Universidad del Azuay

Facultad de Ciencias de la Administración

Escuela de Ingeniería de Sistemas

Publicación de mapas temáticos de la cuenca del río Paute, a través de Openlayers

Trabajo de graduación previo a la obtención del título de Ingeniero de Sistemas

Autor: Diego Patricio Farfán Panamá

Director: Ing. Chester Sellers

Cuenca, Ecuador

2009

Dedicatoria

Quiero dedicar este trabajo a mi esposa y mi hija, quienes son la fuente de inspiración en el día a día. A mis padres y hermanos quienes me han apoyado durante todo este proceso académico. Y a todas las personas que de una forma u otra me han dado su ayuda.

Agradecimientos

Agradezco principalmente a Dios, quien me ha dado la capacidad de concluir esta carrera de manera satisfactoria. A mi esposa, quien me ha apoyado en las largas horas de estudio que requerían esta especialidad. A mis padres por el apoyo incondicional, a mis amigos, profesores y compañeros de trabajo.

Índice de Contenidos

Índice de Ilustraciones y Tablas

Índice de Anexos

Resumen

El desarrollo del siguiente trabajo se centrará en publicar la información cartográfica temática entregada por la Universidad del Azuay, utilizando la interfaz y funcionalidad de código abierto que ofrece *Openlayers*.

Se hará referencia a información externa de las tecnologías en lo que se refiere a lenguajes de programación web. Luego se dará una breve introducción a las librerías *Openlayers* así como una descripción de su funcionalidad, se describirá también *MapServer* que es el medio lógico para publicar mapas. Se configurará y probará la interfaz de *Openlayers* con los estándares de estilos del portal *web* de la Universidad del Azuay, con el objetivo de hacer más versátil y amigable la visualización de los mismos. Finalmente se documentará la información relevante que sirva como referencia para poder agregar nueva información cartográfica digital cuando así se lo requiera.

Abstract

The development of this project will center on the publication of thematic cartographic Information submitted by the Universidad del Azuay, using the interface and functionality of the open code offered by Openlayers.

Reference to external technological information in reference to Web programming languages will be made. Later, a brief introduction to the Openlayers bookshelves will be made as well as a description of its functionality. Mapserver will also be described as it is the logical medium for map publishing. The Openlayers interface will be configured and tested to the style standards of the Web portal of the Universidad del Azuay with the aim of making the visualization of the maps more versatile and user-friendly. Finally, the relevant information will be documented as a reference to facilitate the addition of new digital cartographic information as necessary.

INTRODUCCION

Con la evolución dramática que está sucediendo sobre de las Tecnologías de Información y Comunicaciones (TIC) se ha facilitado en gran medida el acceso a la información en muchos lugares en donde, por el subdesarrollo, la brecha digital era muy amplia. A medida que la sociedad ha tenido acceso al conocimiento, así mismo se ha incrementado la necesidad de brindar más información. Por ello ahora se nos hace más fácil conocer nuestra ubicación geográfica con un par de clics.

Dentro este ámbito y gracias a la información geográfica que dispone la Universidad del Azuay, el presente trabajo tiene como finalidad poner a disposición de mundo dicha información, pero esta vez a través de una interfaz diferente que permite una manipulación más cómoda y rápida de los mapas, utilizando el Internet.

CAPITULO 1

LENGUAJES DE PROGRAMACIÓN *WEB*

1.1. Introducción

En la actualidad existen una variedad de lenguajes que se utilizan para publicar o manipular información en la *web*, dichos lenguajes han ido apareciendo debido a las diferentes tendencias y necesidades de las plataformas, en el presente capítulo se pretende mostrar una breve descripción de los más usados así como las ventajas y desventajas que se generan de su uso.

Desde los inicios de Internet, surgieron diferentes demandas por los usuarios y se dieron soluciones mediante lenguajes estáticos. A medida que pasó el tiempo, las tecnologías fueron desarrollándose y se presentaron nuevos problemas a los cuales se tuvo que dar solución. Esto dio lugar al desarrollo de lenguajes de programación dinámicos para la *web*, que permitieran interactuar con los usuarios y utilizaran sistemas de **Bases de Datos**.

1.2. Lenguaje HTML

Desde el surgimiento de internet se han publicado sitios *web* gracias al lenguaje HTML. Es un lenguaje estático para el desarrollo de sitios *web* (acrónimo en inglés de *HyperText Markup Language*, en español Lenguaje de Marcas Hipertextuales). Desarrollado por el *World Wide Web Consortium* (W3C). Los archivos pueden tener las extensiones (htm, html).

