedar ninguna Oficina en la base de datos
```

Relación

Esta se define como la asociación, vinculación o correspondencia entre entidades. Pueden existir mas de una relación entre entidades.

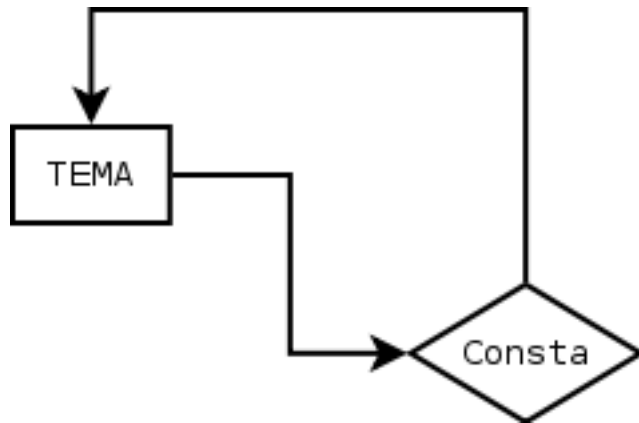


Ejemplo de interrelación:

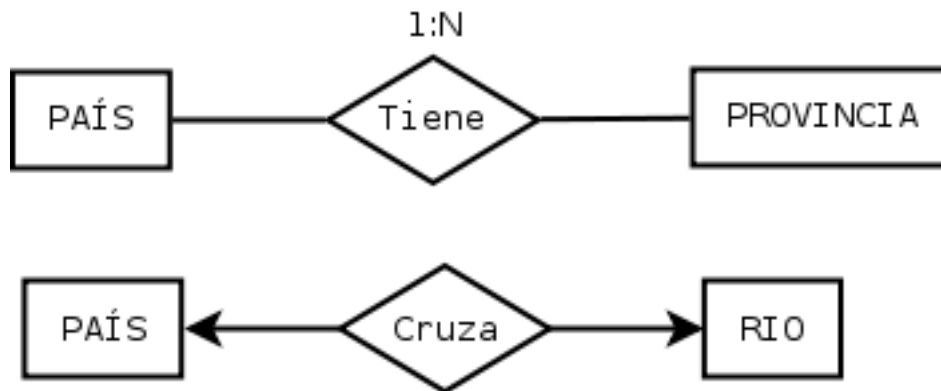
```
País -> tiene -> Parque Natural
```

En una relación se pueden definir los siguientes elementos:

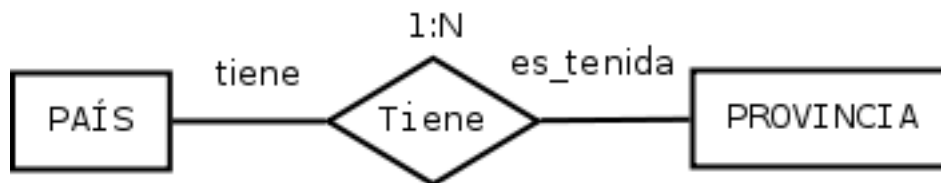
- Nombre, es el valor por el que se distingue del resto. En la representación gráfica se correspondería con la etiqueta incluida en el rombo que representa la relación. Aporta semántica al **modelo relacional**
- Grado, se trata del número de entidades que participan en un tipo de relación. Será de grado 2 (o binaria) cuando asocia dos tipos de entidad. Para las relaciones de grado 2 puede existir un caso particular que son las **reflexivas o recursivas**, en las cuales una entidad se asocia consigo misma.



- Tipo de correspondencia, es el número máximo de ejemplares que pueden estar asociados, en una determinada relación, con un ejemplar de otro tipo. Para representarlo graficamente se pone una etiqueta 1:1, 1:N o N:M en el lado de la relación que corresponda o bien se orienta el arco de la unión en el sentido 1 a N mediante una flecha

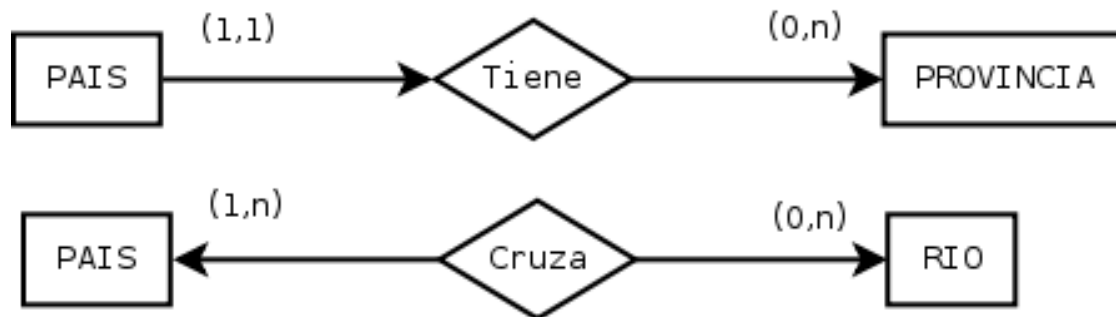


- Papel (“rol”), la función que cada uno de los tipos de entidad realiza en la relación. Se representa poniendo el nombre del **papel** en el arco de cada entidad



Cardinalidad de un tipo de entidad

Se define como el número mínimo y máximo de ejemplares de un tipo de entidad que pueden estar interrelacionadas con un ejemplar del otro, u otros tipos de entidad que participan en el tipo de relación. Se representará graficamente mediante un etiqueta del tipo (0,1), (1,1), (0,N) o (1,N).



Atributos de las relaciones

Se puede dar el caso de que existan atributos para las relaciones. Cuando esto se da en una relación 1:N este atributo debe llevarse a la entidad de cardinalidad máxima. En el caso de relaciones 1:1 o N:M el atributo se mantiene en la relación

Ejemplo de atributos en relación:

```

1:N Curso -> Tiene (Fecha_imparte) -> Edición = Curso -> Tiene -> Edición (Fecha_
->imparte)
1:1 Hombre -> Matrimonio (Fecha) -> Mujer
  
```

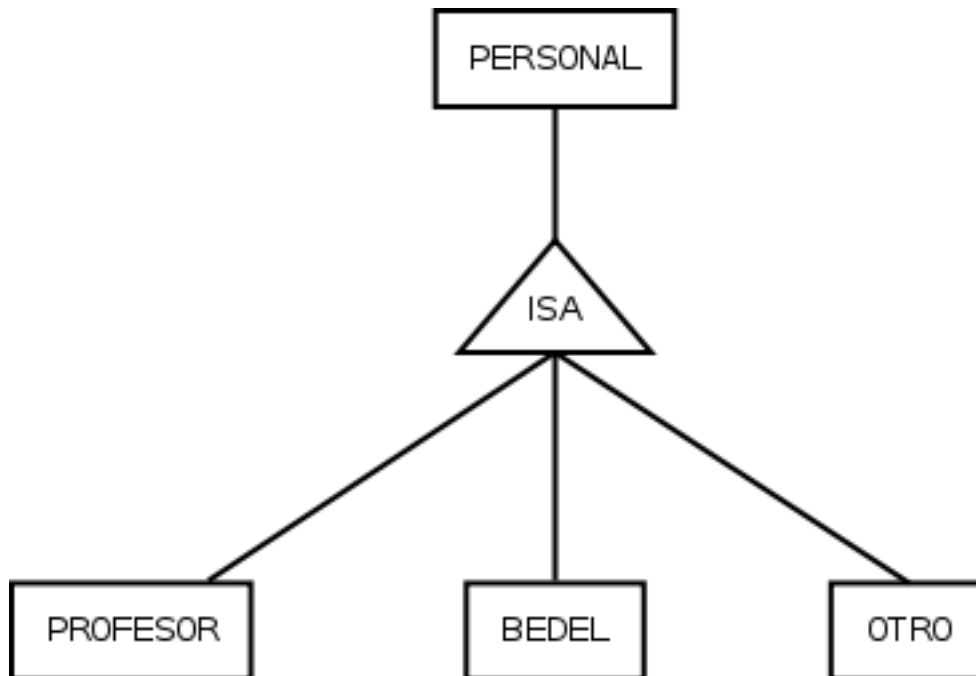
Generalización/Especialización

Entidades is a

Un tipo de entidad **is a** es aquella que se descompone en entidades especializadas. Existen dos tipos de entidades **is a**: **especializaciones** y **generalizaciones**.

Se denomina **especialización** se trata de entidades que se pueden dividir en entidades más concretas. La entidad general comparte con las especializadas sus atributos. Se detecta cuando hay ejemplares para los que no tienen sentido algunos de los atributos mientras que otros si.

La **generalización** es si se agrupan varias entidades en una o mas entidades generales. Se observa generalización si en varias entidades existen atributos iguales.



En estas relaciones se puede hablar de herencia en los atributos, superentidad y subentidad. Mediante un círculo en la superentidad indicaremos que esta es optativa.

También podemos indicar exclusividad, mediante un arco que cruce las líneas de relación. De esta manera indicaremos que la subentidad debe ser única.

Normalización

El proceso de normalización de bases de datos consiste en aplicar una serie de reglas a las relaciones obtenidas tras el paso del modelo entidad-relación al modelo relacional. Las bases de datos relacionales se normalizan para:

- Evitar la redundancia de los datos.
- Evitar problemas de actualización de los datos en las tablas.
- Proteger la integridad de los datos.

Modelización

1. Encontrar entidades (conjuntos de entidades)
2. Identificar atributos de las entidades
3. Buscar identificadores
4. Especificar las relaciones y cardinalidades
5. Identificar entidades débiles
6. Especializar y generalizar entidades donde sea posible

Ejercicio práctico

Se desea realizar una base de datos que gestione equipos de Catastro. Estos equipos están desplegados por todo el país. El país está dividido en departamentos que están divididos en municipios y estos a su vez en barrios. Las departamentos, municipios y barrios además de un identificador deberán guardar su nombre. Cada equipo tiene asignada una zona en la se pueden incluir varios barrios, pero ningún barrio podrá estar en dos zonas diferentes como tampoco podrá haber una zona en la que trabajen dos equipos. Las zonas se denominan por un número que a su vez hace de identificador. Los equipos vendrán identificados por su nombre. Cada equipo tiene un responsable que es el encargado de gestionarlo. Los trabajadores que son encargados no realizarán trabajo de campo. Se deben poder almacenar los datos personales de los usuarios y se identificarán por su numero sanitario. Los usuarios de un equipo podrán participar en campañas en diferentes zonas de ese mismo equipo de manera simultanea.

Advertencia: RESPUESTA

Referencias

Restricciones a la Base de Datos: Integridad y seguridad <http://s3.amazonaws.com/UNED/apuntes/Tema6.pdf>

Bases de datos http://es.wikipedia.org/wiki/Base_de_datos

Modelos de datos relacional http://es.wikipedia.org/wiki/Modelo_relacional

Implantación de sistemas informáticos de gestión. Bases de datos <http://www.slideshare.net/johntoasa2010/teoria-de-base-de-datos>

Teoría de bases de datos <http://si.ua.es/es/documentos/documentacion/office/access/teoria-de-bases-de-datos.pdf>

Diseño conceptual de bases de datos <http://www.jorgesanchez.net/bd/disenioBD.pdf> | <http://www.jorgesanchez.net/bd/index.html> | <http://www.jorgesanchez.net/bd/ejercicioser.html>

Diseño de bases de datos relacionales Adoración de Miguel, Mario Pattini y Esperanza Marcos. Editorial Ra-Ma

Entidades débiles <http://www.dataprix.com/217-entidades-debiles>

ACID <http://es.wikipedia.org/wiki/ACID>

Conceptos básicos de SQL

Nota:	Fecha	Autores
	1 Noviembre 2012	■ Micho García (micho.garcia@geomati.co)
	15 Octubre 2013	■ Jorge Arévalo(jorge.arevalo@geomati.co)

©2012 Micho García

Excepto donde quede reflejado de otra manera, la presente documentación se halla bajo licencia : Creative Commons (Creative Commons - Attribution - Share Alike: <http://creativecommons.org/licenses/by-sa/3.0/deed.es>)

Atención: Pueden descargarse los datos para los ejercicios desde aquí
--

Introducción

El lenguaje de consulta estructurado o SQL (por sus siglas en inglés **Structured Query Language**) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas. Una de sus características es el manejo del álgebra y el cálculo relacional que permiten efectuar consultas con el fin de recuperar de forma sencilla información de interés de bases de datos, así como hacer cambios en ella.

El SQL es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales y permite así gran variedad de operaciones.

Componentes del SQL

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

Comandos

Existen tres tipos de comandos SQL:

Los **DLL(Data Definition Language)** que permiten crear y definir nuevas bases de datos, campos e índices. Los **DML(Data Manipulation Language)** que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos. Los **DCL(Data Control Language)** que se encargan de definir las permisos sobre los datos

Lenguaje de definición de datos (DDL)

Comando	Descripción
CREATE	Utilizado para crear nuevas tablas, campos e índices
DROP	Empleado para eliminar tablas e índices
ALTER	Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

El lenguaje de definición de datos (en inglés Data Definition Language, o DDL), es el que se encarga de la modificación de la estructura de los objetos de la base de datos. Incluye órdenes para modificar, borrar o definir las tablas en las que se almacenan los datos de la base de datos. Existen cuatro operaciones básicas: CREATE, ALTER, DROP y TRUNCATE.

CREATE

Este comando crea un objeto dentro del gestor de base de datos. Puede ser una base de datos, tabla, índice, procedimiento almacenado o vista.

Ejemplo (crear una tabla):

```
# CREATE TABLE Empleado
(
  id serial NOT NULL PRIMARY KEY,
  Nombre VARCHAR(50),
  Apellido VARCHAR(50),
  Direccion VARCHAR(255),
  Ciudad VARCHAR(60),
  Telefono VARCHAR(15),
  Peso VARCHAR(5),
  Edad integer,
  Actividad VARCHAR(100),
  idCargo integer
)
```


ALTER

Este comando permite modificar la estructura de un objeto. Se pueden agregar/quitar campos a una tabla, modificar el tipo de un campo, agregar/quitar índices a una tabla, modificar un trigger, etc.

Ejemplo (agregar columna a una tabla):

```
# ALTER TABLE 'NOMBRE_TABLA' ADD NUEVO_CAMPO INT;
# ALTER TABLE 'NOMBRE_TABLA' DROP COLUMN NOMBRE_COLUMNNA;
```

DROP

Este comando elimina un objeto de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier otro objeto que el motor de la base de datos soporte. Se puede combinar con la sentencia ALTER.

Ejemplo:

```
# DROP TABLE 'NOMBRE_TABLA';
# DROP SCHEMA 'ESQUEMA';
# DROP DATABASE 'BASEDATOS';
```

TRUNCATE

Este comando trunca todo el contenido de una tabla. La ventaja sobre el comando DROP, es que si se quiere borrar todo el contenido de la tabla, es mucho más rápido, especialmente si la tabla es muy grande. La desventaja es que TRUNCATE sólo sirve cuando se quiere eliminar absolutamente todos los registros, ya que no se permite la cláusula WHERE. Si bien, en un principio, esta sentencia parecería ser DML (Lenguaje de Manipulación de Datos), es en realidad una DDL, ya que internamente, el comando TRUNCATE borra la tabla y la vuelve a crear y no ejecuta ninguna transacción.

Ejemplo:

```
# TRUNCATE TABLE 'NOMBRE_TABLA';
```

Lenguaje de manipulación de datos DML(Data Manipulation Language)

Comando	Descripción
SE- LECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado
IN- SERT	Utilizado para cargar lotes de datos en la base de datos en una única operación.
UP- DA- TE	Utilizado para modificar los valores de los campos y registros especificados Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.
DE- LE- TE	Utilizado para eliminar registros de una tabla

Definición

Un lenguaje de manipulación de datos (Data Manipulation Language, o DML en inglés) es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado. El lenguaje de manipulación de datos más popular hoy día es SQL, usado para recuperar y manipular datos en una base de datos relacional.

INSERT

Una sentencia INSERT de SQL agrega uno o más registros a una (y sólo una) tabla en una base de datos relacional.

Forma básica:

```
# INSERT INTO 'tabla' ('columna1', ['columna2,... ']) VALUES ('valor1', ['  
↪ 'valor2,... '])
```

Las cantidades de columnas y valores deben ser iguales. Si una columna no se especifica, le será asignado el valor por omisión. Los valores especificados (o implícitos) por la sentencia INSERT deberán satisfacer todas las restricciones aplicables. Si ocurre un error de sintaxis o si alguna de las restricciones es violada, no se agrega la fila y se devuelve un error.

Ejemplo:

```
# INSERT INTO agenda_telefonica (nombre, numero) VALUES ('Roberto Jeldrez', 4886850);
```

Cuando se especifican todos los valores de una tabla, se puede utilizar la sentencia acortada:

```
# INSERT INTO 'VALUES ('valor1', ['valor2,... '])
```

Ejemplo (asumiendo que 'nombre' y 'número' son las únicas columnas de la tabla 'agenda_telefonica'):

```
# INSERT INTO agenda_telefonica VALUES ('Jhonny Aguiar', 080473968);
```

UPDATE

Una sentencia UPDATE de SQL es utilizada para modificar los valores de un conjunto de registros existentes en una tabla.

Ejemplo:

```
# UPDATE mi_tabla SET campo1 = 'nuevo valor campo1' WHERE campo2 = 'N';
```

DELETE

Una sentencia DELETE de SQL borra uno o más registros existentes en una tabla.

Forma básica:

```
# DELETE FROM 'tabla' WHERE 'columna1' = 'valor1'
```

Ejemplo:

```
# DELETE FROM My_table WHERE field2 = 'N';
```

Clausulas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

Comando	Descripción
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros
GROUP BY	Utilizada para separar los registros seleccionados en grupos específicos
HAVING	Utilizada para expresar condición que debe satisfacer cada grupo
ORDER BY	Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico
WHERE	Utilizada para determinar los registros seleccionados en la clausula FROM

Operadores

Operadores Lógicos

Operador	Uso
AND	Es el “y” lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Es el “o” lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

Operadores de comparación

Operador	Uso
<	Menor que
>	Mayor que
≠	Distinto de
<=	Menor o igual que
>=	Mayor o igual que
BETWEEN	Intervalo
LIKE	Comparación
In	Especificar

Funciones de agregado

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

Comando	Descripción
AVG	Utilizada para calcular el promedio de los valores de un campo determinado
COUNT	Utilizada para devolver el número de registros de la selección
SUM	Utilizada para devolver la suma de todos los valores de un campo determinado
MAX	Utilizada para devolver el valor más alto de un campo especificado
MIN	Utilizada para devolver el valor más bajo de un campo especificado

Consultas

Consultas de selección

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos, esta información es devuelta en forma de conjunto de registros. Este conjunto de registros es modificable.

Básicas

La sintaxis básica de una consulta de selección es:

```
# SELECT Campos FROM Tabla;
# SELECT Nombre, Telefono FROM Clientes;
```

Ordenar los registros

Se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula **ORDER BY**:

```
# SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY Nombre;
```

Se pueden ordenar los registros por mas de un campo:

```
# SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY CodigoPostal, Nombre;
```

Y se puede especificar el orden de los registros: ascendente mediante la cláusula (**ASC** -se toma este valor por defecto) ó descendente (**DESC**):

```
# SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY CodigoPostal DESC ,
↳Nombre ASC;
```

Consultas con predicado

1. ALL Si no se incluye ninguno de los predicados se asume ALL. El Motor de base de datos selecciona todos los registros que cumplen las condiciones de la instrucción SQL:

```
# SELECT ALL FROM Empleados;
# SELECT * FROM Empleados;
```

2. TOP Devuelve un cierto número de registros que entran entre al principio o al final de un rango especificado por una cláusula ORDER BY. Supongamos que queremos recuperar los nombres de los 25 primeros estudiantes del curso 1994:

```
# SELECT TOP 25 Nombre, Apellido
FROM Estudiantes
ORDER BY Nota DESC;
```

Si no se incluye la cláusula ORDER BY, la consulta devolverá un conjunto_
↳arbitrario de 25 registros de la tabla Estudiantes .El predicado TOP no elige_
↳entre valores iguales. En el ejemplo anterior, si la nota media número 25 y la_
↳26 son iguales, la consulta devolverá 26 registros. Se puede utilizar la_
↳palabra reservada PERCENT para devolver un cierto porcentaje de registros que_
↳caen al principio o al final de un rango especificado por la cláusula ORDER BY._
↳Supongamos que en lugar de los 25 primeros estudiantes deseamos el 10 por_
↳ciento del curso::

```
# SELECT TOP 10 PERCENT Nombre, Apellido
FROM Estudiantes
ORDER BY Nota DESC;
```

3. DISTINCT Omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción SELECT se incluyan en la consulta deben ser únicos:

```
# SELECT DISTINCT Apellido FROM Empleados;
```

4. **DISTINCTROW** Devuelve los registros diferentes de una tabla; a diferencia del predicado anterior que sólo se fijaba en el contenido de los campos seleccionados, éste lo hace en el contenido del registro completo independientemente de los campos indicados en la cláusula **SELECT**:

```
# SELECT DISTINCTROW Apellido FROM Empleados;
```

Criterios de selección

Operadores Lógicos

Los operadores lógicos soportados por SQL son:

AND, OR, XOR, Eqv, Imp, Is y Not.

A excepción de los dos últimos todos poseen la siguiente sintaxis:

```
<expresión1> operador <expresión2>
```

En donde expresión1 y expresión2 son las condiciones a evaluar, el resultado de la operación varía en función del operador lógico:

```
# SELECT * FROM Empleados WHERE Edad > 25 AND Edad < 50;
# SELECT * FROM Empleados WHERE (Edad > 25 AND Edad < 50) OR Sueldo = 100;
# SELECT * FROM Empleados WHERE NOT Estado = 'Soltero';
# SELECT * FROM Empleados WHERE (Sueldo > 100 AND Sueldo < 500) OR (Provincia =
↳ 'Madrid' AND Estado = 'Casado');
```

Operador BETWEEN

Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo emplearemos el operador **Between**:

```
# SELECT * FROM Pedidos WHERE CodPostal Between 28000 And 28999;
(Devuelve los pedidos realizados en la provincia de Madrid)

# SELECT IIf(CodPostal Between 28000 And 28999, 'Provincial', 'Nacional') FROM
↳ Editores;
(Devuelve el valor 'Provincial' si el código postal se encuentra en el intervalo,
↳ 'Nacional' en caso contrario)
```

Operador LIKE

Se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis es:

```
expresión LIKE modelo
```

Operador IN

Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los indicados en una lista. Su sintaxis es:

```
expresión [Not] In(valor1, valor2, . . .)
```

```
# SELECT * FROM Pedidos WHERE Provincia In ('Madrid', 'Barcelona', 'Sevilla');
```

Clausula WHERE

La cláusula WHERE puede usarse para determinar qué registros de las tablas enumeradas en la cláusula FROM aparecerán en los resultados de la instrucción SELECT. WHERE es opcional, pero cuando aparece debe ir a continuación de FROM:

```
# SELECT Apellidos, Salario FROM Empleados
WHERE Salario > 21000;
# SELECT Id_Producto, Existencias FROM Productos
WHERE Existencias <= Nuevo_Pedido;
```

Agrupamiento de registros (Agregación)

AVG

Calcula la media aritmética de un conjunto de valores contenidos en un campo especificado de una consulta:

```
Avg(expr)
```

La función Avg no incluye ningún campo Null en el cálculo. Un ejemplo del funcionamiento de **AVG**:

```
# SELECT Avg(Gastos) AS Promedio FROM
Pedidos WHERE Gastos > 100;
```

MAX, MIN

Devuelven el mínimo o el máximo de un conjunto de valores contenidos en un campo específico de una consulta. Su sintaxis es:

```
Min(expr)
Max(expr)
```

Un ejemplo de su uso:

```
# SELECT Min(Gastos) AS ElMin FROM Pedidos
WHERE Pais = 'Costa Rica';
# SELECT Max(Gastos) AS ElMax FROM Pedidos
WHERE Pais = 'Costa Rica';
```

SUM

Devuelve la suma del conjunto de valores contenido en un campo específico de una consulta. Su sintaxis es:

```
Sum(expr)
```

Por ejemplo:

```
# SELECT Sum(PrecioUnidad * Cantidad)
AS Total FROM DetallePedido;
```

GROUP BY

Combina los registros con valores idénticos, en la lista de campos especificados, en un único registro:

```
# SELECT campos FROM tabla WHERE criterio
GROUP BY campos del grupo
```

Todos los campos de la lista de campos de SELECT deben o bien incluirse en la cláusula GROUP BY o como argumentos de una función SQL agregada:

```
# SELECT Id_Familia, Sum(Stock)
FROM Productos GROUP BY Id_Familia;
```

HAVING es similar a WHERE, determina qué registros se seleccionan. Una vez que los registros se han agrupado utilizando GROUP BY, HAVING determina cuales de ellos se van a mostrar.