Sintaxis:

```
<html>                          (Inicio del documento HTML)
        <head>                  (Cabecera)
        </head>
        <body>                  (Cuerpo)
        </body>
</html>
        <b>  </b>               (Negrita)
        <p>  </p>               (Definir párrafo)
```

```
<etiqueta>          (Apertura de la etiqueta)
</etiqueta>         (Cierre de la etiqueta)
```

Ventajas:

- Lenguaje sencillo de escribir
- Texto presentado de forma estructurada y agradable.
- No necesita de grandes conocimientos cuando se cuenta con un editor de páginas *web* o *WYSIWYG*[1].
- Los archivos tienen tamaño reducido al ser internamente de texto.
- Despliegue rápido en navegadores *web*.
- Es admitido por todos los navegadores *web*.

Desventajas:

- La información presentada es estática.
- La interpretación de lenguaje en cada navegador puede ser diferente.
- En archivos largos, o los creados por herramientas, pueden generar muchas etiquetas "basura" y dificultan la corrección.
- El diseño es más lento, si se lo hace escribiendo el código línea por línea.
- Las etiquetas son muy limitadas.

1.3. Lenguaje Javascript

Javascript es un lenguaje interpretado, es decir, no requiere compilación. Fue creado por Brendan Eich en la empresa *Netscape Communications*. Utilizado principalmente en páginas *web*. Es similar a Java, aunque no es un lenguaje orientado a objetos (no dispone de herencias). La mayoría de los navegadores interpretan Javascript.

Ventajas:

- Lenguaje de scripting seguro y fiable.
- Los *script* tienen capacidades limitadas, por razones de seguridad.
- El código Javascript se ejecuta en el cliente.

[1] WYSIWYG: "*What you see is what you get*". Característica que tienen algunos editores de texto, por la cual lo que se ve en la pantalla es el resultado final que se obtiene en la impresión o en el visionado final.

Desventajas:

- Código visible por cualquier usuario.
- El código debe descargarse completamente.
- Puede poner en riesgo la seguridad del sitio, con el actual problema llamado XSS (significa en inglés *Cross Site Scripting* renombrado a XSS).

1.4. Lenguaje PHP

Es un lenguaje de programación utilizado para la creación de sitios *web*. PHP es un acrónimo recursivo que significa "*PHP Hypertext Pre-processor*". Surgió en 1995, desarrollado por *PHP Group*.

PHP es un lenguaje de script interpretado en el lado del servidor utilizado para la generación de páginas *web* dinámicas, embebidas en páginas HTML y ejecutadas en el servidor. PHP no necesita ser compilado para ejecutarse. La mayor parte de su sintaxis ha sido tomada de C, Java y Perl con algunas características específicas. Los archivos cuentan con la extensión (php).

Sintaxis:

La sintaxis utilizada para incorporar código PHP es la siguiente:

```
<?php
$mensaje = "Hola";
echo $mensaje;
?>
```

Ventajas:

- Es un lenguaje fácil de aprender.
- Se caracteriza por ser un lenguaje muy rápido.
- Soporta en cierta medida la orientación a objetos. Clases y herencia.
- Es un lenguaje multiplataforma: Linux, Windows, entre otros.
- Capacidad de conexión con la mayoría de los manejadores de base de datos: MySQL, PostgreSQL, Oracle, MS SQL Server, entre otras.

- Capacidad de expandir su potencial utilizando módulos.
- Posee documentación en su página oficial la cual incluye descripción y ejemplos de cada una de sus funciones.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- No requiere definición de tipos de variables ni manejo detallado del bajo nivel.

Desventajas:

- Se necesita instalar un servidor *web*.
- Todo el trabajo lo realiza el servidor y no delega ningún proceso al cliente. Por tanto puede ser más ineficiente a medida que las solicitudes aumenten de número.
- La legibilidad del código puede verse afectada al mezclar sentencias HTML y PHP.
- La programación orientada a objetos es aún muy deficiente para aplicaciones grandes.
- Dificulta la modularidad.
- Dificulta la organización por capas de la aplicación.

Seguridad:

PHP es un poderoso lenguaje e intérprete, ya sea incluido como parte de un servidor *web* en forma de módulo o ejecutado como un binario CGI[2] separado, es capaz de acceder a archivos, ejecutar comandos y abrir conexiones de red en el servidor. Estas propiedades hacen que cualquier cosa que sea ejecutada en un servidor *web* sea insegura por naturaleza.

PHP está diseñado específicamente para ser un lenguaje más seguro para escribir programas CGI que Perl o C, con la selección correcta de opciones de configuración en tiempos de compilación y ejecución, esto siguiendo algunas prácticas correctas de programación.

[2] CGI: *Common Gateway Interface.* Tecnología de la WWW que permite a un cliente (navegador web) solicitar datos de un programa ejecutado en un servidor web.

1.5. Lenguaje ASP

Es una tecnología del lado de servidor desarrollada por *Microsoft* para el desarrollo de sitio *web* dinámicos. ASP significa en inglés (*Active Server Pages*), fue liberado por *Microsoft* en 1996. Es necesario tener instalado *Internet Information Server* (IIS).

ASP no necesita ser compilado para ejecutarse. Existen varios lenguajes que se pueden utilizar para crear páginas ASP. El más utilizado es VBScript, nativo de Microsoft. ASP se puede hacer también en Perl and Jscript (no JavaScript). El código ASP puede ser insertado junto con el código HTML. Los archivos cuentan con la extensión (asp).

Ventajas:

- Usa *Visual Basic Script*, siendo fácil para los usuarios.
- Comunicación óptima con *SQL Server*.
- Soporta el lenguaje *JScript* (*Javascript* de *Microsoft*).

Desventajas:

- Puede generarse código desorganizado.
- Se necesita escribir mucho código para realizar funciones sencillas.
- Tecnología propietaria.
- Hospedaje de sitios *web* costosos.

1.6. Lenguaje ASP.NET

Este es un lenguaje comercializado por *Microsoft*, y usado por programadores para desarrollar entre otras funciones, sitios *web*. ASP.NET es el sucesor de la tecnología ASP, fue lanzada al mercado mediante una estrategia de mercado denominada .NET. El ASP.NET fue desarrollado para resolver las limitantes que brindaba tu antecesor ASP. Creado para desarrollar *web* sencillas o grandes aplicaciones. Para el desarrollo de ASP.NET se puede utilizar C#, VB.NET o J#. Los archivos cuentan con la extensión (aspx). Para el funcionamiento de las páginas se necesita tener instalado *IIS* con el *Framework .Net*.

Ventajas:

- Completamente orientado a objetos.
- Controles de usuario y personalizados.
- División entre la capa de aplicación o diseño y el código.
- Facilita el mantenimiento de grandes aplicaciones.
- Incremento de velocidad de respuesta del servidor.
- Mayor velocidad.
- Mayor seguridad.

Desventajas:

- Mayor consumo de recursos.

1.7. Lenguaje JSP

Es un lenguaje para la creación de sitios *web* dinámicos, acrónimo de *Java Server Pages*. Está orientado a desarrollar páginas *web* en Java. JSP es un lenguaje multiplataforma. Creado para ejecutarse del lado del servidor.

JSP fue desarrollado por *Sun Microsystems*. Comparte ventajas similares a las de ASP.NET, está desarrollado para la creación de aplicaciones *web* potentes. Posee un motor de páginas basado en los *servlets* de *Java*. Para su funcionamiento se necesita tener instalado un servidor *Tomcat*.

Características:

- Código separado de la lógica del programa.
- Las páginas son compiladas en la primera petición.
- Permite separar la parte dinámica de la estática en las páginas *web*.
- Los archivos se encuentran con la extensión (jsp).
- El código JSP puede ser incrustado en código HTML.

Elementos de JSP

Los elementos que pueden ser insertados en las páginas JSP son los siguientes:

- Código: se puede incrustar código “Java”.
- Directivas: permite controlar parámetros del *servlet*.

- Acciones: permite alterar el flujo normal de ejecución de una página.

Ventajas:

- Ejecución rápida del *servlets*.
- Crear páginas del lado del servidor.
- Multiplataforma.
- Código bien estructurado.
- Integridad con los módulos de Java.
- La parte dinámica está escrita en Java.

Desventajas:

- Complejidad de aprendizaje.

1.8. Lenguaje Python

Es un lenguaje de programación creado en el año 1990 por *Guido van Rossum*, es el sucesor del lenguaje de programación ABC. Python es comparado habitualmente con Perl.

Los usuarios lo consideran como un lenguaje más limpio para programar. Permite la creación de todo tipo de programas incluyendo los sitios *web*.

Es un lenguaje interpretado de programación multiparadigma, lo cual fuerza a que los programadores adopten por un estilo de programación particular:

- Programación orientada a objetos.
- Programación estructurada.
- Programación funcional.
- Programación orientada a aspectos.

Ejemplo de una clase en Phyton:

```
def dibujar_muneco(opcion):

    if opcion == 1:
        C.create_line(580, 150, 580, 320, width=4, fill="blue")
```

```
C.create_oval(510, 150, 560, 200, width=2, fill='PeachPuff')
```

Ventajas:

- Libre y de fuente abierta.
- Lenguaje de propósito general.
- Gran cantidad de funciones y librerías.
- Sencillo y rápido de programar.
- Multiplataforma.
- Orientado a Objetos.
- Portable.

Desventajas:

- Lentitud por ser un lenguaje interpretado.

1.9. Lenguaje Ruby

Es un lenguaje interpretado de muy alto nivel y orientado a objetos. Desarrollado en el 1993 por el programador japonés *Yukihiro "Matz" Matsumoto*. Su sintaxis está inspirada en Phyton, Perl. Es distribuido bajo licencia de software libre (*Opensource*).

Ruby es un lenguaje dinámico para una programación orientada a objetos rápida y sencilla.