```
# SELECT Id_Familia Sum(Stock) FROM Productos GROUP BY Id_Familia HAVING Sum(Stock) >
100 AND NombreProducto Like BOS*;
```

Manejo de varias tablas

Partiendo de la definición de las siguientes tablas:

1. Tabla clientes

cid	nombre	telefono
1	jose	111
2	maria	222
3	manuel	333
4	jesus	4444

2. Tabla Acciones

aid	cid	accion	cantidad
1	2	REDHAT	10
2	4	NOVELL	20
3	4	SUN	30
4	5	FORD	100

Cosultas mediante JOIN

JOIN

La sentencia SQL JOIN se utiliza para relacionar varias tablas. Nos permitirá obtener un listado de los campos que tienen coincidencias en ambas tablas:

```
# select nombre, telefono, accion, cantidad from clientes join acciones on clientes.
↪cid=acciones.cid;
```

resultando:

nombre	telefono	accion	cantidad
maria	222	REDHAT	10
jesus	4444	NOVELL	20
jesus	4444	SUN	30

LEFT JOIN

La sentencia LEFT JOIN nos dará el resultado anterior mas los campos de la tabla de la izquierda del **JOIN** que no tienen coincidencias en la tabla de la derecha:

```
# select nombre, telefono, accion, cantidad from clientes left join acciones on
↪clientes.cid=acciones.cid;
```

con resultado:

nombre	telefono	accion	cantidad
jose	111	NULL	NULL
maria	222	REDHAT	10
manuel	333	NULL	NULL
jesus	4444	NOVELL	20
jesus	4444	SUN	30

RIGHT JOIN

Identico funcionamiento que en el caso anterior pero con la tabla que se incluye en la consulta a la derecha del **JOIN**:

```
# select nombre, telefono, accion, cantidad from clientes right join acciones on
↪clientes.cid=acciones.cid;
```

cuyo resultado será:

nombre	telefono	accion	cantidad
maria	222	REDHAT	10
jesus	4444	NOVELL	20
jesus	4444	SUN	30
NULL	NULL	FORD	100

UNION y UNION ALL

Podemos combinar el resultado de varias sentencias con UNION o UNION ALL. UNION no nos muestra los resultados duplicados, pero UNION ALL si los muestra:

```
# select nombre, telefono, accion, cantidad from clientes left join acciones on
↪ clientes.cid=acciones.cid where accion is null union select nombre, telefono,
↪ accion, cantidad from clientes right join acciones on clientes.cid=acciones.cid
↪ where nombre is null;
```

que mostrará:

nombre	telefono	accion	cantidad
jose	111	NULL	NULL
manuel	333	NULL	NULL
NULL	NULL	FORD	100

Vistas

Las vistas (“views”) en SQL son un mecanismo que permite generar un resultado a partir de una consulta (query) almacenado, y ejecutar nuevas consultas sobre este resultado como si fuera una tabla normal. Las vistas tienen la misma estructura que una tabla: filas y columnas. La única diferencia es que sólo se almacena de ellas la definición, no los datos.

La cláusula CREATE VIEW permite la creación de vistas. La cláusula asigna un nombre a la vista y permite especificar la consulta que la define. Su sintaxis es:

```
# CREATE VIEW id_vista [(columna,...)]AS especificación_consulta;
```

Opcionalmente se puede asignar un nombre a cada columna de la vista. Si se especifica, la lista de nombres de las columnas debe de tener el mismo número de elementos que el número de columnas producidas por la consulta. Si se omiten, cada columna de la vista adopta el nombre de la columna correspondiente en la consulta.

Referencias

SQL en Wikipedia <http://es.wikipedia.org/wiki/SQL>

Tutorial de SQL <http://www.unalmed.edu.co/~mstabare/Sql.pdf>

SQL - JOIN Básico <http://ariel.esdebian.org/27200/sql-join-basico>

SQL Commands - <http://www.postgresql.org/docs/9.1/static/sql-commands.html>

Ejemplos

Crear tabla:

```
# CREATE TABLE Empleado
(
  id serial NOT NULL PRIMARY KEY,
  Nombre VARCHAR(50),
  Apellido VARCHAR(50),
  Direccion VARCHAR(255),
  Ciudad VARCHAR(60),
  Telefono VARCHAR(15),
  Peso VARCHAR (5),
  Edad integer,
  "Actividad" VARCHAR(100),
  idCargo integer
);
```

Añadir columna:

```
alter table empleado add column numero_sanitario varchar(9);
```

Modificar columna a no nulo:

```
alter table empleado alter column numero_sanitario set not null;
```

Eliminar columna:

```
alter table empleado drop column actividad;
```

Eliminar base de datos:

```
drop database pruebas;
```

Eliminar tabla:

```
drop table empleado;
```

Insertar empleado:

```
insert into empleado ('id', 'nombre' ) values (1, 'Pedro');

insert into empleado ("nombre" ) values ("Pedro");

insert into empleado ("nombre" ) values ('Pedro');
insert into empleado ("nombre" ) values ('Martin');
insert into empleado ("nombre" ) values ('Miguel');
insert into empleado ("nombre" ) values ('Maria');
insert into empleado ("nombre" ) values ('Luis');
```

Modificar empleado:

```
update empleado set "nombre" = 'Luis' where nombre = 'Pedro';

¿ update empleado set "nombre" = 'Luis'; ?
```

Eliminar empleado:

```
delete from empleado where "nombre" = 'Luis';

¿ delete from empleado; ?
```

Cargamos datos de ejemplo:

```
psql -U alumno -d curso < datos_ejemplo.sql
```

Mostrar todos los departamentos:

```
select * from curso.departamento;

¿ select id from curso.departamento; ?
¿ select d.id from curso.departamento; ?
```

Mostrar el departamento de los municipios:

```
select distinct departamento from municipio;

¿ select departamento from municipio; ?
```

Mostrar los municipios del departamento 1188 o 1201:

```
select * from municipio where departamento = 1188 OR departamento = 1201;
```

Municipios que no sean el 1200:

```
select * from municipio where departamento <> 1200;
```

Municipios con departamento entre 1196 y 1202:

```
select * from municipio where departamento between 1188 and 1200;

¿ select * from municipio where departamento between 1188 and 1200 order by ↵
↵departamento; ?
```

Trabajador que se llame Miguel:

```
select * from trabajador where nombre like 'Miguel'

¿ select * from trabajador where nombre like 'Miguel' and apellidos like '%Gar%' ?;
```

Media de la población de los municipios:

```
select avg(m.poblacion) from municipio as m;
```

Municipio de mayor superficie y de menor perímetro:

```
select max(m.superficie) from municipio as m;
select min(m.perimetro) from municipio as m;
```

Suma total de la población de los municipios del departamento 1200:

```
select sum(m.poblacion) from municipio as m where m.departamento = 1200;
```

Número de municipios agrupados por departamento:

```
select departamento, count(*) from municipio group by departamento;

¿ select departamento, municipio from municipio group by departamento; ?

¿ select departamento, municipio from municipio group by departamento, municipio; ?
```

Seleccionar municipios con sus departamentos:

```
select * from municipio join departamento on municipio.departamento = departamento.id;

select * from municipio as m, departamento as d where m.departamento = d.id;

¿ select count(*) from municipio as m, departamento as d where m.departamento = d.id; ↵
↵?

¿ select count(*) from municipio; ?
```

Seleccionar todos los municipios con sus departamentos:

```
select * from municipio left join departamento on municipio.departamento = ↵
↵departamento.id;

select count(*) from municipio left join departamento on municipio.departamento = ↵
↵departamento.id;
```

Seleccionar todos los municipios con departamento 1200 junto con los del departamento 1201

```
select * from municipio where departamento = 1200 union select * from municipio where departamento
= 1201
```

Crear una vista con los municipios del departamento de Choluteca

```
create view municipios_choluteca as select m.id as id_mun, d.id as id_dep, m.municipio, m.poblacion,
d.name from municipio as m, departamento as d where m.departamento = d.id and d.name = 'Choluteca';
```

Ejercicios:

- * Mostrar los barrios que no tienen zonas asignadas
- * Mostrar que equipo tiene más zonas asignadas
- * ¿Cuántos barrios no tienen equipos designados?
- * ¿Existe algún equipo sin responsable?
- * ¿Qué población atiende la zona 3?

PostGIS, características espaciales

Introducción

Para el curso que nos compete realizaremos las prácticas con la versión 2.0 de *PostGIS* por ser la que dispone del módulo de Raster y de Topología. *PostGIS* es una extensión espacial para *PostgreSQL* que permite gestionar objetos geográficos, de tal manera que añade esta capacidad al SGBD *PostgreSQL*.

Instalación y configuración de *PostGIS*

En función del sistema operativo que estemos usando, la instalación será de una forma u otra. Como ya hemos mencionado, vamos a contemplar tres sistemas operativos:

- Sistemas Windows XP/7
- Sistemas Mac OS X
- Sistemas basados en Debian

Windows

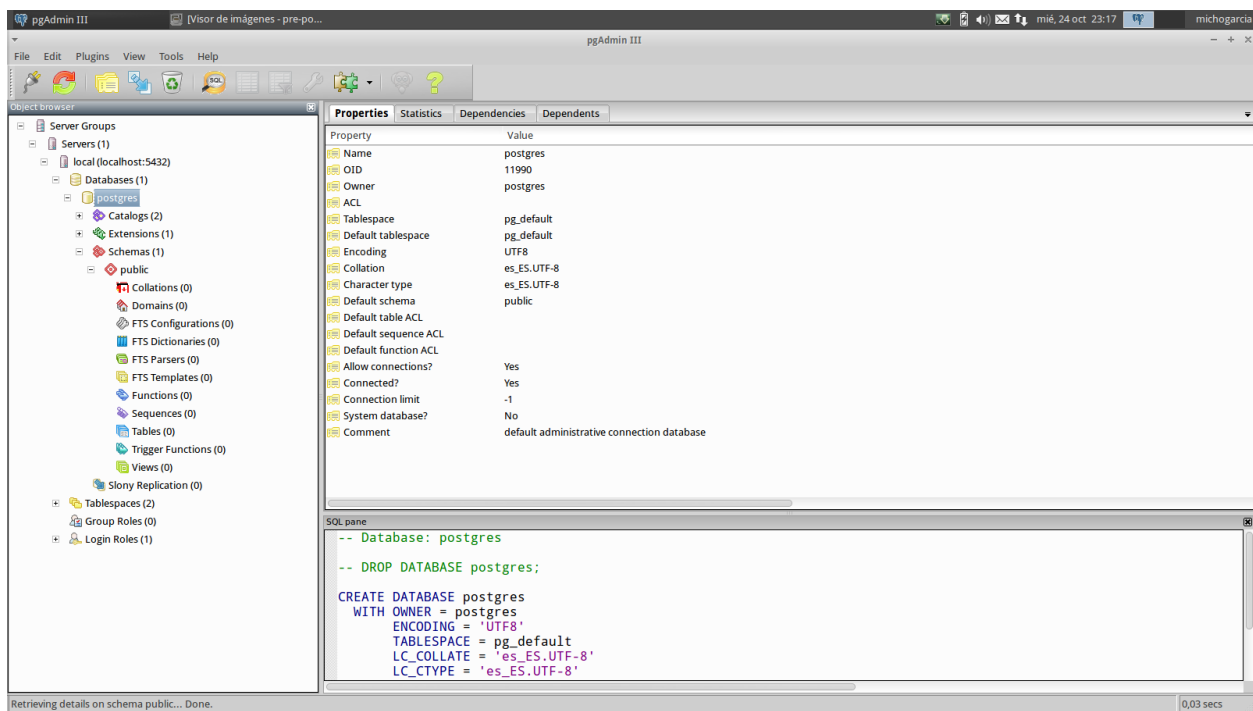
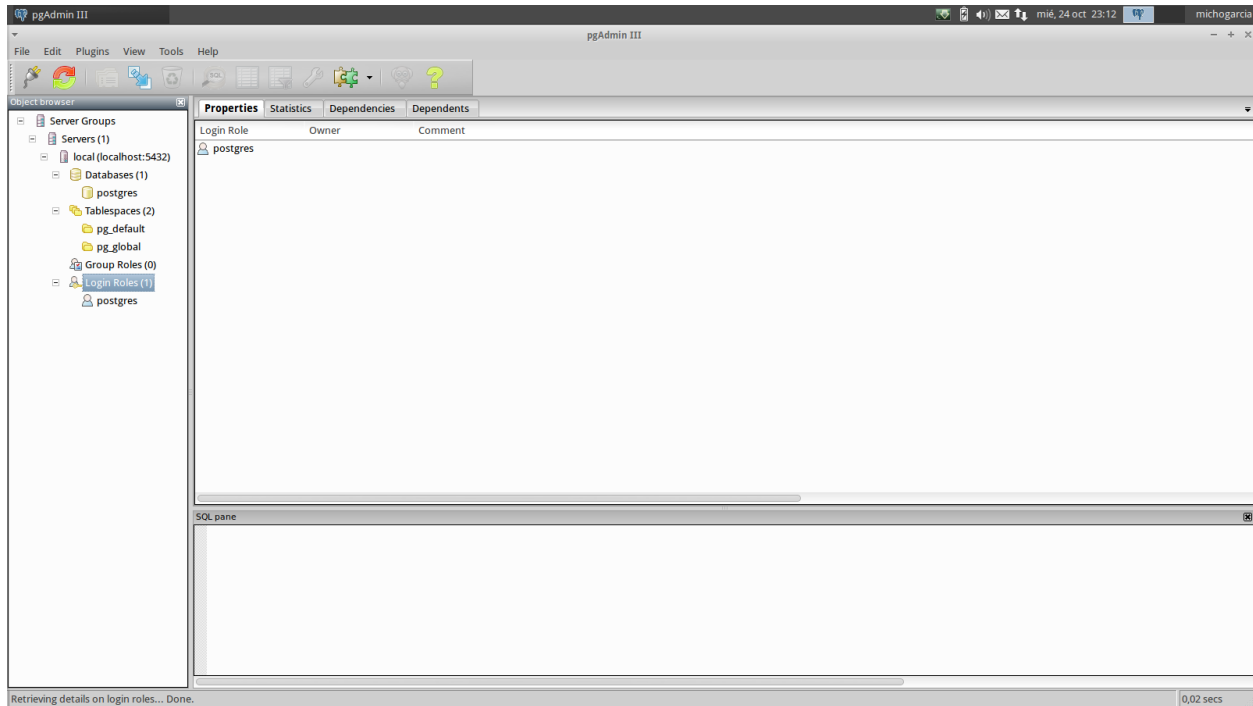
Mac OS X

Debian/Ubuntu/Derivados

Espacialización de una base de datos

La integración de *PostGIS* con *PostgreSQL* está hecha en el lenguaje PL/PGSQL, por lo que para dotar de capacidades espaciales una base de datos existente es necesario primero añadir soporte para este lenguaje. Esto es necesario para versiones de *PostgreSQL* anteriores a la 8.4:

```
$ createlang plpgsql curso
```



Hecho esto, la instalación de *PostGIS* se hará de una forma u otra, en función de si estamos usando una versión de *PostgreSQL* anterior a la 9.1 o no.

Instalación de *PostGIS* en versión de *PostgreSQL* inferior a 9.1

A continuación hay que localizar dos ficheros SQL de PostGIS que al ejecutarse añadan las estructuras necesarias a nuestra base de datos. Estos ficheros se llaman *lwpostgis.sql* (o simplemente *postgis.sql*) y *spatial_ref_sys.sql*.

Para localizarlos podemos utilizar el comando *locate*:

```
$ locate postgis.sql
/usr/share/postgresql/8.4/contrib/postgis-|PG_VERSION|/postgis.sql
/usr/share/postgresql/8.4/contrib/postgis-|PG_VERSION|/uninstall_postgis.sql
/usr/share/postgresql/9.1/contrib/postgis-|PG_VERSION|/postgis.sql
/usr/share/postgresql/9.1/contrib/postgis-|PG_VERSION|/uninstall_postgis.sql

$ locate spatial_ref_sys.sql
/usr/share/postgresql/8.4/contrib/postgis-|PG_VERSION|/spatial_ref_sys.sql
/usr/share/postgresql/9.1/contrib/postgis-|PG_VERSION|/spatial_ref_sys.sql
```

Una vez localizados dos ficheros de la misma versión, es necesario ejecutarlos en el servidor. Existen dos formas de hacerlo con *psql*: el parámetro *-f* y el comando *\i*. Con el parámetro *-f* llamaríamos a *psql* desde la línea de comandos del sistema y especificaríamos el fichero *.sql* que queremos ejecutar con dicho parámetro. Para que el fichero se ejecute en la base de datos que nos interesa hay que especificar también el parámetro *-d* visto anteriormente:

```
$ psql -U postgres -d template_postgis -f /usr/share/postgresql/9.1/contrib/postgis-
→|PG_VERSION|/postgis.sql
$ psql -U postgres -d template_postgis -f /usr/share/postgresql/9.1/contrib/postgis-
→|PG_VERSION|/spatial_ref_sys.sql
```

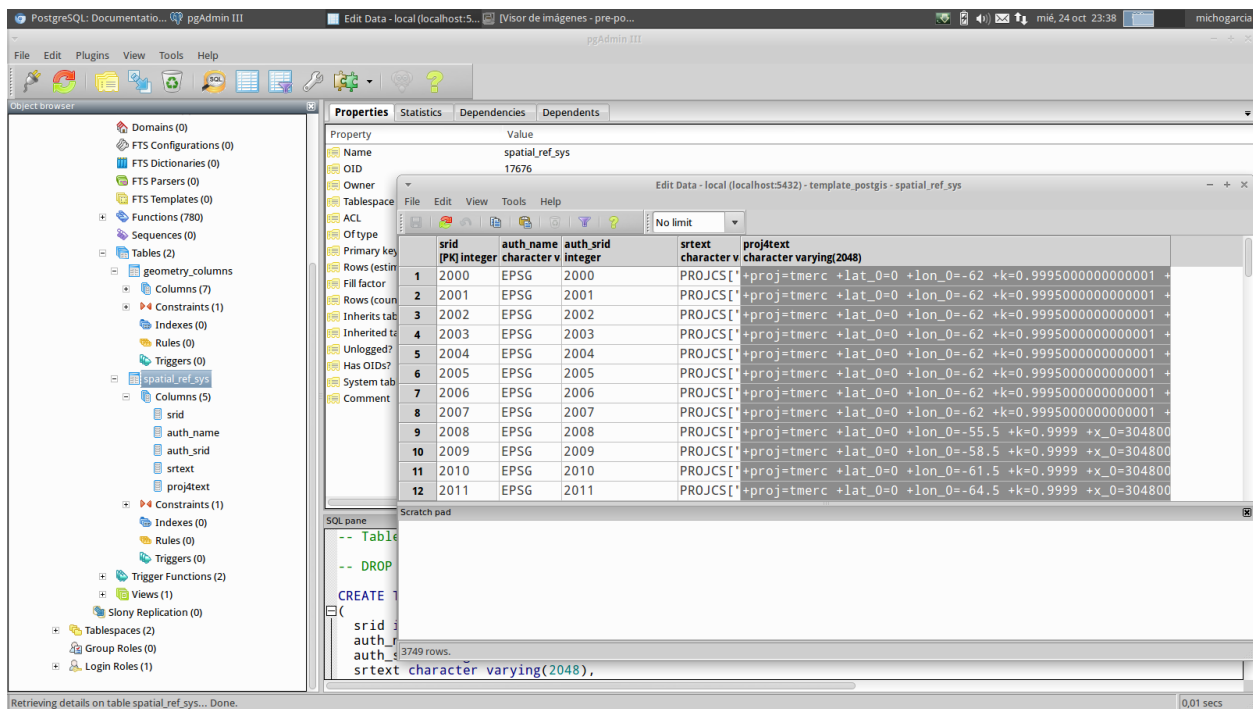
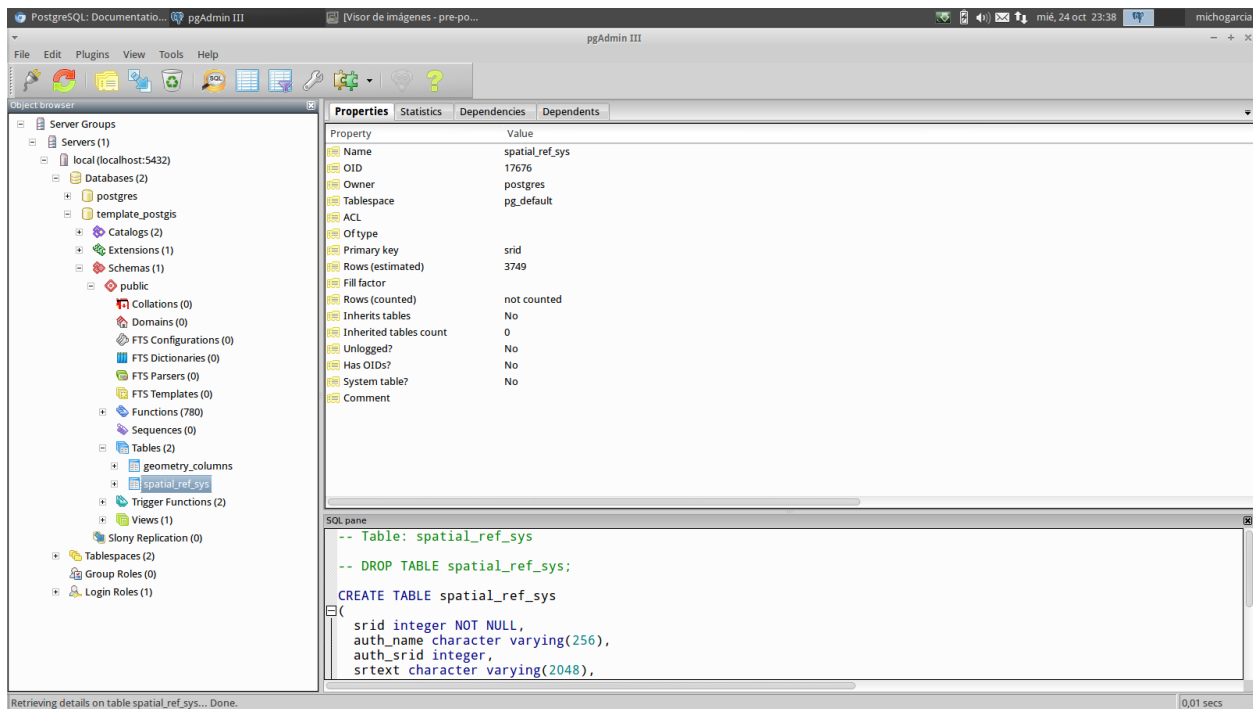
La opción de usar el comando *\i* consiste en entrar al modo interactivo de *psql* conectando a la base de datos de interés y ejecutando el fichero con *\i*:

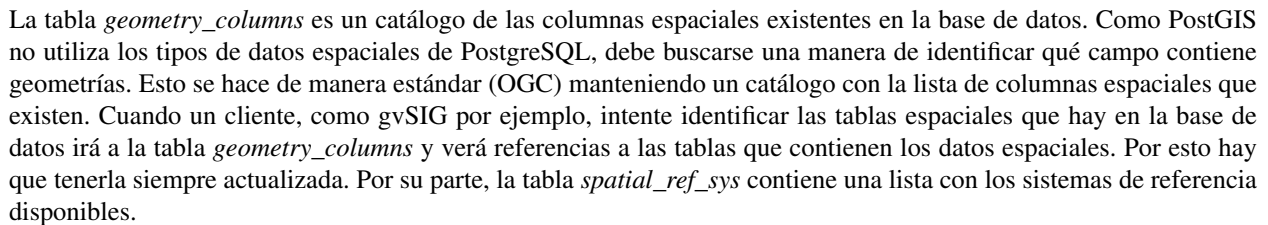
```
$ psql -U postgres -d template_postgis
=# \i /usr/share/postgresql/9.1/contrib/postgis-|PG_VERSION|/postgis.sql
=# \i /usr/share/postgresql/9.1/contrib/postgis-|PG_VERSION|/spatial_ref_sys.sql
```

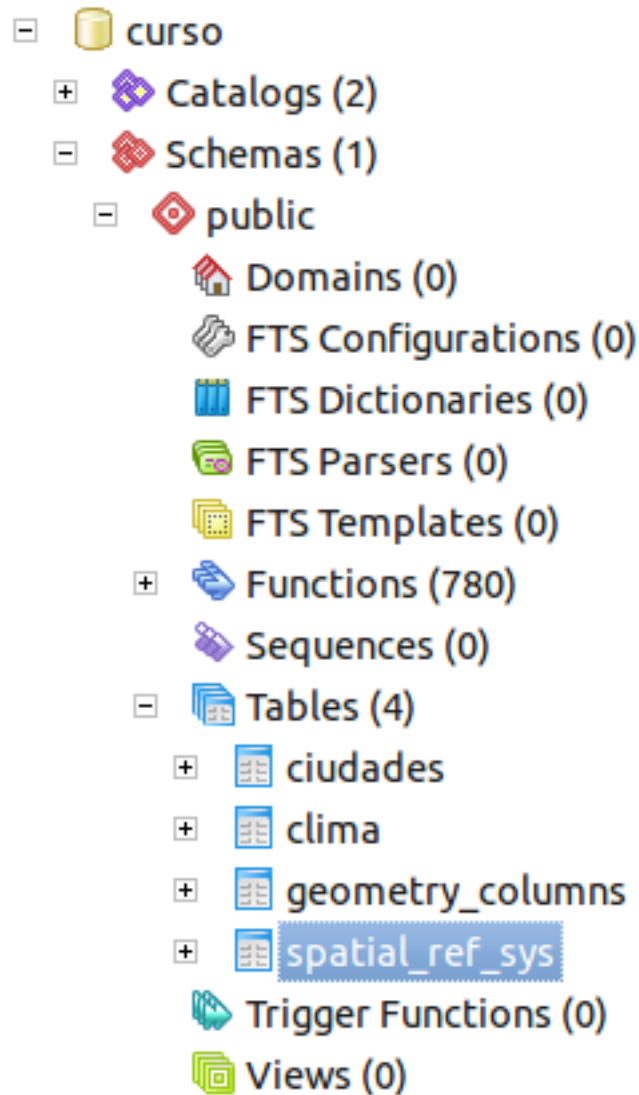
o también se puede entrar a la base de datos por defecto (*postgres*) y conectar interactivamente a nuestra base de datos luego con *\c*:

```
$ psql -U postgres
=# \c template_postgis
=# \i /usr/share/postgresql/9.1/contrib/postgis-|PG_VERSION|/postgis.sql
=# \i /usr/share/postgresql/9.1/contrib/postgis-|PG_VERSION|/spatial_ref_sys.sql
```

Tras esta operación se puede observar que han aparecido dos nuevas tablas: *geometry_columns* y *spatial_ref_sys*, además de numerosas funciones en el esquema *public*.







Podremos comprobar la versión que tenemos instalada de *PostGIS* mediante:

```
# SELECT postgis_full_version();
```

Instalación de *PostGIS* en versión de *PostgreSQL* 9.1 o superior

Si se cuenta con *PostgreSQL* 9.1 o superior, podremos utilizar la expresión **CREATE EXTENSION**.

De manera que instalar *PostGIS* será tan sencillo como:

```
# CREATE EXTENSION postgis;
```

Creación de una plantilla `template_postgis`

Podremos utilizar la base de datos creada inicialmente como plantilla para la posterior creación de bases de datos espaciales evitando tener que repetir el proceso. Para ello simplemente:

```
$ createdb -U postgres -T template_postgis [nueva_base_datos]
```

En caso de querer crear la base de datos con un usuario diferente al utilizado para la creación de la plantilla debemos indicárselo al sistema:

```
# UPDATE pg_database SET datistemplate = TRUE WHERE datname = 'template_postgis';
```

Y seguidamente debemos asignarle permisos al esquema PUBLIC en las tablas de metadatos:

```
# GRANT ALL ON geometry_columns TO PUBLIC;
# GRANT ALL ON geography_columns TO PUBLIC;
# GRANT ALL ON spatial_ref_sys TO PUBLIC;
```

Indices espaciales

Una base de datos ordinaria pone a disposición del usuario una estructura de datos que sirve para agilizar el acceso a determinados registros en función del valor que tienen en un campo. La indexación para tipos de datos estándar que pueden ser ordenados (alfabéticamente o numéricamente) consiste en esencia en ordenar estos registros de manera que sea fácil localizarlos.

Pero en el caso de la información espacial no existe un orden total ya que un polígono puede contener a un punto, cruzarse con una línea, etc. En cambio, se ponen en marcha ciertas estrategias para asociar los registros con determinadas partes del territorio que cubren y así poder obtener los registros que se encuentran cerca de una posición dada.

PostgreSQL implementa un algoritmo de indexación espacial denominado GiST (Generalized Search Tree). *PostGIS* extiende los índices GiST para que funcionen adecuadamente con los tipos `geometry`.

Se recomienda el uso de estos índices cuando el número de registros excede de algunos miles. De esta manera se incrementará la velocidad de la búsqueda espacial y su visualización en SIG de escritorio.

Funciones espaciales

Una base de datos ordinaria proporciona funciones para manipular los datos en una consulta. Estas funciones incluyen la concatenación de cadenas, operaciones matemáticas o la extracción de información de las fechas. Una base de datos espacial debe proporcionar un completo juego de funciones para poder realizar análisis con los objetos espaciales: analizar la composición del objeto, determinar su relación espacial con otros objetos, transformarlo, etc.

La mayor parte de las funciones espaciales pueden ser agrupadas en una de las siguientes cinco categorías:

- Conversión: Funciones que convierten las geometrías a otros formatos externos
- Gestión: Tareas administrativas de PostGIS
- Recuperación: Obtienen propiedades y medidas de las geometrías.
- Comparación: Comparan dos geometrías y obtienen información sobre su relación espacial.
- Generación: Generan geometrías a partir de otros tipos de datos.

La lista de funciones es muy larga. Para obtener una lista comúnmente presente en las bases de datos espaciales se puede consultar el estándar [OGC SFSQL](#), que es implementado por PostGIS.

Otros módulos

En la versión 2.0 de *PostGIS* se incorporan dos módulos nuevos dentro del núcleo del producto, el módulo *Raster* y el módulo de *Topología persistente*. En función de si estamos usando una versión de PostgreSQL inferior a la 9.1 o no, instalaremos ambos módulos de una forma u otra.

Instalación de módulos en PostgreSQL inferior a versión 9.1

Deberemos instalar cada módulo cargando ficheros PL/pgSQL. Lo haremos mediante la herramienta de línea de comandos *psql*

Raster

Este módulo se encarga de gestionar la información raster siguiendo la misma filosofía que el tipo geometry y permitiendo análisis raster y mezclar información raster y vectorial en el análisis.

La instalación de este módulo es similar a la instalación de *PostGIS* realizándose mediante la ejecución de scripts que crean la funcionalidad necesaria para el manejo raster en la base de datos.:

```
$ psql -U postgres -f path_rtpostgis.sql -d [nombre_base_datos]
$ psql -U postgres -f path_raster_comments.sql -d [nombre_base_datos]
```

Topología persistente

Este es una forma de estructurar la información geográfica de manera diferente al modelo *simple features*. Se instala de manera opcional y no se tratará en este curso por exceder los objetivos del mismo.

Instalación de módulos en PostgreSQL inferior a versión 9.1

Como sucede al instalar la extensión *PostGIS*, si contamos con *PostgreSQL* 9.1 o superior, basta con que instalemos los siguientes comandos:

```
# CREATE EXTENSION postgis_raster;
# CREATE EXTENSION postgis_topology;
```

Prácticas

Creé una base de datos espacial que se llame *curso* a partir de la plantilla *template_postgis*.

Cree un esquema *gis* en la base de datos *curso*.

Simple Feature Model

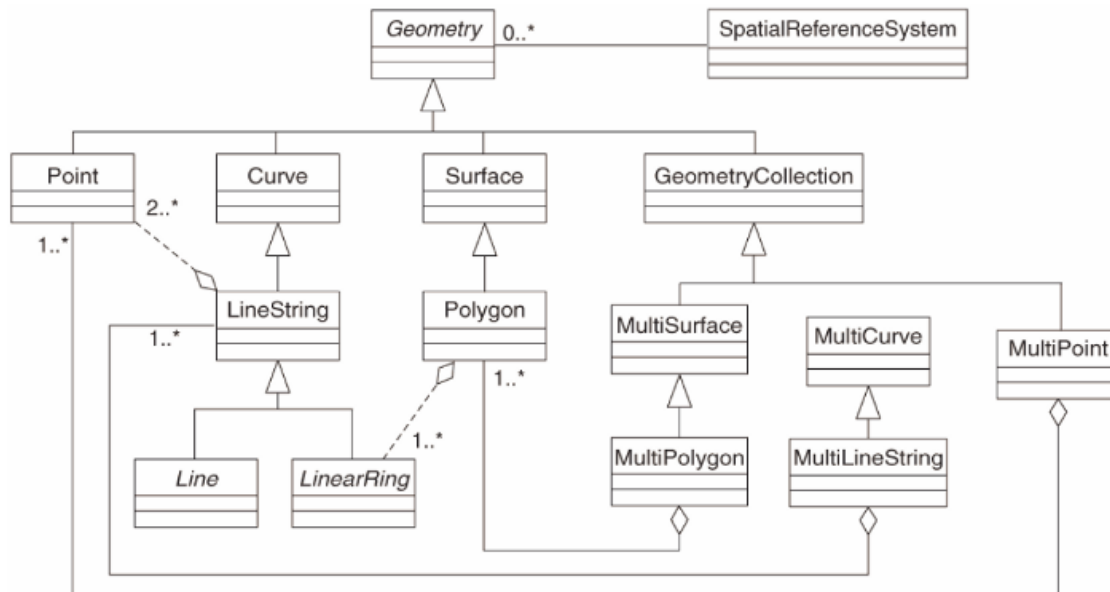
Nota:	Fecha	Autores
	1 Noviembre 2012	■ Micho García (micho.garcia@geomati.co)
	15 Octubre 2013	■ Jorge Arévalo(jorge.arevalo@geomati.co)

©2012 Micho García

Excepto donde quede reflejado de otra manera, la presente documentación se halla bajo licencia : Creative Commons (Creative Commons - Attribution - Share Alike: <http://creativecommons.org/licenses/by-sa/3.0/deed.es>)

OGC y el Simple Feature Model

La OGC o Open Geospatial Consortium y la International Organization for Standardization o ISO, define unas normas que serán utilizadas para la definición posterior de las geometrías como son la SFA y la SQL/MM. Según estas, siendo la segunda una extensión de la primera, las geometrías se definirán en función del siguiente esquema de herencia:



Aquí podremos comprobar que productos tienen implementadas la diferentes versiones del estandar [Products by Spec](#). Dentro de este esquema se definen tres tipos diferentes de geometrías:

- **Geometrías abstractas**, que sirven para modelar el comportamiento de las geometrías que heredan de ellas.
- **Geometrías básicas**, son las principales de PostGIS, soportadas desde los inicios de este y que soportan un análisis espacial completo.
- **Geometrías circulares**, geometrías que soportan tramos circulares

Para poder profundizar en el estándar deberemos conocer antes algunas definiciones como:

Dimensión de una geometría

El concepto de dimensión se explica de una manera sencilla mediante el uso de algunos ejemplos. Hace referencia a las posibles direcciones en el plano en las que se puede uno desplazar dentro de la geometría:

- una entidad de tipo punto, tendrá dimensión 0
- una de tipo línea, tendrá dimensión 1
- una de tipo superficie, tendrá una dimensión igual a 2.

En PostGIS utilizando una función especial, podremos obtener el valor de esta dimensión. Si se trata de una colección de geometrías, el valor que se obtendrá será el de la dimensión de mayor valor de la colección:

```
ST_Dimension(geom)
```

Nos devolverá la dimensión de una geometría, que no hay que confundir con lo devuelto por:

```
ST_NDims(geom)
```

que nos devolverá la dimensión de las coordenadas de la geometría.

Interior, contorno y exterior de las geometrías

Las definiciones las encontraremos en la norma. A continuación se indican los valores para las geometrías básicas.

Tipos de geometría	Interior	Contorno
ST_Point	El propio punto o puntos	Vacio
ST_Linestring	Puntos que permanecen cuando contorno se elimina	Dos puntos finales
ST_MultiLinestring	Idem	Puntos de contorno de un n° impar de elementos
ST_Polygon	Puntos del interior de los anillos	Conjunto de anillos exterior e interior (Rings)
ST_Multipolygon	Idem	Conjunto de anillos exterior e interior (Rings)

WKT y WKB

WKT es el acrónimo en inglés de *Well Known Text*, que se puede definir como una codificación o sintaxis diseñada específicamente para describir objetos espaciales expresados de forma vectorial. Los objetos que es capaz de describir son: puntos, multipuntos, líneas, multilíneas, polígonos, multipolígonos, colecciones de geometría y puntos en 3 y 4 dimensiones. Su especificación ha sido promovida por un organismo internacional, el Open Geospatial Consortium, siendo su sintaxis muy fácil de utilizar, de forma que es muy generalizado su uso en la industria geoinformática. De hecho, WKT es la base de otros formatos más conocidos como el KML utilizado en Google Maps y Google Earth.

Muchas de las bases de datos espaciales, y en especial Postgresql, utiliza esta codificación cuando se carga la extensión PostGIS. Existe una variante de este lenguaje, pero expresada de forma binaria, denominada WKB (*Well Known Binary*), también utilizada por estos gestores espaciales, pero con la ventaja de que al ser compilada en forma binaria la velocidad de proceso es muy elevada.

A efectos prácticos la sintaxis WKT consta de una descripción de los vértices que componen la geometría. Para que esta forma de especificar las geometrías tengan sentido deben de acompañarse de una indicación de la referencia espacial o proyección cartográfica utilizada en dicho vector.

Ejemplos de sintaxis:

```
Punto: POINT(30 50)
Línea: LINESTRING(1 1, 5 5, 10 10, 20 20)
Multilínea: LINESTRING( (1 1, 5 5, 10 10, 20 20), (20 30, 10 15, 40 5) )
Polígono simple: POLYGON ((0 0, 10 0, 10 10, 0 0))
Varios polígono en una sola geometría (multipolígono): POLYGON ( (0 0, 10 0, 10 10, 0
↪10, 0 0), ( 20 20, 20 40, 40 40, 40 20, 20 20) )
Geometrías de distinto tipo en un sólo elemento: GEOMETRYCOLLECTION(POINT(4 6),
↪LINESTRING(4 6,7 10))
Punto vacío: POINT EMPTY
Multipolígono vacío: MULTIPOLYGON EMPTY
```

WKB acrónimo de *Well Known Binary* es la variante de este lenguaje, pero expresada de forma binaria, también utilizada por los gestores espaciales, pero con la ventaja de que al ser compilada en forma binaria la velocidad de proceso es muy elevada.

Tipos de datos espaciales

Una base de datos ordinaria tiene cadenas, fechas y números. Una base de datos añade tipos adicionales para georeferenciar los objetos almacenados. Estos tipos espaciales abstraen y encapsulan estructuras tales como el contorno y la dimensión.

De forma simplificada, tenemos los siguientes tipos de datos espaciales:

Tipo de geometria	WKT
POINT	"POINT(0 0)"
LINESTRING	"LINESTRING(0 0, 1 1, 2 2, 3 4)"
POLYGON	"POLYGON(0 0, 0 1, 1 1, 0 0)"
MULTIPOINT	"MULTIPOINT(0 0, 1 1, 2 2)"
MULTILINESTRING	"MULTILINESTRING ((10 10, 2 2, 10 40), (40 40, 30 30, 40 20, 30 10))"
MULTIPOLYGON	"MULTIPOLYGON (((3 2, 0 0, 5 4, 3 2)))"
GEOMETRY COLLECTION	"GEOMETRYCOLLECTION(POINT(4 6),LINESTRING(4 6,7 10))"

Definición de geometrías básicas

Point y Multipoint

- Geometrias con 0 dimensiones
- El contorno es el conjunto vacio
- Una geometría Multipoint es simple si no tiene ningún punto repetido

Linestring

- Geometrias de 1 dimensión
- Simple si no pasa por el mismo punto dos veces
- Cerrada si su punto inicial y final es el mismo
- El contorno si es cerrada es el conjunto vacio
- El contorno si no es cerrada son su punto final e inicial
- Si es simple y cerrada es un anillo (Ring)

Multilinestring

- Geometrías de 1 dimensión
- Cerrada si todos sus elementos son cerrados
- Si es cerrada su contorno es el conjunto vacio

Polygon

- Geometrías de 2 dimensiones
- Contiene un único interior conectado
- Tiene un anillo exterior y 0 o más anillos interiores
- El contorno es un conjunto de lineas cerradas que se corresponden con sus contornos exterior e interior

Multipolygon

- El interior de cualquiera de las superficies que contiene no puede intersectar
- El contorno de cualquiera de las superficies que contiene puede intersectar pero solo en un número finito de puntos
- Son simples

Práctica

Haciendo uso de las operaciones espaciales mediante PostGIS o a través del software JTS Builder representar:

- Un polígono que no sea simple.
- Dibujar un polígono y un multipolígono que no sean válidos.
- Una multilinestring con tres puntos de contorno.
- Una linestring abierta y que no sea simple.
- Una multilinestring cerrada y simple.

Referencias

Well Known Text en Wikipedia http://en.wikipedia.org/wiki/Well-known_text

Lesson 2. Simple Feature Model [EN] http://manual.linfiniti.com/en/postgis/simple_feature_model.html

Simple Feature Acces in Wikipedia [EN] http://en.wikipedia.org/wiki/Simple_Feature_Access

Crear tabla de ejemplos:

```
CREATE TABLE geometries (name varchar, geom geometry);

INSERT INTO geometries VALUES
  ('Point', 'POINT(0 0)'),
  ('Linestring', 'LINESTRING(0 0, 1 1, 2 1, 2 2)'),
  ('Polygon', 'POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'),
  ('PolygonWithHole', 'POLYGON((0 0, 10 0, 10 10, 0 10, 0 0),(1 1, 1 2, 2 2, 2 1, 1
↪1))'),
  ('Collection', 'GEOMETRYCOLLECTION(POINT(2 0),POLYGON((0 0, 1 0, 1 1, 0 1, 0 0)))');

SELECT Populate_Geometry_Columns();

SELECT name, ST_AsText(geom) FROM geometries;
```

Para la creación de esta tabla se ha realizado mediante CAST, ya que no se indica la operación:

```
INSERT INTO geometries VALUES
  ('Point', ST_GeomFromText('POINT(0 0)'),),
  ('Linestring', ST_GeomFromText('LINESTRING(0 0, 1 1, 2 1, 2 2)'),),
  ('Polygon', ST_GeomFromText('POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'),),
  ('PolygonWithHole', ST_GeomFromText('POLYGON((0 0, 10 0, 10 10, 0 10, 0 0),(1 1, 1
↪2, 2 2, 2 1, 1 1))'),),
  ('Collection', ST_GeomFromText('GEOMETRYCOLLECTION(POINT(2 0),POLYGON((0 0, 1 0, 1
↪1, 0 1, 0 0))'),),
  ('MultyLinestring', ST_GeomFromText('MULTILINESTRING ((1 1, 5 5, 10 10, 20 20),(20
↪30, 10 15, 40 5))'));)
```

Para comprobar la dimensión:

```
select ST_Dimension(geom) from (select * from geometries where name = 'Point') as
↪geom;

select ST_NDims(geom) from (select * from geometries where name = 'Point') as geom;
```

```
select ST_CoordDim(geom) from (select * from geometries where name = 'Point') as geom;
```

Podemos calcular el interior de una geometría:

```
select st_astext(ST_Boundary(geom)) from (select * from geometries where name =  
↪ 'PolygonWithHole') as geom;  
  
select st_astext(ST_Boundary(geom)) from (select * from geometries where name =  
↪ 'Linestring') as geom;
```

Comprobamos la existencia de los métodos básicos de las geometrías:

```
select st_astext(ST_Boundary(geom)) from (select * from geometries where name =  
↪ 'PolygonWithHole') as geom;  
  
select st_srid(geom) from (select * from geometries where name = 'PolygonWithHole')  
↪ as geom;
```

Definición de geometrías:

Point:

```
select ST_NDims(geom) from (select * from geometries where name = 'Point') as geom;  
  
select st_astext(ST_Boundary(ST_GeomFromText('MULTIPOINT ((3 6), (4 6), (3 6))')))  
  
select st_astext(ST_Boundary(geom)) from (select * from geometries where name = 'Point'  
↪ ') as geom;  
  
select st_issimple(ST_GeomFromText('MULTIPOINT ((3 6), (4 6), (3 6))'));
```

Linestring:

```
select ST_Dimension(geom) from (select * from geometries where name = 'Linestring')  
↪ as geom;  
  
select st_issimple(ST_GeomFromText('LINESTRING (3 8, 5 8, 5 9, 3 8, 2 7)'));  
  
select stisclosed(ST_GeomFromText('LINESTRING (3 9, 6 9, 6 7, 3 9)'));  
  
select st_astext(ST_Boundary(ST_GeomFromText('LINESTRING (4 9, 4 6, 6 6)')))  
  
select st_isring(st_geomfromtext('LINESTRING (3 7, 2 4, 6 3, 6 7, 3 7)'));
```

MultiLinestring:

```
select st_Dimension(ST_GeomFromText('MULTILINESTRING ((2 8, 2 8, 0.9 5, 3 3, 6 6.6),  
↪ (5.5 8.6, 9.2 8.8, 8.6 4.7, 5 3), (1.3 1.9, 3.1 0.9, 2.8 3, 6.8 0.1))'));  
  
select stisclosed(st_geomfromtext('MULTILINESTRING ((2 8, 2 6, 4 6, 2 8), (5 5.1, 6  
↪ 7, 7 6))'));  
  
select st_astext(ST_Boundary(st_geomfromtext('MULTILINESTRING ((2 8, 2 6, 4 6, 2 8),  
↪ (5 5.1, 6 7, 7 6))')));
```

MultiPolygon:

```
select st_isvalid(ST_GeomFromText('MULTIPOLYGON (((2.1 9, 6 9, 6 5, 2.1 5, 2.1 9)),
↪((7.1 6, 7.1 3, 4 3, 4 6, 7.1 6)))'));

select st_isvalidreason(ST_GeomFromText('MULTIPOLYGON (((2.1 9, 6 9, 6 5, 2.1 5, 2.1
↪9)), ((7.1 6, 7.1 3, 4 3, 4 6, 7.1 6)))'));

select st_issimple(ST_GeomFromText('MULTIPOLYGON (((2.1 9, 6 9, 6 5, 2.1 5, 2.1 9)),
↪((7.1 6, 7.1 3, 4 3, 4 6, 7.1 6)))'));
```

Importación y exportación de datos

Nota:	Fecha	Autores
	1 Noviembre 2012	■ Micho García (micho.garcia@geomati.co)
	15 Octubre 2013	■ Jorge Arévalo(jorge.arevalo@geomati.co)

©2012 Micho García

Excepto donde quede reflejado de otra manera, la presente documentación se halla bajo licencia : Creative Commons (Creative Commons - Attribution - Share Alike: <http://creativecommons.org/licenses/by-sa/3.0/deed.es>)

En este tema nos introduciremos en el uso de herramientas de importación/exportación de datos hasta/desde PostGIS. Se realizará la importación con archivos de tipo ESRI ShapeFile y con datos descargados de OpenStreetMap. Para realizar estos procesos, se dispondrá de herramientas como `shp2pgsql` que vienen incluidas en PostGIS o se utilizarán otras como `osmosis` u `osm2pgsql` descargadas desde los repositorios.

Importación ESRI shapes mediante `shp2pgsql`

El cargador `shp2pgsql` convierte archivos ESRI Shape en SQL preparado para la inserción en la base de datos. Se utiliza desde la línea de comandos, aunque existe una versión con interfaz gráfica para el sistema operativo Windows. Se puede acceder a la ayuda de la herramienta mediante:

```
$ shp2pgsql -?
```

Para el uso de la herramienta:

```
$ shp2pgsql [<opciones>] <ruta_shapefile> [<esquema>.]<tabla>
```

entre las opciones encontraremos:

- **-s <srid>** Asigna el sistema de coordenadas. Por defecto será -1
- **(-dlalclp)**
 - **-d** Elimina la tabla, la recrea y la llena con los datos del shape
 - **-a** Llena la tabla con los datos del shape. Debe tener el mismo esquema exactamente
 - **-c** Crea una nueva tabla y la llena con los datos. opción por defecto.
 - **-p** Modo preparar, solo crea la tabla
- **-g <geocolumn>** Especifica el nombre de la columna geometría (usada habitualmente en modo **-a**)
- **-D** Usa el formato Dump de postgresql
- **-G** Usa tipo geografía, requiere datos de longitud y latitud
- **-k** Guarda los identificadores en postgresql
- **-i** Usa int4 para todos los campos integer del dbf
- **-I** Crea un índice espacial en la columna de la geometría
- **-S** Genera geometrías simples en vez de geometrías MULTI
- **-w** Salida en WKT
- **-W <encoding>** Especifica la codificación de los caracteres. (por defecto : “WINDOWS-1252”).
- **-N <policy>** estrategia de manejo de geometrías NULL (insert*,skip,abort).
- **-n** Solo importa el archivo DBF
- **-?** Muestra la ayuda

Práctica

Realizar la carga de estos conjuntos de datos en Shape en la misma tabla que llamaremos barrios. Descargar datos

Realice la importación de los datos proporcionados para el taller. Se le proporcionará asistencia con los parámetros a usar.

Es conveniente definir el encoding de la base de datos como LATIN1. Se puede hacer con una sentencia update:

Nota: # update pg_database set encoding=8 where datname='base_de_datos'

Nº de encoding	Name	Descripción
0	SQL_ASCII	ASCII
1	EUC_JP	Japanese EUC
2	EUC_CN	Chinese EUC
3	EUC_KR	Korean EUC
4	JOHAB	Korean EUC (Hangle base)
5	EUC_TW	Taiwan EUC
6	UNICODE	Unicode (UTF-8)
7	MULE_INTERNAL	Mule internal code
8	LATIN1	ISO 8859-1/ECMA 94 (Latin alphabet no.1)
9	LATIN2	ISO 8859-2/ECMA 94 (Latin alphabet no.2)
10	LATIN3	ISO 8859-3/ECMA 94 (Latin alphabet no.3)
11	LATIN4	ISO 8859-4/ECMA 94 (Latin alphabet no.4)
12	LATIN5	ISO 8859-9/ECMA 128 (Latin alphabet no.5)
13	LATIN6	ISO 8859-10/ECMA 144 (Latin alphabet no.6)
14	LATIN7	ISO 8859-13 (Latin alphabet no.7)
15	LATIN8	ISO 8859-14 (Latin alphabet no.8)
16	LATIN9	ISO 8859-15 (Latin alphabet no.9)
17	LATIN10	ISO 8859-16/ASRO SR 14111 (Latin alphabet no.10)
18	ISO_8859_5	ISO 8859-5/ECMA 113 (Latin/Cyrillic)
19	ISO_8859_6	ISO 8859-6/ECMA 114 (Latin/Arabic)
20	ISO_8859_7	ISO 8859-7/ECMA 118 (Latin/Greek)
21	ISO_8859_8	ISO 8859-8/ECMA 121 (Latin/Hebrew)
22	KOI8	KOI8-R(U)
23	ALT	Windows CP866
24	WIN874	Windows CP874 (Thai)
25	WIN1250	Windows CP1250
26	WIN	Windows CP1251
27	WIN1256	Windows CP1256 (Arabic)
28	TCVN	TCVN-5712/Windows CP1258 (Vietnamese)

Comprobar que se ha actualizado correctamente la tabla `geometry_columns`.

Cargar alguno de los ficheros con la GUI de pgAdmin III.

Exportación desde PostGIS a archivos de tipo ESRI Shapefile

Para este proceso utilizaremos la herramienta `pgsql2shp`. Con ella podremos convertir los datos de nuestra base de datos en archivos ESRI Shape. Igual que para el caso anterior, la herramienta se utilizará desde la línea de comandos:

```
$ pgsql2shp [<opciones>] <basedatos> [<esquema>.<tabla>]
$ pgsql2shp [<opciones>] <basedatos> <consulta>
```

las opciones serán:

```
* *-f <nombrearchivo>** Especifica el nombre del archivo a crear
* *-h <host>** Indica el servidor donde realizará la conexión
* *-p <puerto>** Permite indicar el puerto de la base de datos
* *-P <password>** Contraseña
* *-u <user>** Usuario
* *-g <geometry_column>** Columna de geometría que será exportada
```

Práctica

Exportar los barrios de la base de datos a Shapefile que tengan en el nombre la cadena 'Residencial'.

GDAL/OGR

GDAL/OGR es una librería de lectura y escritura de formatos geoespaciales, tanto *Raster* con GDAL como *Vectorial* con OGR. Se trata de una librería de software libre ampliamente utilizada.

ogrinfo

`ogrinfo` obtiene información de los datos vectoriales. Podremos utilizar esta herramienta para la obtención de esta información de las tablas que tenemos almacenadas en la base de datos. El uso se realiza a través de la consola:

```
$ ogrinfo [<opciones>] <ruta fuente datos>
```

Entre las opciones destacaremos:

```
* **-where** muestra los datos de las filas que cumplan la clausula
* **-sql** filtra la información mediante consultas SQL
* **-geom={YES/NO/SUMMARY}** modifica la visualización de la información de la_
↳ columna geométrica
```

Para utilizar `ogrinfo` contra nuestra base de datos, debemos utilizar la opción **PG**: indicándole la cadena de conexión:

```
$ ogrinfo PG:"host=localhost user=usuario dbname=basedatos password=contraseña"
```

seguidamente incluiremos cualquiera de las opciones anteriores. De esta manera por ejemplo podremos indicar:

```
$ ogrinfo PG:"host=localhost user=usuario dbname=basedatos password=contraseña" -sql "
↳ <una consulta>" <fuente de datos>
```

ogr2ogr

OGR es capaz de convertir a PostGIS todos los formatos que maneja, y será capaz de exportar desde PostGIS todos aquellos en los que tiene permitida la escritura. Ejecutando:

```
$ ogr2ogr --formats
```

podremos comprobar los formatos que maneja la herramienta. La etiqueta `write` nos indica si podemos crear este tipo de formatos. Hemos de tener en cuenta el formato de salida para poder manejar los parámetros especiales de cada formato.

En la página principal de GDAL podremos encontrar un listado de todas las opciones que nos permite manejar el comando. Detallamos a continuación algunas de las principales:

- **-select <lista de campos>** lista separada por comas que indica la lista de campos de la capa de origen que se quiere exportar
- **-where <condición>** consulta a los datos de origen
- **-sql** posibilidad de insertar una consulta más compleja

Otras opciones en referencia al formato de destino (las anteriores hacían referencia al de origen):

- **-f <driver ogr>** formato del fichero de salida
- **-lco VARIABLE=VALOR** Variables propias del driver de salida
- **-a_srs <srid>** asigna el SRID especificado a la capa de salida

- **-t_srs <srid>** Reproyecta la capa de salida según el SRID especificado

Práctica

Vamos a cargar en PostGIS directamente un fichero KML y un fichero CSV.

Cargar fichero KML

Descargar de http://forest.jrc.ec.europa.eu/effis/applications/firenews/kml/?&from_date=08/09/2013&to_date=15/09/2013 el fichero firenews.kml

A continuación, cargarlo en PostGIS con esta instrucción:

```
# ogr2ogr -a_srs epsg:4326 -f "PostgreSQL" PG:"dbname=gis host=localhost user=alumno_
↳password=alumn0 port=5432" firenews.kml
```

Ya tendríamos el fichero cargado.

Exportar a KML

Exportar la tabla de barrios a KML utilizando ogr2ogr para poder publicarla en Google Maps

```
# ogr2ogr -f KML barrios.kml PG:"host=localhost dbname=gis user=alumno password=alumn0" -s_srs
EPSG:32616 -t_srs EPSG:4326
```

Cargar fichero CSV

Vamos a usar el fichero con los incendios detectados en las últimas 24 horas por Modis. Está en http://firms.modaps.eosdis.nasa.gov/active_fire/text/Global_24h.csv

Ahora, podemos elegir una de dos opciones:

- Crear a mano una tabla con los campos necesarios y usar el comando COPY de PostgreSQL para copiar directamente el CSV.
- Crear un fichero VRT a partir del CSV y cargar con ogr2ogr dicho fichero VRT

Para el primer caso, la tabla a crear es como sigue:

```
# CREATE TABLE incendios_modis_24h (
ogc_fid integer NOT NULL,
the_geom public.geometry(Point,3857),
latitude character varying,
longitude character varying,
brightness character varying,
scan character varying,
track character varying,
acq_date character varying,
acq_time character varying,
satellite character varying,
confidence character varying,
version character varying,
bright_t31 character varying,
frp character varying
);
```

Y la línea a ejecutar desde psql o pgAdmin III:

```
# COPY incendios_modis24h FROM '/path/to/csv/file/incendios_modis.csv' WITH DELIMITER
↳ ';' CSV HEADER;
```

Para el caso de usar ogr2ogr, primero creamos el VRT:

```
<OGRVRTDataSource>
  <OGRVRTLayer name="Global_24h">
    <SrcDataSource>Global_24h.csv</SrcDataSource>
    <GeometryType>wkbPoint</GeometryType>
    <LayerSRS>EPSG:4326</LayerSRS>
    <GeometryField encoding="PointFromColumns" x="longitude" y="latitude"/
↳ >
  </OGRVRTLayer>
</OGRVRTDataSource>
```

Y luego ejecutamos ogr2ogr:

```
# ogr2ogr -a_srs epsg:4326 -f "PostgreSQL" PG:"dbname=taller_semana_geomatica_
↳ host=localhost user=postgres password=postgres port=5432" incendios_modis.vrt
```

Importación datos OSM a PostGIS

OpenStreetMap (también conocido como OSM) es un proyecto colaborativo para crear mapas libres y editables.

Los mapas se crean utilizando información geográfica capturada con dispositivos GPS móviles, ortofotografías y otras fuentes libres. Esta cartografía, tanto las imágenes creadas como los datos vectoriales almacenados en su base de datos, se distribuye bajo licencia abierta Open Database Licence (ODbL).

OSM dispone de un modelo de datos particular que no responde al modelo característico de los SIG. Este está compuesto de:

- Node
- Way
- Relation

a diferencia de las geometrías características como:

- Punto
- Línea
- Polígono

una característica particular es la ausencia de polígonos dentro del modelo, estos se realizan mediante la asignación de una relación a una línea cerrada. Esta particularidad no impide que los datos de OSM puedan ser adaptados al modelo de geometrías normal mediante cargadores de datos OSM. A continuación se presentan dos de los más utilizados

osm2pgsql

Mediante el uso de este programa podremos incorporar en nuestra base de datos los datos obtenidos desde OSM. Una vez que hemos realizado la importación, aparecerán en nuestra base de datos las tablas que serán el resultado de esta importación:

- *planet_osm_point*

- *planet_osm_line*
- *planet_osm_polygon*
- *planet_osm_roads*

Al disponer el modelo de OSM de cientos de etiquetas, la importación crea en las tablas un gran número de campos de los que la mayoría tendrán valor NULL.

La ejecución se realiza desde la consola:

```
$ osm2pgsql [opciones] ruta_fichero.osm otro_fichero.osm
$ osm2pgsql [opciones] ruta_planet.[gz, bz2]
```

algunas de las opciones se detallan a continuación:

- *-H* Servidor PostGIS
- *-P* <puerto> Puerto
- *-U* <usuario> Usuario
- *-W* pregunta la password del usuario
- *-d* <base_de_datos> base de datos de destino
- *-a* añade datos a las tablas importadas anteriormente
- *-l* almacena las coordenadas en latitud/longitud en lugar de Spherical Mercator
- *-s* utiliza tablas secundarias para la importación en lugar de hacerlo en memoria
- *-S* <fichero_de_estilos> ruta al fichero que indica las etiquetas de OSM que se quiere importar
- *-v* modo verborrea, muestra la salida de las operaciones por consola

En caso de no disponer del SRID 900913 en nuestro PostGIS podremos utilizar la definición que hay de él en `osm2pgsql`. Simplemente ejecutaremos el script `900913.sql`

Práctica

Vamos a exportar datos de OpenStreetMap y cargarlos en PostGIS con `osm2pgsql`. Para ello, vamos primero a <http://www.openstreetmap.org/export#>

Veremos que, si el área a exportar es muy grande, la página nos redireccionará a servicios de descarga masiva, como <http://download.geofabrik.de>. Pero, **ojo**: si hay muchos datos y la máquina no es muy potente, puede tardar mucho en cargarlos.

Una vez hemos descargado lo que queremos, vamos a proceder a activar en PostGIS la extensión `hstore`. Esto permite la creación de una nueva estructura de almacenamiento en PostGIS llamada `hstore`. No es más que una estructura de datos pensada para almacenar en una columna un dato de tipo *clave => valor*. Gracias a ello, podremos usar etiquetas en las consultas que lancemos:

```
# SELECT way, tags FROM planet_osm_polygon WHERE (tags -> 'landcover') = 'trees';
```

Para tener más información, ir a <http://wiki.openstreetmap.org/wiki/Osm2pgsql#hstore>

Para poder utilizar datos `hstore` primero debemos cargar esa extensión:

```
$ sudo apt-get install postgresql-contrib-9.1
```

y después ejecutar:

```
# create extension hstore;
```

Seguidamente instalaremos osm2pgsql:

```
$ sudo apt-get install osm2pgsql
```

Para cargar en PostGIS el fichero exportado, ejecutaríamos esta orden (**no ejecutarla**):

```
# osm2pgsql -d gis -U alumno --hstore tegucigalpa.osm
```

El problema es que eso cargaría nuestros datos en una proyección 900913 (WebMercator). Si lo queremos en 4326 (WGS84), la instrucción es:

```
# osm2pgsql -d gis -U alumno --latlong --hstore tegucigalpa.osm
```

Si tras ejecutar la instrucción obtenemos este error:

```
# Projection code failed to initialise
```

El problema es que osm2pgsql no sabe dónde buscar las definiciones de los sistemas de coordenadas. Debemos definir la variable de entorno *PROJ_LIB* para que apunte donde es debido. En Linux sería:

```
# export PROJ_LIB=/usr/local/share/proj
```

Si da un error al cargar los datos deben cargarse las funcionalidades legacy.sql y crear el operador:

```
CREATE OPERATOR CLASS gist_geometry_ops
    FOR TYPE geometry USING GIST AS
    STORAGE box2df,
    OPERATOR      1      <<      ,
    OPERATOR      2      &<      ,
    OPERATOR      3      &&      ,
    OPERATOR      4      &>      ,
    OPERATOR      5      >>      ,
    OPERATOR      6      ~=      ,
    OPERATOR      7      ~       ,
    OPERATOR      8      @       ,
    OPERATOR      9      &<|     ,
    OPERATOR     10      <<|     ,
    OPERATOR     11      |>>    ,
    OPERATOR     12      |&>    ,

    OPERATOR     13      <-> FOR ORDER BY pg_catalog.float_ops,
    OPERATOR     14      <#> FOR ORDER BY pg_catalog.float_ops,
    FUNCTION      8      geometry_gist_distance_2d (internal, geometry, int4),

    FUNCTION      1      geometry_gist_consistent_2d (internal, geometry, ↵
↵int4),
    FUNCTION      2      geometry_gist_union_2d (bytea, internal),
    FUNCTION      3      geometry_gist_compress_2d (internal),
    FUNCTION      4      geometry_gist_decompress_2d (internal),
    FUNCTION      5      geometry_gist_penalty_2d (internal, internal, ↵
↵internal),
    FUNCTION      6      geometry_gist_picksplit_2d (internal, internal),
    FUNCTION      7      geometry_gist_same_2d (geom1 geometry, geom2 ↵
↵geometry, internal);
```

Cambiamos el esquema a la tabla ya que lo ha creado en public:


```
# ALTER TABLE planet_osm_line SET SCHEMA gis;
# ALTER TABLE planet_osm_point SET SCHEMA gis;
# ALTER TABLE planet_osm_polygon SET SCHEMA gis;
# ALTER TABLE planet_osm_roads SET SCHEMA gis;
```

Crearemos una tabla con las farmacias de Tegucigalpa:

```
# create table gis.farmacias as select * from planet_osm_point where amenity='pharmacy
↪';
```

Y la publicaremos con nuestro GeoServer.

Esto cargaría los datos de OSM en nuestra base de datos. Si nos fijamos en la tabla de polígonos, vemos que tienen definido un campo *population*. Desde QGIS podemos configurar para que solo nos muestre los polígonos con los datos de población, y compararlos con los que hemos metido a mano en la tabla *barrios_de_bogota*, actualizados en 1998.

osmosis

Esta herramienta también realiza la importación de datos desde OSM a PostGIS, pero a diferencia de la anterior, esta mantiene las relaciones entre los objetos de OSM importados. Se recomienda acudir a la documentación de la herramienta para comprender mejor su uso.

Consulta mediante visores web y SIG escritorio

Mediante el uso de diferentes Software tanto de escritorio como de entorno web, accederemos a los datos que hemos importado y podremos tanto visualizarlos como crear servicios web adaptados de estos datos.

Prácticas

Operaciones con QGIS: mostrar tablas de PostGIS, etiquetar, colorear, etc.

Referencias

ogr2ogr [EN] <http://www.gdal.org/ogr2ogr.html>

GDAL [EN] <http://www.gdal.org/>

OpenStreetMap en Wikipedia <http://es.wikipedia.org/wiki/OpenStreetMap>

OpenStreetMap <http://www.openstreetmap.org>

osm2pgsql [EN] <http://wiki.openstreetmap.org/wiki/Osm2pgsql>

osmosis [EN] <http://wiki.openstreetmap.org/wiki/Osmosis>

Cambiar encoding de UTF8 a Latin1 en PostGIS <http://ingdesistemasvzla.blogspot.com.es/2011/02/cambiar-encoding-de-utf-8-latin1-en.html>

CAPÍTULO 10

Indexación espacial

Nota:	Fecha	Autores
	1 Noviembre 2012	■ Micho García (micho.garcia@geomati.co)
	15 Octubre 2013	■ Jorge Arévalo(jorge.arevalo@geomati.co)
	1 Diciembre 2013	■ Micho García (micho.garcia@geomati.co)

©2012 Micho García

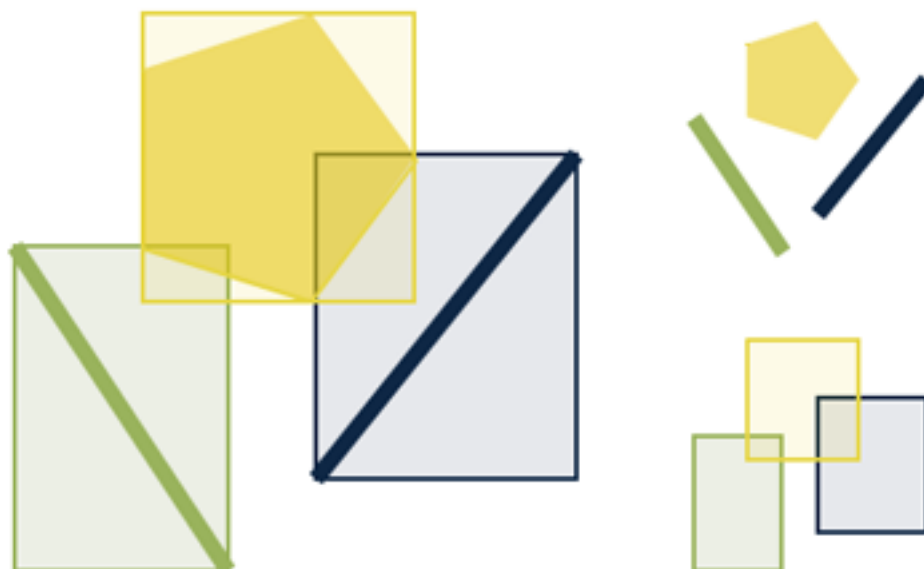
Excepto donde quede reflejado de otra manera, la presente documentación se halla bajo licencia : Creative Commons (Creative Commons - Attribution - Share Alike: <http://creativecommons.org/licenses/by-sa/3.0/deed.es>)

La indexación espacial es una de las funcionalidades importantes de las bases de datos espaciales. Los índices consiguen que las búsquedas espaciales en un gran número de datos sean eficientes. Sin indexación, la búsqueda se realizaría de manera secuencial teniendo que buscar en todos los registros de la base de datos. La indexación organiza los datos en una estructura de árbol que es recorrida rápidamente en la búsqueda de un registro.

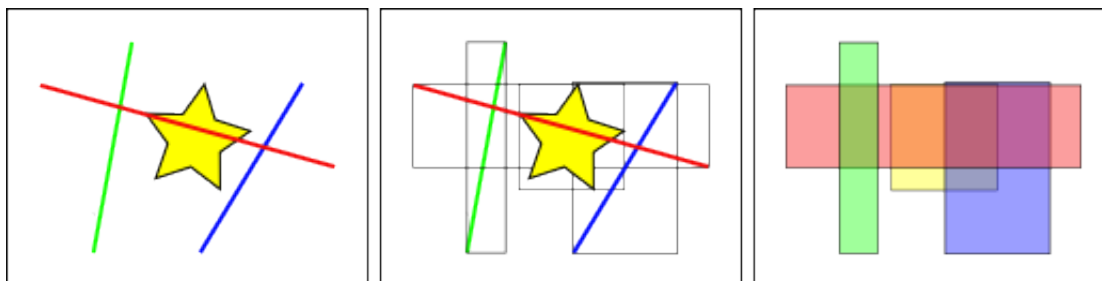
Como funcionan los índices espaciales

Las base de datos estándar crean un árbol jerárquico basados en los valores de las columnas. Los índices espaciales funcionan de una manera diferente, los índices no son capaces de indexar las geometrías, e indexarán las cajas (box) de las geometrías.

Bounding Boxes



La caja (box) es el rectángulo definido por las máximas y mínimas coordenadas X e Y de una geometría.



En la figura se puede observar que solo la línea intersecta a la estrella amarilla, mientras que si utilizamos los índices comprobaremos que la caja amarilla es intersectada por dos figuras la caja roja y la azul. El camino eficiente para responder correctamente a la pregunta **¿qué elemento intersecta la estrella amarilla?** es primero responder a la pregunta **¿qué cajas intersectan la caja amarilla?** usando el índice (consulta rápida) y luego calcular exactamente **¿quien intersecta a la estrella amarilla?** sobre el resultado de la consulta de las cajas.

Creación de índices espaciales

La sintaxis será la siguiente:

```
CREATE INDEX [Nombre_del_indice] ON [Nombre_de_tabla] USING GIST ([campo_de_
→geometria]);
```

Esta operación puede requerir bastante tiempo en tablas de gran tamaño.

Ejemplo

A partir de las capas ríos y suelos calcularemos el número de ríos que intersectan con los tipos de suelo:

```
# select count(*) from suelos su, rios ri where (su.geom && ri.geom) and st_relate(su.
↪geom, ri.geom, 'T*****');

count
-----
5004
```

Esta consulta tardará del orden de 10s

```
# select count(*) from suelos su, rios ri where (su.geom && ri.geom)

count 12739
```

En este último caso ha tardado muy pocos segundos.

Si eliminamos los índices y repetimos la consulta, veremos que el tiempo de cálculo se incrementa

ANALYZE y VACUUM

El planificador de PostGIS se encarga de mantener estadísticas sobre la distribución de los datos de cada columna de la tabla indexada. Por defecto PostgreSQL ejecuta la estadísticas regularmente. Si hay algún cambio grande en la estructura de las tablas, es recomendable ejecutar un ANALYZE manualmente para actualizar estas estadísticas. Este comando obliga a PostgreSQL a recorrer los datos de las tablas con columnas indexadas y actualizar sus estadísticas internas.

No solo con crear el índice y actualizar las estadísticas obtendremos una manera eficiente de manejar nuestras tablas. La operación VACUUM debe ser realizada siempre que un índice sea creado o después de un gran número de UPDATES, INSERTs o DELETEs. El comando VACUUM obliga a PostgreSQL a utilizar el espacio no usado en la tabla que dejan las actualizaciones y los borrados de elementos.

Hacer un VACUUM es crítico para la eficiencia de la base de datos. PostgreSQL dispone de la opción Autovacuum. De esta manera PostgreSQL realizará VACUUMs y ANALYZEs de manera periodica en función de la actividad que haya en la tabla:

```
VACUUM ANALYZE [Nombre_tabla]
VACUUM ANALYZE [Nombre_tabla] ([Nombre_columna])
```

Esta orden actualiza las estadísticas y elimina los datos borrados que se encontraban marcados como eliminados.

Planificador

¿Que pasa si ejecutamos la consulta invirtiendo el orden de los predicados?:

```
# select count(*) from suelos su, rios ri where st_relate(su.geom, ri.geom, 'T*****
↪') and (su.geom && ri.geom);
```

Comprobaremos que el resultado es el mismo, ya que el orden en las consultas es indiferente para PostGIS siendo el planificador el que gestiona esto.

Para utilizar el planificador:

```
EXPLAIN [ ANALYZE ] [ VERBOSE ] sentenciaSQL
```

ANALYZE ejecuta la consulta y muestra el plan de la misma mientras que si no se indica, PostgreSQL realiza una aproximación:

```
# explain analyze select count(*) from suelos su, rios ri where (su.geom && ri.geom)
```

Aggregate (cost=415523.85..415523.86 rows=1 width=0) (actual time=27952.663..27952.664 rows=1 loops=1)

-> Nested Loop (cost=0.00..415494.59 rows=11703 width=0) (actual time=2.200..27944.733 rows=12379 loops=1)

Join Filter: (su.geom && ri.geom) -> Seq Scan on suelos su (cost=0.00..513.71 rows=3871 width=6752) (actual time=0.007..2.607 rows=3871 loops=1) -> Seq Scan on rios ri (cost=0.00..82.09 rows=2009 width=832) (actual time=0.002..0.958 rows=2009 loops=3871)

Total runtime: 27952.715 ms"

```
# create index suelos_geom_gist on suelos using gist(geom); # create index rios_geom_gist on rios using gist(geom);
```

Aggregate (cost=1937.17..1937.18 rows=1 width=0) (actual time=218.263..218.264 rows=1 loops=1)

-> Nested Loop (cost=0.00..1907.91 rows=11703 width=0) (actual time=0.065..213.180 rows=12379 loops=1)

-> Seq Scan on suelos su (cost=0.00..513.71 rows=3871 width=6752) (actual time=0.005..2.644 rows=3871 loops=1) -> **Index Scan using rios_geom_gist on rios ri** (cost=0.00..0.35 rows=1 width=832) (actual time=0.035..0.045 rows=3 loops=3871)

Index Cond: (su.geom && geom)"

Total runtime: 218.310 ms

Operador embebido

La mayor parte de las funciones en PostGIS (ST_Contains, ST_Intersects, ST_DWithin, etc) incluyen un filtrado por índice automáticamente.

Para hacer que una función utilice el índice, hay que hacer uso del operador **&&**. Para las geometrías, el operador **&&** significa "la caja que toca (touch) o superpone (overlap)" de la misma manera que para un número el operador = significa "valores iguales"

En el ejemplo anterior se realizaba la consulta con la función ST_Relate que no hace uso de los índices espaciales, de ahí que le hayamos forzado el uso mediante el operador de caja. ¿Qué resultado muestra el planificador si utilizamos simplemente ST_Intersects?

Aggregate (cost=2885.42..2885.43 rows=1 width=0) (actual time=1791.892..1791.892 rows=1 loops=1)

-> Nested Loop (cost=0.00..2875.66 rows=3901 width=0) (actual time=0.806..1786.258 rows=5004 loops=1)

Join Filter: _st_intersects(su.geom, ri.geom) -> Seq Scan on suelos su (cost=0.00..513.71 rows=3871 width=6752) (actual time=0.007..4.484 rows=3871 loops=1) -> **Index Scan using rios_geom_gist on rios ri** (cost=0.00..0.35 rows=1 width=832) (actual time=0.031..0.045 rows=3 loops=3871)

Index Cond: (su.geom && geom)

Total runtime: 1791.959 ms

Vemos que es un resultado similar al anterior y que hace uso del operador de caja. Esto es porque en la definición de la función, ya lleva embebido la llamada al operador de caja:

```
CREATE OR REPLACE FUNCTION st_intersects(geom1 geometry, geom2 geometry)
    RETURNS boolean AS
'SELECT **$1 && $2** AND _ST_Intersects($1,$2)'
LANGUAGE sql IMMUTABLE
    COST 100;
ALTER FUNCTION st_intersects(geometry, geometry)
OWNER TO alumno;
COMMENT ON FUNCTION st_intersects(geometry, geometry) IS 'args: geomA, geomB -
↳Returns TRUE if the Geometries/Geography "spatially intersect in 2D" - (share any
↳portion of space) and FALSE if they dont (they are Disjoint). For geography --
↳tolerance is 0.00001 meters (so any points that close are considered to intersect)';
```

Esta función en SQL lo que hace es llamar a una función `_ST_Intersects` cuya definición será:

```
CREATE OR REPLACE FUNCTION _st_intersects(geom1 geometry, geom2 geometry)
    RETURNS boolean AS
'$libdir/postgis-2.0', 'intersects'
LANGUAGE c IMMUTABLE STRICT
    COST 100;
ALTER FUNCTION _st_intersects(geometry, geometry)
OWNER TO alumno;
```

Que es la que se comunica directamente con la librería GEOS.

Relaciones espaciales

Nota:	Fecha	Autores
	1 Noviembre 2012	■ Micho García (micho.garcia@geomati.co)
	15 Octubre 2013	■ Jorge Arévalo(jorge.arevalo@geomati.co)
	1 Diciembre 2013	■ Micho García (micho.garcia@geomati.co)

©2012 Micho García

Excepto donde quede reflejado de otra manera, la presente documentación se halla bajo licencia : Creative Commons (Creative Commons - Attribution - Share Alike: <http://creativecommons.org/licenses/by-sa/3.0/deed.es>)

Introducción

Estos métodos lo que hacen es verificar el cumplimiento de determinados predicados geográficos entre dos geometrías distintas. Los predicados geográficos toman dos geometrías como argumento, y devuelven un valor booleano que indica si ambas geometrías cumplen o no una determinada relación espacial. Las principales relaciones espaciales contempladas son equals, disjoint, intersects, touches, crosses, within, contains, overlaps.

Matriz DE-9IM

Estas relaciones o predicados son descritas por la matriz DE-9IM (Dimensionally Extended 9 intersection Matrix), que es una construcción matemática de la rama de la topología.

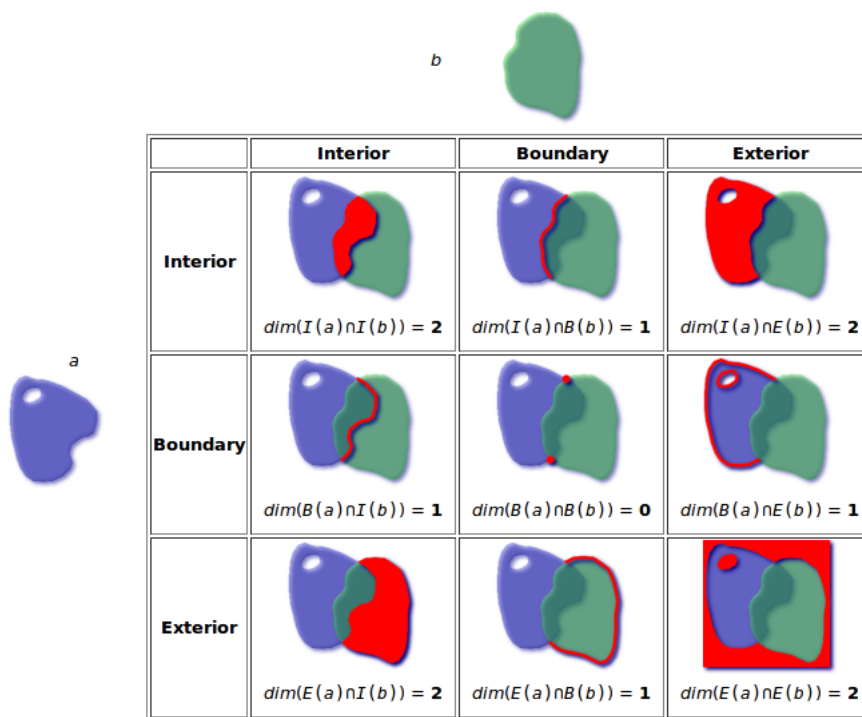


Figura: Mátriz DE-9IM de dos figuras geométricas dadas. Fuente: wikipedia en inglés [7] (<http://en.wikipedia.org/wiki/DE-9IM>)

En esta matriz se muestra las diferentes combinaciones entre las geometrías, su interior, exterior y contorno. Los valores posibles son:

- 0, 1, 2 como los valores de la dimensión
- F como el conjunto vacío.

Los patrones vienen definidos por el uso de los siguientes valores:

- T para cualquier valor.
- F para el conjunto vacío.

Por ejemplo en el caso de una intersección, el patrón que se debe cumplir debe ser:

```
T*****, *T*****, **T***** O ***T*****
```

lo que indica que o el interior de A con el interior de B, o el interior de A con el contorno de B o el interior de A con el exterior de B o el contorno de A con el interior de B deben tener alguna dimensión. Si comprobamos esto en varios casos:

```
# select st_relate('POLYGON ((100 400, 300 400, 300 200, 100 200, 100 400))',
↪ 'LINESTRING (200 450, 130 300, 250 300, 150 150, 200 150)');

# select st_relate('POINT (200 200)', 'LINESTRING (100 400, 200 300, 200 200, 300 100)
↪ ');
```

En el primer caso cumple todos los patrones menos el segundo, y en el segundo caso cumple con el primero. Podemos ver que en ambos casos se cumplen varios patrones, por lo que además de intersectarse, se cruzan y se tocan.

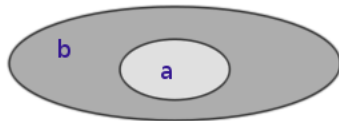
Para el siguiente caso:

```
# select st_relate('LINESTRING (200 400, 200 300, 200 200, 200 100)', 'LINESTRING_
↳ (100 400, 200 300, 200 200, 300 100, 300 200)')
```

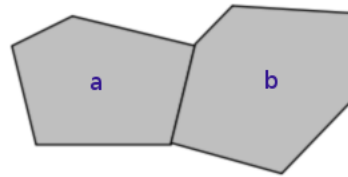
el resultado será: “1F1FF0102”, indicándonos en este caso, que las geometrías se intersecan, pero no se cruzan, ya que la dimensión de sus interiores es igual a 1, mientras que el patrón nos dice que para cruzarse en el caso de dimensiones de las geometrías iguales, la dimensión de sus interiores debe ser 0.

Predicados espaciales

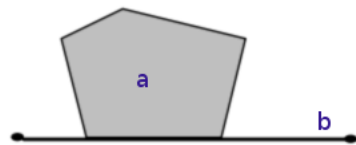
Within(a,b)



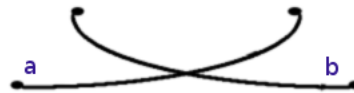
Touches(a,b)



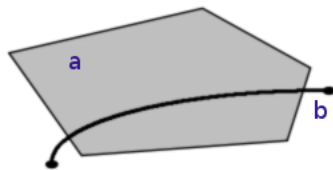
Touches(a,b)



Crosses(a,b)



Crosses(a,b)



Overlaps(a,b)

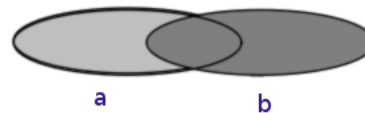


Figura: Ejemplos de predicados espaciales. Fuente: wikipedia. <http://en.wikipedia.org/wiki/File:TopologicSpatialRelarions2.png>

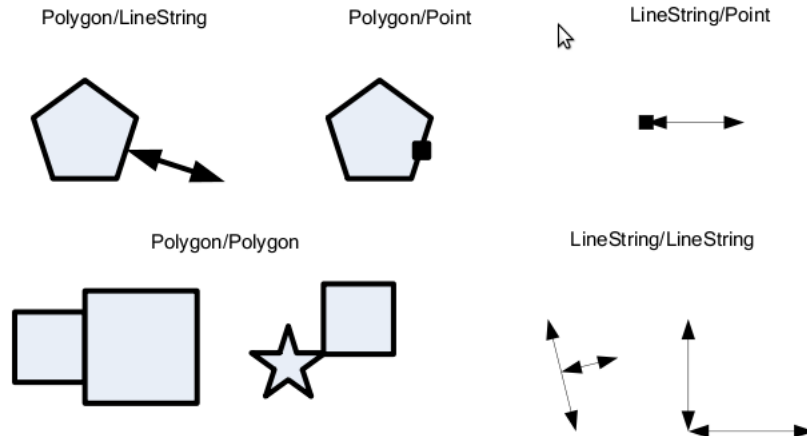
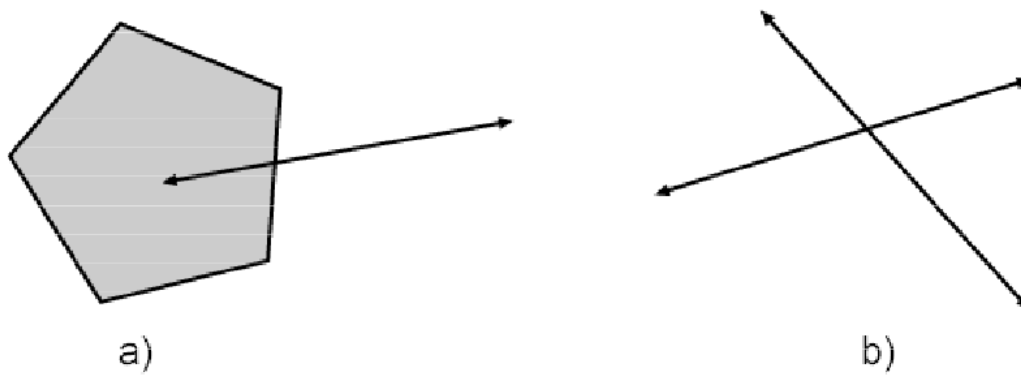


Figure 20: Examples of the Touches relationship

Figura: Ejemplos de la relación “Touch” (toca). Fuente: “OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture”



**Figure 21: Examples of the Crosses relationship
Polygon/LineString (a)
and LineString/LineString (b)**

Figura: Ejemplos de la relación “Crosses” (cruza). Fuente: “OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture”

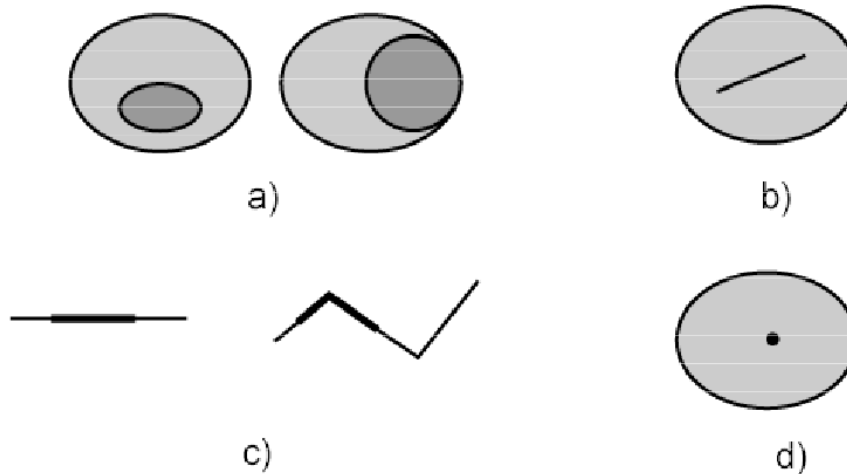


Figure 22: Examples of the “Within” relationship
Polygon/Polygon (a), Polygon/LineString (b), LineString/LineString (c), and Polygon/Point (d)

Figura: Ejemplos de la relación “Within” (contenido en). Fuente: “OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture”

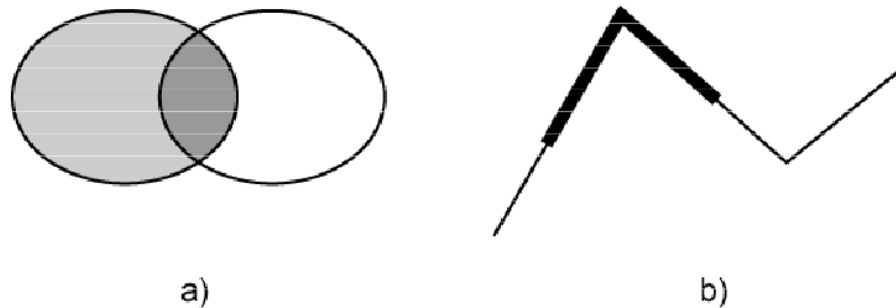


Figure 23: Examples of the Overlaps relationship
Polygon/LineString (a)
and LineString/LineString (b)

Figura: Ejemplos de la relación “Overlaps” (solapa). Fuente: “OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture”

Los principales métodos de la clase Geometry para chequear predicados espaciales entre la geometría en cuestión y otra proporcionada como parámetro son:

- **Equals (A, B):** Las geometrías son iguales desde un punto de vista topológico
- **Disjoint (A, B):** No tienen ningún punto en común, las geometrías son disjuntas
- **Intersects (A, B):** Tienen por lo menos un punto en común. Es el inverso de Disjoint
- **Touches (A, B):** Las geometrías tendrán por lo menos un punto en común del contorno, pero no puntos interiores
- **Crosses (A, B):** Comparten parte, pero no todos los puntos interiores, y la dimensión de la intersección es menor que la dimensión de al menos una de las geometrías
- **Contains (A, B):** Ningún punto de B está en el exterior de A. Al menos un punto del interior de B está en el interior de A
- **Within (A, B):** Es el inverso de Contains. Within(B, A) = Contains (A, B)

- **Overlaps (A, B):** Las geometrías comparten parte pero no todos los puntos y la intersección tiene la misma dimensión que las geometrías.
- **Covers (A, B):** Ningún punto de B está en el exterior de A. B está contenido en A.
- **CoveredBy (A, B):** Es el inverso de Covers. CoveredBy(A, B) = Covers(B, A)

ST_Equals

ST_Equals(geometry A, geometry B), comprueba que dos geometrías sean espacialmente iguales.

ST_Equals devuelve TRUE si dos geometrías del mismo tipo tienen idénticas coordenadas x,y.

Ejemplo

Sobre la capa países vecinos:

```
# select st_equals(tabla.geom, pv.geom) as iguales, pv.definicion from gis.países_
↪vecinos as pv, (select geom from gis.países_vecinos where definicion = 'Guatemala')
↪as tabla
```

ST_Intersects, ST_Disjoint, ST_Crosses y ST_Overlaps

Comprueban la relación entre los interiores de las geometrías.

ST_Intersects(geometry A, geometry B)

Devuelve TRUE si la intersección no es un resultado vacío.

Ejemplo

```
# select count(*) as tramos_rio from (select * from gis.honduras_departamentos as de)
↪as departamentos, (select * from gis.rios) as rios where st_
↪intersects(departamentos.geom, rios.geom) and departamentos.name_1 = 'Choluteca'
```

ST_Disjoint

ST_Disjoint(geometry A , geometry B)

Es el inverso de ST_Intersects. indica que dos geometrías no tienen ningún punto en común. Es menos eficiente que ST_Intersects ya que esta no está indexada. Se recomienda comprobar NOT ST_Intersects

Ejemplo

Comprobar los departamentos que no tocan Choluteca

```
# create table gis.disjuntos as select de.gid, de.name_1, de.geom from gis.honduras_
↪departamentos as de, (select * from gis.honduras_departamentos where name_1 =
↪'Choluteca') as departamento where st_disjoint(de.geom, departamento.geom)
```

Realizar la misma operación utilizando NOT ST_Intersects

ST_Crosses

```
ST_Crosses(geometry A, geometry B)
```

Se cumple esta relación si el resultado de la intersección de dos geometrías es de dimensión menor que la mayor de las dimensiones de las dos geometrías y además esta intersección está en el interior de ambas.

Ejemplo

```
# select count(*) as tramos_rio from (select * from gis.honduras_departamentos as de)
↳as departamentos, (select * from gis.rios) as rios where st_crosses(departamentos.
↳geom, rios.geom) and departamentos.name_1 = 'Choluteca'
```

ST_Overlap

```
ST_Overlaps(geometry A, geometry B)
```

compara dos geometrías de la misma dimensión y devuelve TRUE si su intersección resulta una geometría diferente de ambas pero de la misma dimensión

ST_Touches

```
ST_Touches(geometry A, geometry B)
```

Devuelve TRUE si cualquiera de los contornos de las geometrías se cruzan o si sólo uno de los interiores de la geometría se cruza el contorno del otro.

Ejemplo

```
# create table gis.juntos as select de.gid, de.name_1, de.geom from gis.honduras_
↳departamentos as de, (select * from gis.honduras_departamentos where name_1 =
↳'Choluteca') as departamento where st_touches(de.geom, departamento.geom)
```

ST_Within y ST_Contains

```
ST_Within(geometry A , geometry B)
```

es TRUE si la geometría A está completamente dentro de la geometría B. Es el inverso de ST_Contains

```
ST_Contains(geometry A, geometry B)
```

Devuelve TRUE si la geometría B está contenida completamente en la geometría A

Ejemplo

```
¿Cuántas escuelas hay en el Departamento de Valle?

# select count(*) from (select * from gis.honduras_departamentos where name_1 = 'Valle
↪') as departamento, (select * from gis.edificaciones where descripc like '%scuela%'
↪) as escuelas where ST_Contains(departamento.geom, escuelas.geom)
```

ST_Distance and ST_DWithin

```
ST_Distance(geometry A, geometry B)
```

Calcula la menor distancia entre dos geometrías.

```
ST_DWithin(geometry A, geometry B, distance)
```

Permite calcular si dos objetos se encuentran a una distancia dada uno del otro.

Ejemplo

```
Calcular las edificaciones del municipio de Choluteca que se encuentran a menos de 1
↪Km de la mina
```

Primero creamos una tabla solo con las edificaciones del municipio para acelerar los procesos de análisis:

```
create table gis.ed_choluteca as select ed.*
from gis.honduras_municipios as mu, gis.edificaciones as ed
where ST_Contains(mu.geom, ed.geom) and mu.municipio = 'Choluteca';

create index ed_choluteca_gist on gis.ed_choluteca using gist(geom);

select count(*), descripc, tipo from gis.ed_choluteca group by descripc, tipo;
```

Creamos la tabla con las edificaciones que hay a menos de 1000m de la mina, edificación tipo=12.:

```
create table gis.ed_near_mina as select ed.geom, ed.tipo, ed.descripc
from gis.ed_choluteca as ed, (select geom from gis.ed_choluteca where tipo = 12) as
↪mina
where ST_DWithin(ed.geom, mina.geom, 1000);
```

El uso del tipo **geography** para medir distancias, no obstante, es el recomendado cuando se trata de medir la distancia entre dos puntos de la Tierra muy distantes entre sí.

En estos casos, un sistema de referencia plano no es una buena elección. Estos sistemas suelen dar buenos resultados a la hora de mostrar mapas en planos, porque conservan las direcciones, pero las distancias y áreas pueden estar bastante distorsionadas con respecto a la realidad. Es necesario utilizar un sistema de referencia espacial que conserve las distancias, teniendo en cuenta la curvatura terrestre. El tipo **geography** de PostGIS es un buen ejemplo, puesto que realiza los cálculos sobre una esfera, y no sobre un esferoide.

JOINS espaciales

Permite combinar información de diferentes tablas usando relaciones espaciales como clave dentro del JOIN. Es una de las características más potentes de las bases de datos espaciales.

Veamos un ejemplo: Los nombres de los municipios del departamento Yoro

```
# select mu.municipio, mu.depart from gis.honduras_municipios as mu, gis.honduras_
↳ departamentos as de where st_contains(de.geom, mu.geom) and de.name_1 = 'Yoro'
↳ order by mu.municipio
```

Cualquier función que permita crear relaciones TRUE/FALSE entre dos tablas puede ser usada para manejar un JOIN espacial, pero comunmente las más usadas son:

- ST_Intersects
- ST_Contains
- ST_DWithin

JOIN y GROUP BY

El uso de las relaciones espaciales junto con funciones de agregación, como **group by**, permite operaciones muy poderosas con nuestros datos. Veamos un ejemplo sencillo: EL número de municipios de los departamentos de Honduras:

```
# select de.name_1, count(*) from gis.honduras_departamentos as de, gis.honduras_
↳ municipios as mu where ST_Contains(de.geom, mu.geom) group by de.name_1
```

1. La clausula JOIN crea una tabla virtual que incluye los datos de los departamentos y de los municipios
2. Las filas resultantes son agrupadas por el nombre del barrio y rellenas con la función de agregación count().

CAPÍTULO 12

Validación

Nota:	Fecha	Autores
	1 Noviembre 2012	■ Micho García (micho.garcia@geomati.co)
	15 Octubre 2013	■ Jorge Arévalo(jorge.arevalo@geomati.co)
	1 Diciembre 2013	■ Micho García (micho.garcia@geomati.co)

©2012 Micho García

Excepto donde quede reflejado de otra manera, la presente documentación se halla bajo licencia : Creative Commons (Creative Commons - Attribution - Share Alike: <http://creativecommons.org/licenses/by-sa/3.0/deed.es>)

Validar geometrías

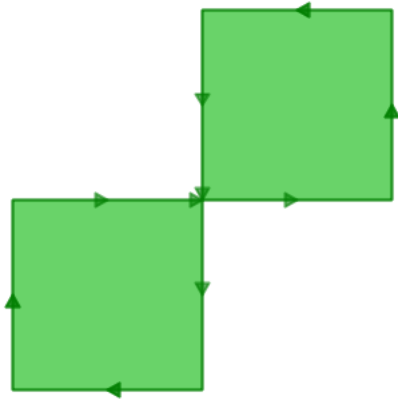
Una operación común cuando se trabaja con datos vectoriales es validar que dichos datos cumplen ciertas condiciones que los hacen óptimos para realizar análisis espacial sobre los mismos. O de otra forma, que cumplen ciertas condiciones topológicas.

Los puntos y las líneas son objetos muy sencillos. Intuitivamente, podemos afirmar que no hay manera de que sean *topológicamente inválidos*. Pero un polígono es un objeto más complejo, y debería cumplir ciertas condiciones. Y debe cumplirlas porque muchos algoritmos espaciales son capaces de ejecutarse rápidamente gracias a que asumen una consistencias de los datos de entrada. Si tuviéramos que forzar a que esos algoritmos revisaran las entradas, serían mucho más lentos.

Veamos un ejemplo de porqué esto es importante. Supongamos que tenemos este polígono sencillo:

```
# POLYGON((0 0, 0 1, 2 1, 2 2, 1 2, 1 0, 0 0));
```

Gráficamente:



Podemos ver el límite exterior de esta figura como un símbolo de *infinito* cuadrado. O sea, que tiene un *lazo* en el medio (una intersección consigo mismo). Si quisiéramos calcular el área de esta figura, podemos ver intuitivamente que tiene 2 unidades de área (si hablamos de metros, serían 2 metros cuadrados).

Veamos qué piensa PostGIS del área de esta figura:

```
# SELECT ST_Area(ST_GeometryFromText('POLYGON((0 0, 0 1, 1 1, 2 1, 2 2, 1 2, 1 1, 1 0,
↪ 0 0))'));
```

El resultado será:

```
# st_area
-----
0
```

¿Qué es lo que ha sucedido aquí?

El algoritmo de cálculo de áreas de PostGIS (muy rápido) asume que los anillos no van a intersectar consigo mismos. Un anillo que cumpla las condiciones adecuadas para el análisis espacial, debe tener el área que encierra **siempre** en el mismo lado. Sin embargo, en la imagen mostrada, el anillo tiene, en una parte, el área encerrada en el lado izquierdo. Y en la otra, el área está encerrada en el lado derecho. Esto causa que las áreas calculadas para cada parte del polígono tengan valores opuestos (1 y -1) y se anulen entre sí.

Este ejemplo es muy sencillo, porque podemos ver rápidamente que el polígono es inválido, al contener una intersección consigo mismo (algo que ESRI permite en un SHP, pero PostGIS no, porque implementa SFSQL: <http://www.opengeospatial.org/standards/sfs>). Pero, ¿qué sucede si tenemos millones de polígonos? Necesitamos una manera de detectar si son válidos o inválidos. Afortunadamente, PostGIS tiene una función para esto: **ST_IsValid**, que devuelve TRUE o FALSE:

```
# SELECT ST_IsValid(ST_GeometryFromText('POLYGON((0 0, 0 1, 1 1, 2 1, 2 2, 1 2, 1 1,
↪ 1 0, 0 0))'))
```

Devuelve:

```
# st_isvalid
-----
f
```

Incluso tenemos una función que nos dice la razón por la que una geometría es inválida:

```
# SELECT ST_IsValidReason(ST_GeometryFromText('POLYGON((0 0, 0 1, 1 1, 2 1, 2 2, 1 2,
↪1 1, 1 0, 0 0))'));;
```

Que devuelve:

```
# st_isvalidreason
-----
Self-intersection[1 1]
```

Mediante el uso de la función `st_isvaliddetail` podremos obtener la información similar de las funciones anteriores de una sola llamada. Esta función dispone de la opción de no tener en cuenta los anillos autointersectados que son permitidos en el modelo ESRI.

Diferencias entre el modelo ESRI y el modelo propuesto por OGC

Estas son algunas de las diferencias entre el modelo ESRI y el OGC:

- En el modelo ESRI los anillos de los polígonos tienen un sentido concreto mientras que en el OGC no.
- ESRI no permite puntos repetidos, en OGC sí.
- PostGIS permite geometrías nulas, ESRI no.
- PostGIS permite geometrías vacías, mientras que ESRI no.
- ESRI permite polígonos lazo

Por lo que conviene realizar operaciones de validación de geometrías después de la exportación de datos de ESRI y antes de la importación de datos a ESRI:

Conviene realizar operaciones antes de intercambiar datos entre ambos modelos:

De PostGIS a ESRI:

- Eliminar geometrías nulas
- Eliminar vértices repetidos
- Eliminar geometrías vacías
- Corregir geometrías no válidas

De ESRI a PostGIS:

- Comprobar validez geometrías
- Corregir no validez

Por ejemplo, si tenemos la geometría:

```
POLYGON ((20 30, 30 30, 30 20, 20 20, 20 30), (22 28, 28 28, 22 24.8, 28.1 22, 22 22,
↪22 28))
```

y ejecutamos:

```
# select st_isvaliddetail(st_geomfromtext('POLYGON ((20 30, 30 30, 30 20, 20 20, 20
↪30), (22 28, 28 28, 22 24.8, 28.1 22, 22 22, 22 28))'))
```

obtendremos un `valid_detail(valid, reason, location)`:

```
(f, "Ring Self-intersection", 0101000000000000000003640CDCCCCCCCCC3840)
```

si queremos acceder a cada una de los valores:

```
valid((f, "Ring Self-intersection", 010100000000000000003640CDCCCCCCCC3840)) nos_
↳ devolverá f
```

Activando el “ESRI Flag”:

```
# select st_isvaliddetail(st_geomfromtext('POLYGON ((20 30, 30 30, 30 20, 20 20, 20
↪30), (22 28, 28 28, 22 24.8, 28.1 22, 22 22, 22 28))'), 1)
```

el resultado será:

 $(t, ,)$

Esto es así porque mediante este flag, PostGIS marcará como válidos los polígonos con anillos que se encuentran autointersectados.

Práctica

Comprobar el funcionamiento de ST_MakeValid

Partiremos del polígono con forma de lazo mostrado anteriormente:

POLYGON((0 0, 0 1, 2 1, 2 2, 1 2, 1 0, 0 0))

Como vimos se trataba de una geometría no válida por lo que procederemos a su corrección mediante el uso de la función `ST_MakeValid`:

```
# SELECT ST_AsText(ST_MakeValid(ST_GeomFromText('POLYGON((0 0, 0 1, 2 1, 2 2, 1 2, 1 2, 0 0))')));
```

Que dará como resultado un MULTIPOLYGON:

```
MULTIPOLYGON(((0 0,0 1,1 1,1 0,0 0)),((1 1,1 2,2 2,2 1,1 1)))
```

Si quisiera mantener esta geometría en mi tabla debería:

```
# SELECT (st_dump(ST_MakeValid(ST_GeomFromText('POLYGON((0 0, 0 1, 2 1, 2 2, 1 2, 1 0,
→ 0 0)'))))).geom
```

Nos devolverá las geometrías tipo polígono.

En algunos casos el uso de la función `ST_Buffer` con valor 0 resuelve el problema de validez, pero podemos ver en este caso que esto a veces no es posible:

```
# SELECT ST_AsText(ST_buffer(ST_GeomFromText('POLYGON((0 0, 0 1, 2 1, 2 2, 1 2, 1 0, 0 0))'), 0));
```

POLYGON ((1 1, 1 2, 2 2, 2 1, 1 1))

Comprobar como el uso de ST_Collect genera geometrías no válidas

La función ST_Collect genera geometrías MULTI a partir de geometrías sencillas. Lo hace sin tener en cuenta la validez de la geometría resultado. Hay que recordar que para que una geometría MULTI sea válida, todas las geometrías que la compongan deben ser válidas. Si tenemos dos geometrías que en origen son válidas:

```
POLYGON ((0 2, 1 2, 1 1, 0 1, 0 2))' & POLYGON ((0.5 1.5, 1.5 1.5, 1.5 0.5, 0.5 0.5, 0.5 1.5))
```

Utilizando la función ST_Collect podremos generar una geometría inválida:

```
#SELECT ST_isvaliddetail(ST_Collect(ST_GeomFromText('POLYGON ((0 2, 1 2, 1 1, 0 1, 0 2))'), ST_GeomFromText('POLYGON ((0.5 1.5, 1.5 1.5, 1.5 0.5, 0.5 0.5, 0.5 1.5))'))
(f,Self-intersection,010100000000000000000000F03F000000000000F83F)
```

¿Qué pasaría si utilizase la función ST_MakeValid en este caso?.

Generar una restricción para evitar la carga de geometrías no válidas

Podemos evitar que se introduzcan datos no válidos en nuestras tablas simplemente mediante el uso de restricciones. Si creamos la siguiente tabla:

```
# CREATE TABLE gis.geometries
(
  gid serial NOT NULL,
  geom geometry(Polygon),
  CONSTRAINT pk_geometries PRIMARY KEY (gid )
)
```

Añadiremos una restricción que no permita la inclusión de geometrías no válidas:

```
# ALTER TABLE gis.geometries
ADD CONSTRAINT geometry_valid_check
CHECK (ST_IsValid(geom));
```

Al tratar de insertar un polígono no válido la restricción no lo permitirá:

```
# INSERT INTO gis.geometries (geom) VALUES (ST_GeomFromText('POLYGON((0 0, 0 1, 2 1, 2 2, 1 2, 1 0, 0 0))'));

NOTICE: Self-intersection at or near point 1 1
ERROR: new row for relation "geometries" violates check constraint "geometry_valid_check"

***** Error *****

ERROR: new row for relation "geometries" violates check constraint "geometry_valid_check"
SQL state: 23514
```

Advertencia: Estas restricciones pueden afectar a procesos de carga masivos mediante scripts.

Comprobar la validez de las geometrías del shapefile *world_borders*

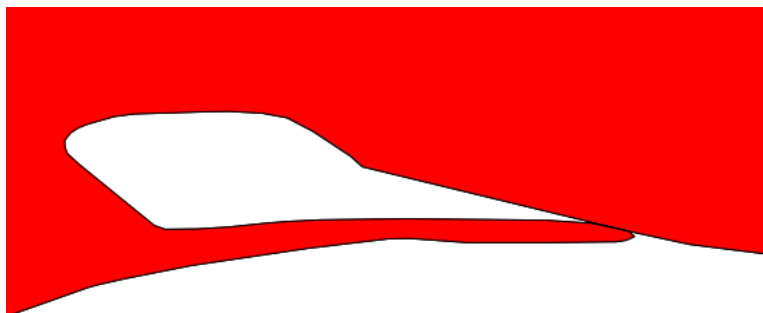
Nota: Puedes obtener los datos desde aquí[1]

```
# SELECT gid, name, ST_IsValidReason(geom) FROM tm_world_borders WHERE ST_
↪IsValid(geom)=false;
```

Obtenemos el resultado:

#	gid	name	st_isvalidreason
24	Canada	Ring	Self-intersection[-53.756367 48.5032620000001]
33	Chile	Ring	Self-intersection[-70.917236 -54.708618]
155	Norway	Ring	Self-intersection[5.33694400000002 61.592773]
175	Russia	Ring	Self-intersection[143.661926 49.31221]

Observamos que hay 4 polígonos con intersecciones consigo mismos. Esto es un ejemplo del aspecto que tienen estas auto-intersecciones:



Para resolver estos errores topológicos, tenemos a nuestra disposición la función *ST_MakeValid*. Esta función es nueva en PostGIS 2.0. Hasta entonces, estos problemas se resolvían con técnicas como hacer un buffer de tamaño 0 alrededor de la geometría inválida, y dejar que la función *ST_Buffer* la arreglara. Esto es así porque *ST_Buffer* en realidad construye una nueva geometría réplica de la antigua y construyendo un buffer alrededor de ella. Si este buffer es de tamaño 0, el resultado es solo la réplica de la anterior geometría. Pero al ser construida siguiendo las reglas topológicas de OGC, solucionaba muchos problemas como éste.

La función *ST_MakeValid* es más apropiada para arreglar geometrías. Únicamente requiere **GEOS 3.3.0** o superior para funcionar (**GEOS 3.3.4** si estamos usando PostGIS 2.1). Para saber qué versión de GEOS tenemos instalada basta con ejecutar:

```
# SELECT postgis_full_version()
```

Si se tiene una versión de GEOS inferior a la 3.3.0, se pueden seguir los consejos de Paul Ramsey: <http://blog.opengeo.org/2010/09/08/tips-for-the-postgis-power-user/>

Para comprobar el funcionamiento de *ST_MakeValid* vamos a crear una tabla nueva donde almacenemos únicamente uno de los polígonos conflictivos, marcado como *erroneo*. A continuación, crearemos un nuevo registro en dicha tabla con el polígono corregido.

Para hacerlo, ejecutemos esta query, que es algo compleja. Como sabemos que el problema es una auto-intersección que forma un anillo, vamos a *desmontar* la geometría en su lista de anillos y quedarnos solo con aquel que intersecta con el punto donde se detectó el error:

```
# SELECT * INTO invalid_geometries
FROM (
SELECT 'broken'::varchar(10) as status,
ST_GeometryN(geom, generate_series(1, ST_NRings(geom))::geometry(Polygon,4326) as
↳the_geom
FROM tm_world_borders
WHERE name = 'Chile') AS foo
WHERE ST_Intersects(the_geom, ST_SetSRID(ST_Point(-70.917236,-54.708618), 4326));
```

Con eso hemos creado la tabla *invalid_geometries* y añadido el anillo que contiene el error. Ahora añadamos un nuevo registro con el resultado de llamar a *ST_MakeValid* sobre el polígono erróneo:

```
# INSERT INTO invalid_geometries
VALUES ('repaired', (SELECT ST_MakeValid(the_geom) FROM invalid_geometries));
```

La función *ST_MakeValid*, realmente solo ha añadido un anillo más a la geometría inválida, para hacerla válida. Lo podemos comprobar con:

```
# SELECT status, ST_NRings(the_geom) FROM invalid_geometries;
```

Que devuelve:

```
# status | st_nrings
-----+-----
broken   |         1
repaired |         2
```

Ahora que ya hemos comprobado cómo funciona *ST_MakeValid*, podemos arreglar todas las geometrías inválidas:

```
# UPDATE tm_world_borders
SET the_geom = ST_MakeValid(the_geom)
WHERE ST_IsValid(the_geom) = false;
```

Una manera de evitar tener tablas con geometrías inválidas es definir una *constraint* que lo impida:

```
# CREATE TABLE

# ALTER TABLE tm_world_borders
ADD CONSTRAINT geometry_valid_check
CHECK (ST_IsValid(geom));
```

Nota:	Fecha	Autores
	1 Noviembre 2012	■ Micho García (micho.garcia@geomati.co)
	15 Octubre 2013	■ Jorge Arévalo(jorge.arevalo@geomati.co)
	1 Diciembre 2013	■ Micho García (micho.garcia@geomati.co)

©2012 Micho García

Excepto donde quede reflejado de otra manera, la presente documentación se halla bajo licencia : Creative Commons (Creative Commons - Attribution - Share Alike: <http://creativecommons.org/licenses/by-sa/3.0/deed.es>)

El análisis de datos con SIG tiene por finalidad descubrir estructuras espaciales, asociaciones y relaciones entre los datos, así como modelar fenómenos geográficos. Los resultados reflejan la naturaleza y calidad de los datos, además de la pertinencia de los métodos y funciones aplicadas. Las tareas y transformaciones que se llevan a cabo en el análisis espacial precisan datos estructurados, programas con las funciones apropiadas y conocimientos sobre la naturaleza del problema, para definir los métodos de análisis.

El proceso convierte los datos en información útil para conocer un problema determinado. Es evidente que los resultados del análisis espacial añaden valor económico y, sobre todo, información y conocimiento a los datos geográficos

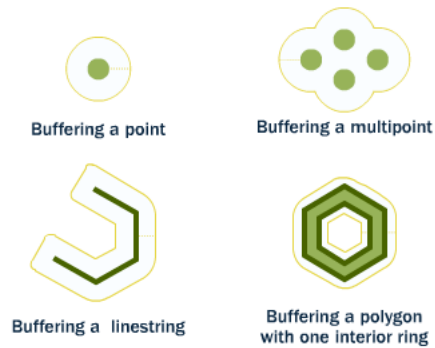
Operadores espaciales

Estos son los encargados de realizar operaciones geométricas entre las geometrías que se les pasa como argumentos. Están definidos en la norma SFA y PostGIS soporta todos ellos.

Buffer

Es el conjunto de puntos situados a una determinada distancia de la geometría

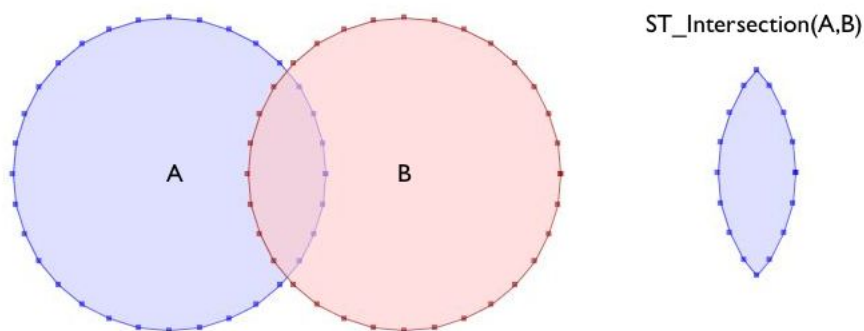
ST_Buffer



Acepta distancias negativas, pero estas en líneas y puntos devolverán el conjunto vacío.

Intersección

Genera una geometría a partir de la intersección de las geometrías que se les pasa como parámetros.



¿Cuál es el area en común de dos círculos situados en los puntos (0 0) y (3 0) de radio 2?:

```
SELECT ST_AsText(ST_Intersection(
  ST_Buffer('POINT(0 0)', 2),
  ST_Buffer('POINT(3 0)', 2)
));
```

Práctica

Trabajaremos con la capa honduras_carreteras y edificaciones. Comprobaremos en el estado de Olancho, cuantas casas se encuentran afectadas por el ruido de este tipo de vías. Para determinar si una vivienda se ve afectada por el ruido, tomaremos como valor 500m el mínimo recomendado para vivir alejado de este tipo de vías.

Primero crearemos una capa con las vías de este tipo que se encuentren en Choluteca:

```
# SELECT * INTO gis.viasl_choluteca
FROM (
  SELECT ca.gid, ca.tipo, ca.geom FROM gis.honduras_carreteras as ca,
```

```
gis.honduras_departamentos as de
WHERE ST_Intersects(ca.geom, de.geom)
AND de.name_1 like 'Choluteca'
AND ca.tipo like 'Primary Route'
) as foo
```

Ahora calcularemos el buffer de 500m para las vías y lo guardaremos en otra tabla:

```
# SELECT * INTO gis.buffer_vias1_choluteca
FROM (
    SELECT ca.gid, ca.tipo, ST_Buffer(ca.geom, 500, 'endcap=round join=round
→)::geometry('POLYGON', 32616) as geom FROM gis.vias1_choluteca as ca
) as buf
```

Y podremos obtener las edificaciones tipo casa que se encuentren a menos de 500m de la vía:

```
# SELECT * INTO gis.casas_afectadas
FROM (
    SELECT casas_cho.gid, casas_cho.geom
    FROM (
        SELECT ed.gid, ed.geom
        FROM gis.edificaciones as ed, gis.honduras_departamentos as dep
        WHERE ST_Intersects(dep.geom, ed.geom)
        AND ed.tipo = 1
        AND dep.name_1 like 'Choluteca'
    ) as casas_cho, gis.buffer_vias1_choluteca as buf
    WHERE ST_Intersects(buf.geom, casas_cho.geom)
) as foobar
```

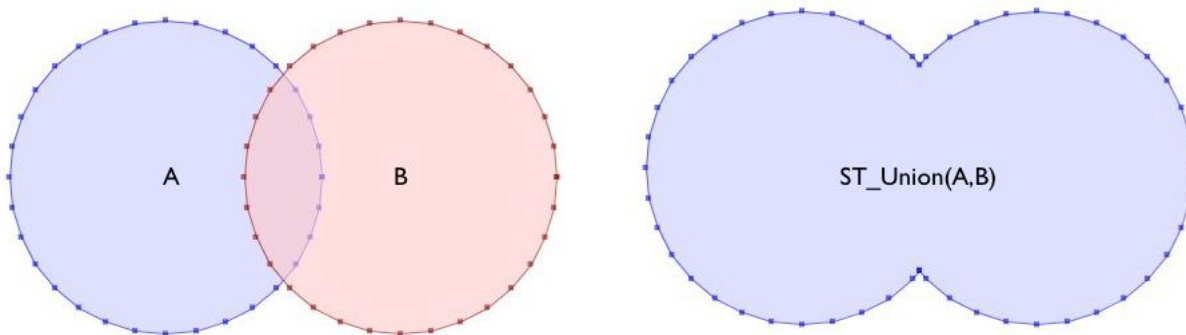
Unión

Al contrario que en el caso anterior, la unión produce una geometría común con las geometrías que se le pasa a la función como argumento. Esta función acepta como parámetro dos opciones, las geometrías que serán unidas:

```
ST_Union(Geometría A, Geometría B)
```

o una colección de geometrías:

```
ST_Union([Geometry])
```



Práctica

Tratar de simplificar todos los municipios de Olancho en un único polígono y comprobar si generan una geometría idéntica a la de su departamento.

Para esto se podría usar la versión agregada **ST_Union**, que toma como entrada un conjunto de geometrías y devuelve la unión de las mismas también como geometría.

La consulta SQL es ésta:

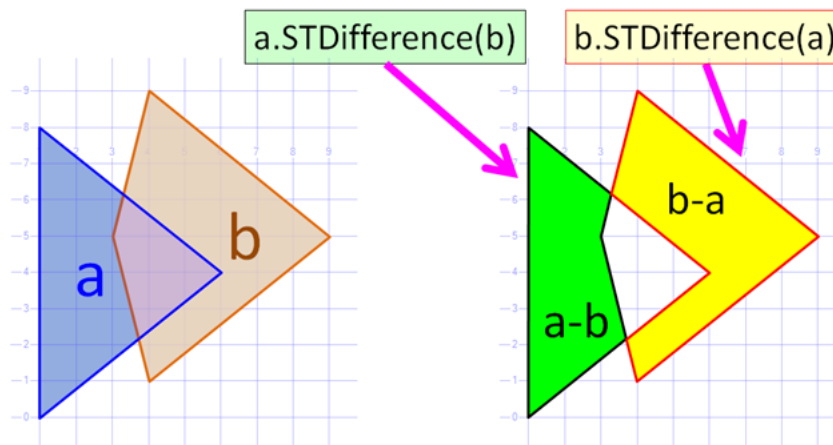
```
# SELECT * INTO gis.municipios_olancho
FROM (
    SELECT mun.depart as nombre, ST_Union(mun.geom) as geom FROM gis.honduras_
    ↪municipios as mun,
    gis.honduras_departamentos as dep
    WHERE mun.depart like 'Olancho' group by mun.depart
) as foo
```

Para comprobar si son iguales:

```
# SELECT ST_Equals(mun.geom, dep.geom)
FROM gis.honduras_departamentos as dep, gis.municipios_olancho as mun
WHERE dep.name_1 like 'Olancho';
```

Diferencia

La diferencia entre dos geometrías A y B, son los puntos que pertenecen a A, pero no pertenecen a B



```
ST_Difference(Geometría A, Geometría B)
```

Diferencia simétrica

Es el conjunto de puntos que pertenecen a A o a B pero no a ambas.

```
ST_SymDifference(Geometría A, Geometría B)
```

Tipos de geometrías devueltas

El tipo de geometrías que devuelven estas operaciones no tienen porque ser igual al tipo de geometrías que le son pasadas como argumentos. Estas operaciones devolverán:

- Una única geometría
- Una geometría *Multi* si está compuesta por varias geometrías del mismo tipo
- Una *GeometryCollection* si está formada por geometrías de distinto tipo.

En este último caso habrá que proceder a una homogeneización de las geometrías que son devueltas, para ello podremos utilizar diferentes estrategias:

- El uso de clausulas de filtrado, por ejemplo indicando que solo se devuelvan aquellas geometrías cuya intersección sea una línea.
- Crear las tablas de salida de tipo *Multi*, en este caso las geometrías que no sean multi podrán ser convertidas a este tipo mediante la función `ST_Multi`
- En caso de que las geometrías devueltas sean tipo *GeometryCollection*, será necesario iterar esta colección, y extraer mediante la función `ST_CollectionExtract` las geometrías en las que estamos interesados, indicándole para ello a la función la dimensión de las geometrías.

Transformación y edición de coordenadas

Mediante el uso de diferentes funciones seremos capaces de manejar transformaciones entre sistemas de coordenadas o hacer reproyecciones de las capas. Para un manejo básico de estas utilizaremos las funciones que PostGIS pone a nuestra disposición:

- `ST_Transform(geometría, srid)`, que nos permite la transformación de la geometría al SRID que le pasamos por parámetro.
- `ST_SRID(geometria)` nos muestra el SRID de la geometría
- `ST_SetSRID(geometria, srid)` asigna el SRID a la geometría pero sin relizar la transformación

En la tabla `spatial_ref_sys` encontraremos la definición de los sistemas de coordenadas de los que disponemos. Podremos consultar la descripción de ellos mediante consultas `select` del estilo:

```
# select * from spatial_ref_sys where srid=4326;
```

Para transformar las geometrías en otros sistemas de coordenadas, lo primero que debemos saber es el sistema de coordenadas de origen y el de destino. Hemos de consultar que estos se encuentran en la tabla `spatial_ref_sys`. En caso de que alguna de nuestras tablas no tenga asignado un SRID, el valor de este será 0, valor por defecto, por lo que habrá que asignarle el sistema de coordenadas antes de la transformación.

Cambio del SRS de una tabla

Tras la importación de una capa mediante `shp2pgsql` nos damos cuenta de que en el momento de la carga no se eligió el SRS de la misma por lo que ahora nos aparece con el SRS por defecto 0 (versiones de PostGIS >= 2.0). Para no realizar la importación de nuevo vamos a ver como se puede cambiar el SRS de la capa.

```
# UPDATE gis.farmacias SET geom = ST_SetSRID(geom, 32616);

ERROR: Geometry SRID (32616) does not match column SRID (4326)

***** Error *****
```

ERROR: Geometry SRID (32616) does not match column SRID (4326) SQL state: 22023

Esto nos indica que no es posible almacenar en la tabla datos con el SRS 32616 ya que solo permite SRS 4326.

Podremos modificar el tipo de dato si tiene restricciones que comprueben el SRS mediante la eliminación de la restricción y la recreación de la misma con el nuevo SRS. PostGIS nos ofrece la función `updateGeometrySRID` que realizaría de manera automática las operaciones anteriores:

```
# SELECT ST_AsEWKT(way) FROM gis.farmacias;

"SRID=4326;POINT(-87.2238485 14.0978939) "

# SELECT updateGeometrySRID('gis','farmacias','way', 32616);

# SELECT ST_AsEWKT(way) FROM gis.farmacias;

"SRID=32616;POINT(-87.2238485 14.0978939) "
```

Podremos comprobar fácilmente que se ha realizado el cambio del sistema de coordenadas, pero también veremos que el valor de las mismas no ha cambiado.

Reproyección de la capa

Para realizar la reproyección de una capa disponemos de la función `ST_Transform`. Si queremos reproyectar una capa podremos hacerlo fácilmente transformando las coordenadas y creando con ellas una nueva tabla:

```
# SELECT * INTO gis.farmacias_reproyectadas
FROM (
    SELECT osm_id,ST_Transform(way, 32616) FROM gis.farmacias
) as foo
```

Si lo que queremos es reproyectar la misma capa:

```
# ALTER TABLE gis.farmacias ALTER COLUMN way TYPE geometry;
# UPDATE gis.farmacias SET way = ST_Transform(way, 32616);
# ALTER TABLE gis.farmacias ALTER COLUMN way TYPE geometry('POINT', 32616);
```

Simplificación de geometrías

`ST_RemoveRepeatedPoints(geom)`, elimina los puntos repetidos de los vértices. En el modelo usado por PostGIS se permiten vértices con puntos repetidos.

```
# SELECT ST_AsText(ST_RemoveRepeatedPoints('LINESTRING(0 0, 1 0, 1 0, 3 0, 3 0, 6 0, ↵
↵ 8 0)'));

"LINESTRING(0 0, 1 0, 3 0, 6 0, 8 0) "
```

`ST_SnapToGrid(geom)`, realiza un redondeo de las coordenadas de los vértices de una geometría. Además el uso de esta función elimina los vértices que se encuentran en una misma celda. Para realizar el proceso se utiliza una rejilla que se pasa a la función como parte de los parámetros:

```
# SELECT st_isvalid(geom) FROM gis.honduras;

# SELECT * INTO gis.honduras_snap
FROM (select st_snaptogrid(geom, 100)::geometry('MULTIPOLYGON', 32616) as geom from ↵
↵ gis.honduras) as foo
```



```
# SELECT sum(st_npoints(geom)) FROM gis.honduras_snap  
# SELECT st_isvalid(geom) FROM gis.honduras_snap;
```

Otras herramientas de simplificación de geometrías son ST_Simplify y ST_SimplifyPreserveTopology

GeoNetwork como catálogo de Metadatos

Nota:	Fecha	Autores
	1 Diciembre 2012	■ Micho García (micho.garcia@geomati.co)

©2012 Micho García

Excepto donde quede reflejado de otra manera, la presente documentación se halla bajo licencia : Creative Commons (Creative Commons - Attribution - Share Alike: <http://creativecommons.org/licenses/by-sa/3.0/deed.es>)

Introducción a los metadatos

¿Qué son los metadatos?

Los metadatos ayudan a organizar y mantener la inversión de una organización en los datos y proporciona además esa información en forma de catálogo. El desarrollo coordinado de los metadatos evita duplicar esfuerzos asegurando que la organización es consciente de la existencia del conjuntos de datos.[1]

Los usuarios pueden localizar todos los datos geoespaciales y los importantes asociados a un área de interés.

Los metadatos aparecen en muchos más sitios de los que se piensa, por ejemplo:

Description De Producto

Problema
El cliente desea que tenga su información geográfica almacenada en un formato que sea compatible con los estándares de la industria, para poder compartir la información con otros usuarios y poder acceder a ella desde cualquier sistema de información geográfica. El cliente desea que la información sea compatible con los estándares de la industria, para poder compartir la información con otros usuarios y poder acceder a ella desde cualquier sistema de información geográfica.

Propuesta de Solución
La solución propuesta es el uso de un formato de archivo que sea compatible con los estándares de la industria, para poder compartir la información con otros usuarios y poder acceder a ella desde cualquier sistema de información geográfica.

Beneficios
El cliente podrá compartir su información geográfica con otros usuarios y poder acceder a ella desde cualquier sistema de información geográfica.

Características
La solución propuesta es el uso de un formato de archivo que sea compatible con los estándares de la industria, para poder compartir la información con otros usuarios y poder acceder a ella desde cualquier sistema de información geográfica.

Requisitos
El cliente debe tener un sistema de información geográfica que sea compatible con los estándares de la industria, para poder compartir la información con otros usuarios y poder acceder a ella desde cualquier sistema de información geográfica.

Entregables
El cliente recibirá un archivo de información geográfica que sea compatible con los estándares de la industria, para poder compartir la información con otros usuarios y poder acceder a ella desde cualquier sistema de información geográfica.

Estos no son mas que información (datos) sobre la información (mas datos).

Cuando realizamos un fotografía con nuestro terminal, se almacenan en dicha fotografía cierta información como el modelo de la cámara del terminal, la posición donde se tomó la foto (si se dispone de GPS), valores de exposición... cuanto más información se disponga sobre la fotografía, tendremos mejor disposición a la hora de buscar en nuestra colección de fotografías. Por ejemplo, podremos buscar las fotografías que hicimos en determinado lugar o cerca de él, las que se hicieron con una u otra cámara... Famosos portales de fotografía hacen un uso intensivo de los metadatos a la hora de catalogar esa información.

Esto es posible porque los metadatos de las fotografías se engloban dentro de una especificación Exif que han adoptado muchos de los fabricantes de cámaras y que están implementados en los tipos de archivos ampliamente extendidos como JPEG o TIFF.[2]

La función de los metadatos sobre la información espacial es la misma, la de aportar más información sobre esta información espacial, por definirlo de una manera más formal: “los metadatos son “datos sobre los datos”. Describen el contenido, la calidad, la condición y otras características de los datos. Ayudan a una persona o sistema inteligente a localizar y entender los datos espaciales disponibles.”[3]

Para que los metadatos sean facilmente explotables deben de tener un orden, deben de poder ser entendidos por cualquier persona o sistema sin necesidad de ser explicados. Si en el caso de las fotografías, la posición, por ejemplo, la guardásemos en un campo nombrado de diferente manera en cada dispositivo, la información del metadato sería difícilmente recuperable y explotable, por eso, en el caso de las fotografías los fabricantes asumen la especificación que sería similar a un estandar.

En el caso de los metadatos geográficos, se han definido diferentes estándares para el manejo de los metadatos. Algunos se describen a continuación.

Definición de algunas de las normas

ISO19115:2033 - Geographic Information Metadata

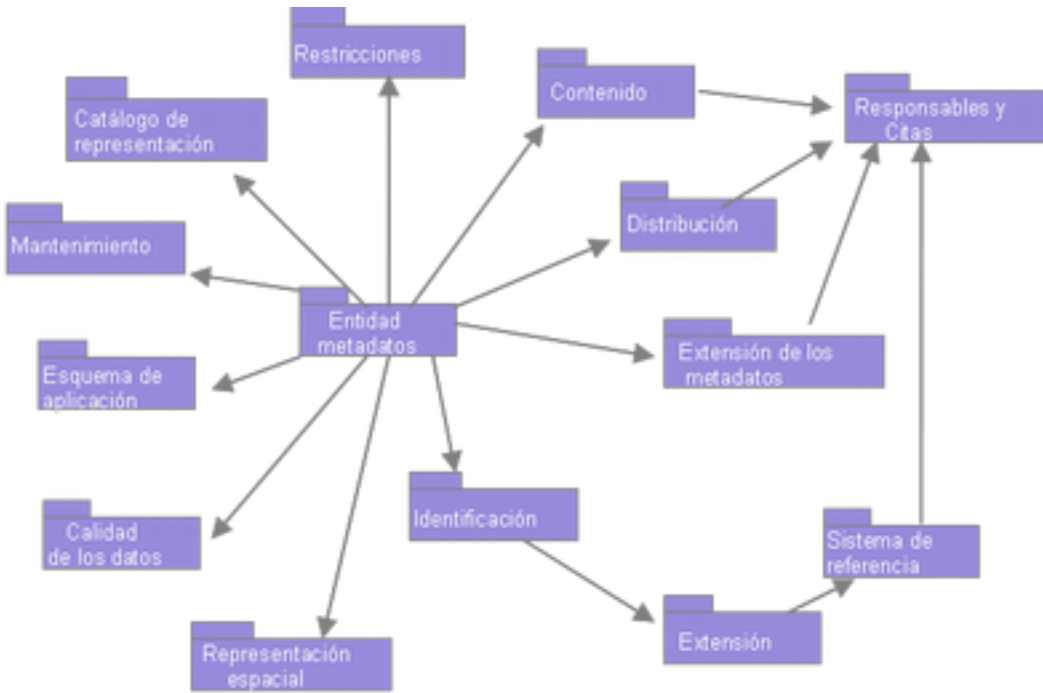
“La Norma Internacional ISO 19115:2003 - Geographic Information - Metadata, define el modelo requerido para describir información geográfica y servicios. Proporciona información sobre la identificación, la extensión, la calidad, el modelo espacial y temporal, la referencia espacial y la distribución de los datos geográficos digitales.

La Norma Internacional define:

- Secciones de metadatos obligatorios y condicionales, entidades de metadatos y elementos de metadatos;
- El conjunto mínimo de metadatos requeridos para soportar todo el rango de aplicaciones de metadatos (descubrimiento de datos, determinación de la idoneidad de unos para un uso, acceso a los datos, transferencia de datos y utilización de datos digitales);

- Elementos de metadatos opcionales, para permitir una descripción normalizada más amplia de los datos geográficos, si así se requiere;
- Un método para crear extensiones de metadatos para adaptarse a necesidades especializadas.”[4]

Se compone a su vez de diferentes elementos:



En esta imagen se ve como se relacionan los diferentes elementos que componen la norma con el Elemento Metadato que es el núcleo de la misma. En su totalidad la norma ISO19115 dispone de mas de 400 campos que pueden ser alimentados con información.

A pesar de la extensión de la norma, esta dispone de un núcleo - **Core** de metadatos de obligado cumplimiento que todo perfil que se base en esta norma debe incorporar de manera obligatoria.

ISO19139 - Geographic Information- Metadata – XML schema implementation

“La norma ISO 19115 proporciona una estructura para describir información geográfica mediante elementos de metadatos y establece una terminología común para los mismos pero no desarrolla como poder llevar a cabo su implementación.”[5]

Lo que plantea esta norma es una implementación en XML de la ISO19115:2003 basandose en el uso de los XML-Schema[6], documentos que definen la estructura que debe tener el documento XML. Una de las definiciones que se disponen en el documento es el uso de espacios de nombre permitiendo de esa manera el uso de etiquetas con el mismo nombre pero que puedan tener significados diferentes en contextos diferentes.

Ejemplo de un metadato implementado con la norma ISO19139

ISO15836:2003 - Information and Documentation- The Dublin Core Metadata Element Set

Esta set de metadatos, creado en 1995, promueve la difusión de estándares/normas de metadatos interoperables y el desarrollo de vocabularios de metadatos especializados que permitan la construcción de sistemas de búsqueda de información más inteligentes. Se trata de un estandar para la descripción de todo tipo de recursos de manera

independiente de su formato, área de especialización y está desarrollado para que pueda ser utilizado por sistema de búsquedas inteligentes.

Su simplicidad, dispone de quince descriptores básicos, hace que esté siendo adoptada por muchas organizaciones dedicadas a la IG. Mediante el uso de este estándar se posibilita la incorporación más rápida de datos a un catálogo o la posibilidad de, mezclandolos con las ISO19115 utilizarlos como una primera aproximación al metadato más desarrollado de la ISO19115.

Este estandar está propuesto por la OGC como modelo básico de búsqueda y presentación de documentos. Asimismo Dublin Core se puede utilizar combinandolo con RDF, el “lenguaje” de la web semántica.

CSW el estandar OGC para servir los metadatos

CSW, o Catalog Service for the Web[7], se trata de un estandar de la OGC para la publicación, descubrimiento y explotación de los metadatos. Lo que hace es exponer en la web un catálogo de metadatos. Está desarrollado por la OGC y su última versión en el momento de escribir este documento era la 2.0.2. Generalmente los estándares de servicios de la OGC definen una serie de operaciones y resultados que se deben cumplir para incorporar el estandar. En el caso del CSW se definirán las siguientes operaciones:

GetCapabilities

Que, al igual que en el resto de servicios de la OGC donde se implementa esta operación (W*S), nos devolverá una descripción de las propiedades de las operaciones de las que dispone el servidor.

DescribeRecord

Devuelve información que describe los registros basada en el model de la información soportado por el catálogo sobre el que se ejecuta la consulta. Describe como se muestran los registros, como están formados.

GetRecords

Devolverá los registros que cumplan las condiciones de filtrado que le podemos enviar. En esta operación es donde tenemos disponible la capacidad para obtener los metadatos que deseamos. La infinita capacidad de filtrado que nos permite, hace que sea muy potente a la hora de explotar el servicio de catálogo.

GetRecordById

Devuelve el registro que tenga el ID que se le envíe en la consulta.

GeoNetwork implementa la última versión de CSW y permite su configuración a través del panel de administración y su testeo desde su CSW ISO Profile Test, herramienta muy interesante para el uso y entendimiento del estándar.

GeoNetwork

Instalación y configuración

En la versión 2.10 de Geonetwork se ha simplificado mucho el proceso de instalación. Para disponer de **GeoNetwork** en nuestro servidor, descargaremos de la web de **GeoNetwork** el instalador correspondiente al sistema operativo

que estemos usando. En el caso de Linux, descargaremos el archivo .jar. Para ejecutar este archivo deberemos tener instalada la versión 6 de Java. Esto lo podremos comprobar ejecutando en un terminal:

```
$ java -version
```

que nos mostrará algo del estilo:

```
java version "1.6.0_27"
OpenJDK Runtime Environment (IcedTea6 1.12.6) (6b27-1.12.6-1ubuntu0.12.04.4)
OpenJDK Client VM (build 20.0-b12, mixed mode, sharing)
```

Dependiendo de la plataforma sobre la que estemos trabajando. En caso de no disponer de la versión 6 de Java se recomienda buscar en la web el proceso de instalación de la misma.

Una vez instalada correctamente la versión de Java, procederemos a instalar el servidor **GeoNetwork**, para ello ejecutaremos:

```
$ java -jar geonetwork-install-2.X.X-X.jar
```

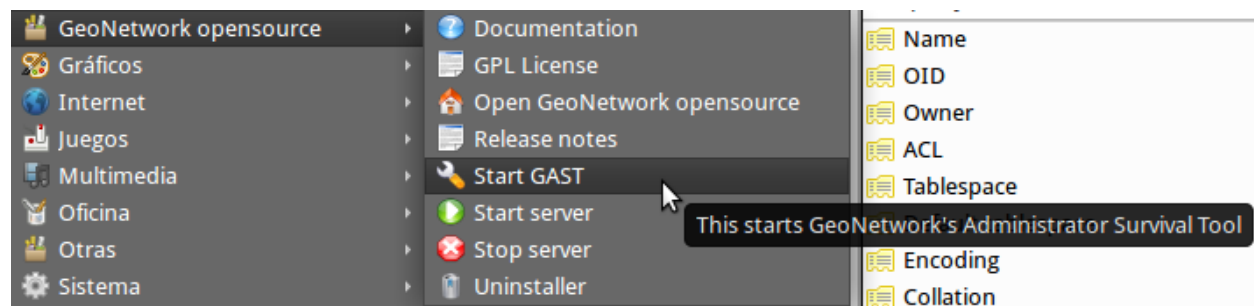
lo que nos arrancará un instalador que nos guiará durante todo el proceso. Llegados un punto en la instalación, nos pedirá si deseamos instalar GAST. GAST, acrónimo de Geonetwork's Administration Survival Tool[8]. GAST es una herramienta de ayuda a la administración de **GeoNetwork**. Actualmente la mayoría de las funcionalidades están migradas al panel de administración de **GeoNetwork**, pero sigue manteniendo un interfaz de usuario para una configuración básica de la conexión JDBC con la base de datos.

Arquitectura de GeoNetwork

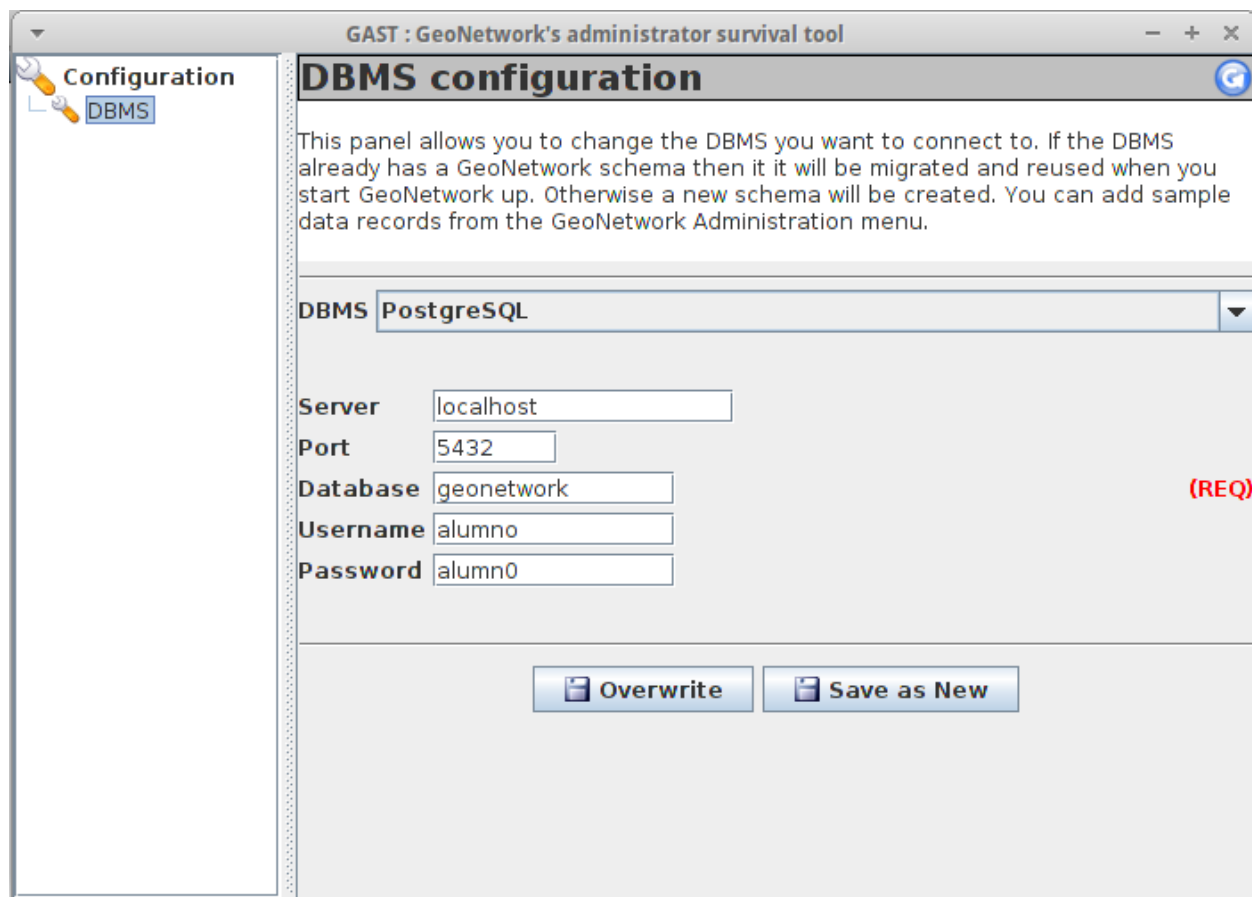
GeoNetwork se apoya en diferentes tipos de software para llevar a cabo su tarea. En líneas generales se trata de una base de datos que permite el almacenamiento de los metadatos, un motor de búsqueda de texto en esos metadatos y una herramienta que adapte esos metadatos a las diferentes plantillas. Describiremos una a una las tecnologías:

Base de datos

Por defecto **GeoNetwork** trae instalada una base de datos H2[9]. Se trata de una base de datos que puede funcionar embebida. En el caso de que tengamos a nuestra disposición una base de datos diferente, podremos configurar nuestro **GeoNetwork** para que utilice nuestra base de datos en vez de la base de datos H2 que trae por defecto. Para cambiar esta base de datos podremos utilizar la herramienta GAST.



Desde el menu de **GeoNetwork** arrancaremos GAST. En el listado de DBMS, seleccionaremos el que nos interese, en nuestro caso **PostgreSQL**, y añadiremos los datos necesarios para generar la cadena de conexión. Una vez hecho esto, lo guardamos en caso de que sea nuevo, o lo sobrescribimos y reiniciamos el servidor.



Si vamos a la base de datos, veremos que ha generado el modelo de datos necesario para el manejo de **GeoNetwork**.

Contenedor de servlets y servidor web

GeoNetwork dispone de una parte web a través de la se publican los servicios y se accede a la funcionalidad y visualización de los metadatos. Para este fin **GeoNetwork** dispone de Jetty[11], un contenedor de servlets al estilo de Apache Tomcat, que permite funcionar embebido en el proyecto. Jetty ofrece un servicio de calidad[12] por lo que se puede mantener. En caso de que se desee el uso de otro contenedor de servlets, **GeoNetwork** es perfectamente funcional en Apache Tomcat al tratarse de una aplicación relizada en Java.

La importancia del contenedor de servlets, es que **GeoNetwork** ha desarrollado toda su funcionalidad mediante servicios web. De esta manera, cuando estamos insertando un metadato, estamos realizando una llamada al servicio:

```
Url:
http://localhost:8080/geonetwork/srv/en/metadata.insert

Mime-type:
application/xml

Post request:
<?xml version="1.0" encoding="UTF-8"?>
<request>
  <group>2</group>
  <category>_none_</category>
  <styleSheet>_none_</styleSheet>
  <data><![CDATA[
```



```

<gmd:MD_Metadata xmlns:gmd="http://www.isotc211.org/2005/gmd"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
...
    </gmd:DQ_DataQuality>
    </gmd:dataQualityInfo>
</gmd:MD_Metadata>]]>
</data>
</request>

```

Hay una descripción de los servicios de **GeoNetwork** en [13]

Apache Lucene

La funcionalidad de **GeoNetwork** al final, es la de buscar en cantidades ingentes de información tipo texto. Keywords, titles... se tratan de textos que están almacenados en la base de datos y que deben poder ser buscados desde **GeoNetwork**. Para realizar esto hace uso de Apache Lucene[14]. Apache Lucene es un buscador de texto.

Podremos configurar la Lucene con los campos en los que queramos que nos realice la búsqueda. Para ello deberemos modificar los campos en el archivo *config-lucene.xml* y a su vez las plantillas sobre las que queremos añadir los campos. Con la herramienta Luke[15] podremos acceder al índice de Lucene para manejarlo.

Jeeves

Jeeves se basa en transformaciones XSLT que permiten un desarrollo rápido y sencillo (a la vez que potente) de interfaces tanto para un usuario como para máquinas (XML).[20]

Z39.50

“Z39.50 es un protocolo cliente-servidor dirigido a facilitar la búsqueda y recuperación de información en distintos sistemas a través de una misma interfaz. Su aplicación en el mundo de las bibliotecas y de los centros de documentación permite la consulta de recursos distribuidos en distintas bases de datos, desde un mismo punto de acceso.” [18]

“**GeoNetwork** usando el protocolo Z39.50 puede acceder a catálogos remotos y hace que sus datos estén disponibles para otros servicios de catálogo”. [19]

Entendiendo XML, XML - Schema y XSLT

XML

“La tecnología XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y que esas partes se componen a su vez de otras partes. Entonces se tiene un árbol de trozos de información.”[21]

Un ejemplo de XML sería:

```

<?xml version="1.0" encoding="UTF-8" ?>

<Edit_Mensaje>
    <Mensaje>
        <Remitente>
            <Nombre>Nombre del remitente</Nombre>
            <Mail> Correo del remitente </Mail>
        </Remitente>
    </Mensaje>
</Edit_Mensaje>

```

```

        <Destinatario>
            <Nombre>Nombre del destinatario</Nombre>
            <Mail>Correo del destinatario</Mail>
        </Destinatario>
        <Texto>
            <Asunto>
                Este es mi documento con una estructura muy sencilla
                no contiene atributos ni entidades...
            </Asunto>
            <Parrafo>
                Este es mi documento con una estructura muy sencilla
                no contiene atributos ni entidades...
            </Parrafo>
        </Texto>
    </Mensaje>
</Edit_Mensaje>

```

“Los documentos denominados como «bien formados» (del inglés well formed) son aquellos que cumplen con todas las definiciones básicas de formato y pueden, por lo tanto, analizarse correctamente por cualquier analizador sintáctico (parser) que cumpla con la norma. Se separa esto del concepto de validez que se explica más adelante.

- Los documentos han de seguir una estructura estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente incluida en otra, es decir, las etiquetas deben estar correctamente anidadas. Los elementos con contenido deben estar correctamente cerrados.
- Los documentos XML sólo permiten un elemento raíz del que todos los demás sean parte, es decir, solo pueden tener un elemento inicial.
- Los valores atributos en XML siempre deben estar encerrados entre comillas simples o dobles.
- El XML es sensible a mayúsculas y minúsculas. Existe un conjunto de caracteres llamados espacios en blanco (espacios, tabuladores, retornos de carro, saltos de línea) que los procesadores XML tratan de forma diferente en el marcado XML.
- Es necesario asignar nombres a las estructuras, tipos de elementos, entidades, elementos particulares, etc. En XML los nombres tienen alguna característica en común.
- Las construcciones como etiquetas, referencias de entidad y declaraciones se denominan marcas; son partes del documento que el procesador XML espera entender. El resto del documento entre marcas son los datos «entendibles» por las personas.”

XML - Schema

“XML Schema es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML. Se consigue así una percepción del tipo de documento con un nivel alto de abstracción.” [23]

Disponemos del siguiente Schema:

```

<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
    <xs:element name="Employee_Info" type="EmployeeInfoType" />
    <xs:complexType name="EmployeeInfoType">
        <xs:sequence>
            <xs:element ref="Employee" minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>

```

```

<xs:element name="Employee" type="EmployeeType" />
<xs:complexType name="EmployeeType">
  <xs:sequence >
    <xs:element ref="Name" />
    <xs:element ref="Department" />
    <xs:element ref="Telephone" />
    <xs:element ref="Email" />
  </xs:sequence>
  <xs:attribute name="Employee_Number" type="xs:int" use="required"/>
</xs:complexType>
<xs:element name="Name" type="xs:string" />
<xs:element name="Department" type="xs:string" />
<xs:element name="Telephone" type="xs:string" />
<xs:element name="Email" type="xs:string" />
</xs:schema>

```

Y queremos definir nuestro XML en función de ese Schema:

```

<?xml version="1.0"?>
<Employee_Info
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="employee.xs">
  <Employee Employee_Number="105">
    <Name>Masashi Okamura</Name>
    <Department>Design Department</Department>
    <Telephone>03-1452-4567</Telephone>
    <Email>okamura@xmltr.co.jp</Email>
  </Employee>
  <Employee Employee_Number="109">
    <Name>Aiko Tanaka</Name>
    <Department>Sales Department</Department>
    <Telephone>03-6459-98764</Telephone>
    <Email>tanaka@xmltr.co.jp</Email>
  </Employee>
</Employee_Info>

```

Podremos probar que se trata de un documento que se adapta al XML - Schema definido en cualquier validador de XML - Schema online como este:

<http://www.utilities-online.info/xsdvalidation>

El Schema marca las reglas que debe cumplir un determinado documento XML.

XSLT

“XSLT o Transformaciones XSL es un estándar de la organización W3C que presenta una forma de transformar documentos XML en otros e incluso a formatos que no son XML. Las hojas de estilo XSLT - aunque el término de hojas de estilo no se aplica sobre la función directa del XSLT - realizan la transformación del documento utilizando una o varias reglas de plantilla. Estas reglas de plantilla unidas al documento fuente a transformar alimentan un procesador de XSLT, el que realiza las transformaciones deseadas poniendo el resultado en un archivo de salida, o, como en el caso de una página web, las hace directamente en un dispositivo de presentación tal como el monitor del usuario.”[22]

Creación y publicación de metadatos

Para la creación de metadatos lo primero es tener los permisos necesarios para poder insertar registros. Como mínimo necesitamos el perfil de `Editor` para poder incluir datos en el sistema.

Después tendremos que tener cargadas nuestras plantillas del perfil que estemos usando sobre las que se apoyarán los metadatos. Para cargar las plantillas de metadatos debemos desde Administración -> Add templates y seleccionaremos las plantillas que deseamos cargar.

Una vez que tenemos las plantillas cargadas, desde la pestaña de Administración -> New Metadata, seleccionamos la plantilla sobre la que queremos trabajar y el grupo de usuarios para el que estará disponible.

Lo siguiente es utilizar el editor para rellenar los valores de los metadatos.

Importación de metadatos

Otra manera de incluir metadatos en nuestro sistema es mediante la importación de los mismos. Para ello utilizaremos la herramienta de importación de metadatos de **GeoNetwork**. Primero obtendremos un metadato que deseemos importar. Podemos generarlo desde una herramienta de creación de metadatos como CatMEdit[16] o podemos descargar algún metadato de un catálogo ya existente, como este[17].

Desde la página de Administración` -> ``Metadata Insert rellenaremos los campos e insertaremos para tener incluido en nuestro catálogo el metadato.

También podremos importar metadatos desde un directorio simplemente utilizando la opción Batch Import. Para ello desde la pestaña de administración, seleccionamos el directorio donde se almacenan los metadatos y ejecutamos. Debemos definir la plantilla que vamos a utilizar a la hora de cargarlos.

La importación se puede realizar a través de archivos MEF (Metadata Exchange File), archivos creados por **GeoNetwork** que adoptan un determinado formato y que permiten la exportación e importación masiva de metadatos permitiendo realizar backups a través de ellos. Para una definición completa de MEF se recomienda visitar la web de **GeoNetwork** [24]

Consumo de metadatos de diferentes nodos

Mediante **GeoNetwork** tenemos la posibilidad de consumir metadatos de diferentes nodos. Esta tal vez es una de las características más potentes de **GeoNetwork** que nos permite compartir los metadatos de otros catálogos desde el nuestro. Para ello, **GeoNetwork** dispone de un proceso de Harvesting, recolección, mediante el cual programamos las rutas en las que se encuentran los otros catálogos, servicios, instancias de Thredds... desde las que queremos obtener los metadatos y definiremos cuando queremos que nuestro servidor haga la recolección de datos.

Para activar este proceso debemos ir a Administración -> Harvesting Management y definir un nuevo nodo desde el que obtener más información.

Configuración del servidor CSW

CSW es el estandar que utiliza **GeoNetwork** para explotar los metadatos. CSW viene activado por defecto en **GeoNetwork**. Para configurarlo y realizar pruebas sobre el servidor CSW desde la pestaña de Administración -> CSW Server Configuration y ahí modificaremos los parámetros que nos interesen. Principalmente definiremos los metadatos de nuestro servicio CSW. Desde esa misma pestaña, podemos acceder al CSW ISO Profile Test desde donde podremos probar las diferentes operaciones de las que disponemos en el estandar CSW.

CAPÍTULO 15

Indices and tables

- `genindex`
- `modindex`
- `search`