Características:

- Existe diferencia entre mayúsculas y minúsculas.
- Múltiples expresiones por líneas, separadas por punto y coma ";".
- Dispone de manejo de excepciones.
- Ruby puede cargar librerías de extensiones dinámicamente si el (Sistema Operativo) lo permite.
- Portátil.

Ventajas:

- Permite desarrollar soluciones a bajo Costo.

- Software libre.
- Multiplataforma.

CAPITULO 2

POSTGRESQL & POSTGIS

2.1. Introducción

PostgreSQL es un sistema de gestión de base de datos relacional orientada a objetos de software libre, publicado bajo la licencia BSD[1].

Al ser un proyecto de Software Libre, el continuo desarrollo de *PostgreSQL* es operado por una comunidad de desarrolladores y casas comerciales. Dicha comunidad recibe el nombre de *PostgreSQL Global Development Group* (PGDG)

Se le conoce comúnmente por su abreviatura Postgres, que fue su nombre original, pues en sus inicios Postgres manejaba únicamente el lenguaje de bases de datos Ingres, de ahí que sus desarrolladores lo denominaron "*Post*" + "In*gres*" o simplemente Postgres.

A medida que los asiduos de este gestor de bases de datos se fueron incrementando, de igual manera tuvo que mejorarse su código y sus prestaciones para hacerlo más estable. Después de algunas versiones del original Postgres, el proyecto concluyó y el grupo de desarrolladores se disolvió. Sin embargo en 1995, dos estudiantes de la *Universidad de Berkeley* decidieron incorporar a Postgres el soporte para el lenguaje de consultas estructurado o SQL[2] por sus siglas en Ingles, lo bautizaron como Postgres95, que finalmente para dar a conocer la nueva compatibilidad con SQL lo denominaron PostgreSQL.

2.2. Características Principales

Entre las características de PostgreSQL se pueden detallar:

[1] *Berkeley Software Distribution* (en español, Distribución de Software Berkeley) y se utiliza para identificar un sistema operativo derivado del sistema Unix nacido a partir de los aportes realizados a ese sistema por la Universidad de California en Berkeley.
[2] *Structured Query Language*: Lenguaje de consulta estructurado

- Alta concurrencia: Utilizando el sistema MVCC[3], PostgreSQL da la facilidad que mientras se realice la escritura en una tabla, otros procesos puedan acceder a la misma sin necesidad de bloqueos.
- Tipos de datos nativos: PostgreSQL provee soporte para: Números de precisión arbitraria - Texto de largo ilimitado - Figuras geométricas (con una variedad de funciones asociadas) - Direcciones IP (IPv4 e IPv6) - Bloques de direcciones estilo CIDR (Encaminamiento Inter-Dominios sin Clases) - Direcciones MAC - Arrays. Adicionalmente los usuarios pueden crear sus propios tipos de datos, los que pueden ser completamente indizables gracias a la infraestructura GiST de PostgreSQL. Algunos ejemplos son los tipos de datos GIS creados por el proyecto PostGIS que se verán más adelante en este capítulo.
- Otras características:
    - Claves foráneas
    - *Triggers* o disparadores, que soporta programación en C, C++, Java PL/Java Web, PL/Perl, plPHP entre otros
    - Vistas.
    - Integridad transaccional.
    - Herencia de tablas.
    - Tipos de datos y operaciones geométricas.

2.3. La Extensión postGIS

PostGIS es una extensión al sistema de base de datos objeto-relacional PostgreSQL, que permite el uso de objetos *GIS* (*Geographic information systems*). PostGIS incluye soporte de funciones básicas para el análisis de objetos GIS.

Esta creado por *Refractions Research Inc*, como un proyecto de investigación de tecnologías de bases de datos espaciales. Está publicado bajo licencia GNU[4].

Con PostGIS podemos usar todos los objetos que aparecen en la especificación OpenGIS como puntos, lineas, polígonos, multilíneas, multipuntos, y colecciones geométricas.

[3] MVCC Acceso concurrente multi-versión, por sus siglas en inglés

[4] GNU: Acrónimo recursive *GNU is Not Unix*, el proyecto GNU fue iniciado por Richard Stallman con el objetivo de crear un sistema operativo completamente libre.

2.3.1. Objetos GIS.

Los objetos GIS soportados por PostGIS son de características simples definidas por OpenGIS.

Ejemplos de la representación en modo texto:

- `POINT(0 0 0)`
- `LINESTRING(0 0,1 1,1 2)`
- `POLYGON((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))`
- `MULTIPOINT(0 0 0,1 2 1)`
- `MULTILINESTRING((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))`
- `MULTIPOLYGON(((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0)),((- 1 -1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1 0)))`
- `GEOMETRYCOLLECTION(POINT(2 3 9),LINESTRING((2 3 4,3 4 5))`

En los ejemplos se pueden apreciar características con coordenadas de 2D y 3D (ambas son permitidas por PostGIS). Podemos usar las funciones force_2d() y force_3d() para convertir una característica a 3d o 2d.

2.3.2. Forma CANONICA vs ESTANDAR.

OpenGIS define dos formas de representar los objetos espaciales:

- (WKT)*Well-known text* como los ejemplos anteriores.
- (WKB)*Well-know binary*.

Las dos formas guardan información del tipo de objeto y sus coordenadas.
Además la especificación OpenGIS requiere que los objetos incluyan el identificador del sistema de referencia espacial (`SRID`).El `SRID` es requerido cuando insertamos un objeto espacial en la base de datos.

Ejemplo:

```
INSERT INTO SPATIALDATABASE(THE_GEOM,THE_NAME)
VALUES(GeometryFromText('POINT(-126.4 45.32)',312),'Un Lugar')\
```

La función `GeometryFromText` requiere un numero `SRID`.

2.4. Estándar OpenGIS.

La especificación para SQL de características simples de OpenGIS define tipos de objetos GIS estándar, los cuales son manipulados por funciones, y un conjunto de tablas de metadatos. Hay 2 tablas de meta-datos en la especificación OpenGIS:

2.4.1. La tabla SPATIAL_REF_SYS:

Contiene un identificador numérico y una descripción textual del sistema de coordenadas espacial de la base de datos.

```
CREATE TABLE SPATIAL_REF_SYS(
SRID INTEGER NOT NULL PRIMARY KEY,
AUTH_NAME VARCHAR(256),
AUTH_SRID INTEGER,
SRTEXT VARCHAR(2048),
PROJ4TEXT VARCHAR(2048))
```

Las columnas de las tablas son las siguientes:

- SRID: Valor entero que identifica el sistema de referencia espacial.
- AUTH_NAME: El nombre del estándar para el sistema de referencia. Por ejemplo: EPSG.
- AUTH_SRID: El identificador según el estándar AUTH_NAME.
- SRTEXT: Una representación *Well-know text* para el sistema de referencia espacial.

```
PROJCS[“NAD83/UTM Zone 10N”,
GEOGCS[“NAD83”,DATUM[“North_American_Datum_1983”,
SPHEROID[“GRS 1980”,6378137,298.257222101]],
PRIMEM[“Greenwich”,0],UNIT[“degree”,0.0174532925199433]],
PROJECTION[“Transverse_Mercator”],
PARAMETER[“latitude_of_origin”,0],
PARAMETER[“central_meridian”,-123],
PARAMETER[“scale_factor”,0.9996],
PARAMETER[“false_easting”,500000],
PARAMETER[“false_northing”,0],
UNIT[“metre”,1]]
```

- PROJ4TEXT: Proj4 es una librería que usa PostGIS para transformar coordenadas. Esta columna contiene una cadena con definición de las coordenadas de Proj4 para un SRID dado.
  Ejemplo:

```
+proj=utm +zone=10 +ellps=clrk66 +datum=NAD27 +units=m
```

2.4.2. La tabla GEOMETRY_COLUMNS:

La tabla geometry_columns se define de la siguiente manera:

```
CREATE TABLE GEOMETRY_COLUMNS(
F_TABLE_CATALOG VARCHAR(256) NOT NULL,
F_TABLE_SCHEMA VARCHAR(256) NOT NULL,
F_TABLE_NAME VARCHAR(256) NOT NULL,
F_GEOMETRY_COLUMN VARCHAR(256) NOT NULL,
COORD_DIMENSION INTEGER NOT NULL,
SRID INTEGER NOT NULL,
TYPE VARCHAR(30) NOT NULL)
```

Descripción de cada una de las columnas:

- F_TABLE_CATALOG, F_TABLE_SCHEMA,F_TABLE_NAME.: Distingue íntegramente la tabla de características que contiene la columna geométrica.
- F_GEOMETRY_COLUMN: Nombre de la columna geométrica en la tabla de características.
- COORD_DIMENSION: Dimensión espacial de la columna (2D o 3D).
- SRID: Es una clave foránea que referencia a SPATIAL_REF_SYS.
- TYPE: Tipo de objeto espacial. POINT, LINESTRING, POLYGON, MULTYPOINT, GEOMETRYCOLLECTION. Para un tipo heterogéneo se debe usar el tipo GEOMETRY.

2.5. Creación de una tabla espacial

Para crear una tabla con datos espaciales realizamos dos pasos:

1. Creamos una tabla no espacial.

```
CREATE TABLE CALLES_GEOM(ID int4,NAME varchar(25))
```

2. Añadimos una columna (campo) espacial a la tabla usando la función *AddGeometryColumn* de OpenGIS.

```
AddGeometryColumn(<db_name>,<table_name>,
                  <column_name>,<srid>,
                  <type>,<dimension>)
```

Ejemplo:

```
SELECT AddGeometryColum('calles_db',
                        'calles_geom',
                        'geom',423,
                        'LINESTRING',2)
```

Ejemplos de creación de tablas con columnas geométricas:

- Parques:

```
CREATE TABLE PARQUES( PARQUE_ID int4,
                      PARQUE_NOMBRE varchar(128),
                      PARQUE_FECHA  date);
```

```
SELECT AddGeometryColumn( 'parque_db',
                          'parque',
                          'parque_geom',
                          128, 'MULTIPOLYGON' , 2 ) ;
```

- Calles: El tipo de dato espacial es genérico(GEOMETRY).

```
CREATE TABLE CALLES( CALLE_ID int4,
                     CALLE_NOMBRE varchar(128));
```

```
SELECT AddGeometryColumn( 'calles_db',
                          'calles',
                          'calles_geom',-1,'GEOMETRY',3);
```

2.6. Cargar datos GIS en la base de datos Espacial.

Hay tres formas de cargar datos en las tablas de nuestra base de datos.

- Usando el lenguaje SQL,
- Usando un cargador de archivos de figuras
- Y la más amigable, usando un software especial para esto denominado gvSIG[5], cuyo proceso lo veremos en el capítulo final de este trabajo.

- **Usando SQL.**

Usar el formato SQL es una manera sencilla de insertar los datos en PostGIS.

Podemos crear un archivo de texto con de sentencias INSERT y cargarlo con SQL monitor. Ejemplo: calles.sql

```
BEGIN;
INSERT INTO CALLES_GEOM(ID,GEOM,NAME)
VALUES (1, GeometryFromText ('LINESTRING(191232 243118,191108
243242)',-1),'Jeff Rd');
INSERT INTO CALLES_GEOM(ID,GEOM,NAME)
VALUES (1, GeometryFromText
('LINESTRING(189141 244158,189265 244817)',-1),'Geordie Rd');
INSERT INTO CALLES_GEOM(ID,GEOM,NAME)
VALUES (1, GeometryFromText
('LINESTRING(192783 228138,192612 229814)',-1),'Paul St');
INSERT INTO CALLES_GEOM(ID,GEOM,NAME)
VALUES (1, GeometryFromText
('LINESTRING(189412 252431,189631 259122)',-1),'Graeme Ave');
INSERT INTO CALLES_GEOM(ID,GEOM,NAME)
VALUES (1, GeometryFromText
('LINESTRING(190131 224148,190871 228134)',-1),'Phil Tce');
INSERT INTO CALLES_GEOM(ID,GEOM,NAME)
VALUES (1, GeometryFromText
('LINESTRING(198231 263418,198213 268322)',-1),'Dave Cres');
END;
```

El archivo puede cargar en la base de datos usando “psql”:

```
psql -d [base de datos] -f calles.sql
```

[5] gvSIG, es un proyecto de la Universidad Valenciana para manipular datos geográficos

- **Usar el Cargador.**

El cargador de datos "**shp2pgsql**" convierte archivos de figuras ESRI a SQL para su inserción en una base de datos PostGIS/PostgreSQL. El cargador tiene varios modos de operación que se seleccionan con los parámetros desde línea de comandos:

- **-d** Elimina la tabla de la base de datos antes de crear la tabla con los datos del archivo de figuras.
- **-a** Añade los datos del archivo de figuras a las tabla de la base de datos. El fichero debe tener los mismos atributos que la tabla.
- **-c** Crea una nueva tabla y llena esta con el archivo de figuras. Este es el modo por defecto.
- **-D** Crea una tabla nueva llenándola con los datos del fichero de formas. Pero usa el formato *dump* para la salida de datos que es más rápido que el *insert* de SQL. Se usará este para conjuntos de datos largos.
- **-s<SRID>** Crea y rellena una tabla geométrica con el SRID que se le pasa como parámetro.

Ejemplos:

```
shp2pgsql shapecalles tablacalles callesdb>calles.sql
psql -d callesdb -f calles.sql
```

Se puede hacer de una vez usando el símbolo de pipe:

```
shp2pgsql shapecalles tablacalles callesdb | psql -d callesdb
```

2.7. Recuperar datos GIS.

Tenemos 2 formas básicas para recuperar datos.
Mediante SQL o con un cargador de archivos de figuras.

- **Usar SQL.**

La forma más directa de hacerlo es usando un *SELECT*. Ver Ilustración 2.

```
db=# SELECT id, AsText(geom) AS geom, name FROM ROADS_GEOM;
id | geom                                          | name
---+-----------------------------------------------+-----------
 1 | LINESTRING(191232 243118,191108 243242) | Jeff Rd
 2 | LINESTRING(189141 244158,189265 244817) | Geordie Rd
 3 | LINESTRING(192783 228138,192612 229814) | Paul St
 4 | LINESTRING(189412 252431,189631 259122) | Graeme Ave
 5 | LINESTRING(190131 224148,190871 228134) | Phil Tce
 6 | LINESTRING(198231 263418,198213 268322) | Dave Cres
 7 | LINESTRING(218421 284121,224123 241231) | Chris Way
(6 rows)
```

- **Usar un cargador (*DUMPER*).**

Pgsql2shp conecta directamente con la base de datos y convierte una tabla en un archivo de figuras. La sintaxis es:

```
pgsql2shp [<opciones>] <basededatos> <tabla>
```

Opciones:

-d	Escribe un archivo de figuras 3D siendo el 2D el que tiene por defecto.
-f<archivo>	archivo de salida.
-p<puerto>	puerto de conexión con la base de datos.
-h<host>	host donde está la base de datos.
-p<password>	*password* para el acceso
-u<user>	usuario de acceso.
-g<col geom>	Si la tabla tiene varias columnas geométricas, selecciona la columna geométrica a usar.

CAPITULO 3

MAPSERVER & *OPENLAYERS*

3.1. Introducción

En los últimos tiempos se han hecho muy populares los mashups[1] que hacen uso de los más populares servicios de mapas de la actualidad -como por ejemplo: *Google Maps, Microsoft VE, Yahoo! Maps o Ask.com*- para usarlos como herramientas para la representación de información muy variada. Sin embargo, a los ya mencionados, y cuyos derechos pertenecen a las empresas o instituciones que les da nombre, se suma un servicio denominado *OpenLayers* cuya principal particularidad es ser de código abierto.

3.2. ¿Qué es *OpenLayers*?

En definitiva, *OpenLayers* es una librería en *JavaScript* pura para el manejo de mapas en los modernos navegadores web. Implementa un *JavaScript API* para construir aplicaciones geográficas, similar a *Google Maps* y *MSN Virtual Heart APIs*, con una importante diferencia – *OpenLayers* es Software Libre.

Uno de los principios básicos del desarrollo de *OpenLayers* ha sido mantener un conjunto de ejemplos[2] sencillos de mucha funcionalidad.

3.2.1. Herramientas

OpenLayers ofrece una variedad de herramientas para trabajar sobre los mapas. Entre estas herramientas está:

- *PanZoomBar*: Añade al mapa en cuestión una barra para poder un zoom panorámico.

[1] *Mashup*: Una aplicación web híbrida (mashup o remezcla), es un sitio web o aplicación web que usa contenido de otras aplicaciones Web
[2] Ejemplos disponibles en: http://openlayers.org/dev/examples/

- *LayerSwitcher*: Permite intercambiar *layers*, mostrando uno o varios a la vez. Aquí es conveniente utilizar transparencias para poder visualizar mapas de *layers* inferiores.
- *Permalink*: Indica un enlace hacia el mapa original por defecto.
- *Measurement*: permite dibujar líneas o polígonos y muestra las distancias o áreas respectivamente.

Entre otras herramientas que se mostraran en la interfaz final de este trabajo.

3.3. ¿Qué es Mapserver?

MapServer es un popular proyecto *Open Source* cuyo propósito es de mostrar mapas espaciales dinámicos sobre Internet, entre algunas características tenemos:

- Soporte para la visualización de una gran variedad de formatos raster, vectoriales y de bases de datos.
- Ejecución sobre diferentes plataformas (Windows, Linux, Mac OS X, etc.)
- Soporte para los principales lenguajes de scripting y entornos de desarrollo (*PHP, Python, Perl, Ruby, Java, .NET)*
- Proyecciones *on-the-fly*, "al vuelo".
- Alta calidad en *rendering* (representación de imágenes).
- Salidas de información completamente personalizables.
- Entorno de aplicación *Open Source, ready-to-use* (listo para usar)
- *MapServer* es un programa CGI que permanece inactivo en el servidor web, hasta cuando se le envíe un requerimiento o petición.
- *MapServer* utiliza información pasada en una petición (URL) y un mapfile crea la imagen a ser devuelta.
- La petición también puede devolver imágenes de leyendas, barras de escala, mapas de referencia y valores pasados como variables CGI

3.3.1. Composición de una aplicación Mapserver

Una simple aplicación *Mapserver* consiste de:

- **Mapfile:**

  Es un archivo de configuración de texto estructurado para la aplicación *Mapserver*. Este define el área del mapa, éste archivo le indica al programa *Mapserver* en donde están los datos y hacia donde se mostrarán las imágenes. También define las capas (layers), incluyendo la fuente de cada una, además de proyecciones y simbología. El mapfile tiene la extensión .map

- **Datos Geográficos:**

  Mapserver puede utilizar fuentes de datos geográficos de cualquier tipo. El formato por defecto es un archivo shape de ESRI[3]. Puede mostrar también formatos raster y vectoriales.

- **Página HTML:**

  Es la interfaz entre el usuario y *MapServer*. Esta normalmente está situada en la raíz del servidor web

- **MapServer CGI:**

  Es el archivo binario o ejecutable que recibe las peticiones y retorna imágenes, datos, etc. Está situado por lo general en el directorio *"cgi-bin"* o *"scripts"* del servidor http. Este programa se llama "mapserv".

- **HTTP Server:**

  Contiene las paginas HTML que serán mostradas a los usuarios a través de un navegador web. Un servidor HTTP o servidor web puede ser *Apache* o *Microsoft Internet Information Server* y debe estar instalado en la maquina donde funcionará *Mapserver*.

Una ilustración de la estructura de Mapserver se muestra a continuación:

[3] *Enviromental Systems Research Institute:* es un empresa dedicada al desarrollo y comercialización de Sistemas de Información Geográfica

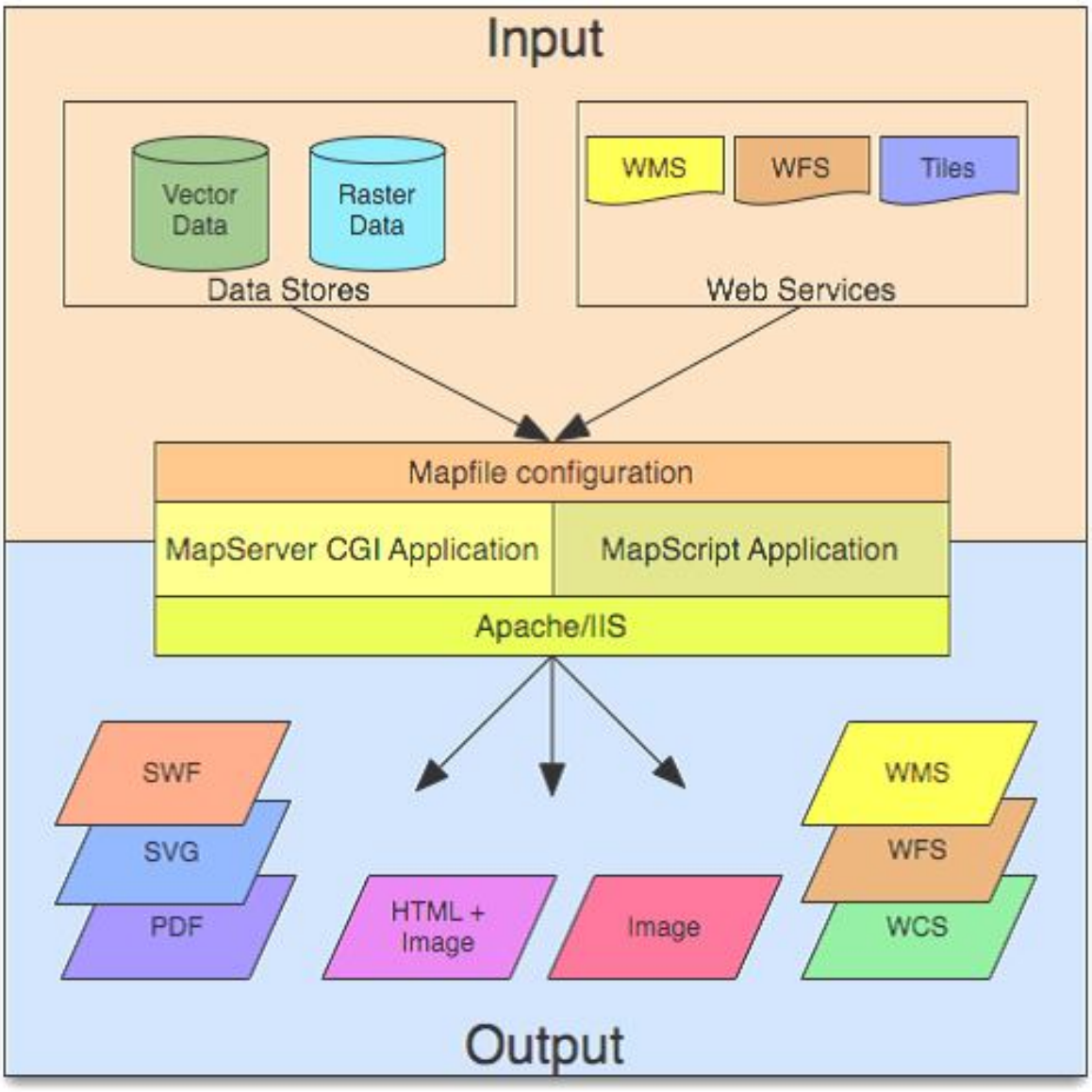


Figura 3.1. Anatomía de Mapserver [4]

3.3.2. Estructura del mapfile

El .map es el archivo de configuración básica de *Mapserver* para acceder a datos geográficos. El archivo es un archivo de texto *ASCII* [5], y está compuesto de diferentes

[4] *MapServer Documentation, Release 5.2.2. The MapServer Team, April 18, 2009*
[5] *American Standard Code for Information Interchange* - Código Estadounidense Estándar para el Intercambio de Información

objetos, donde cada objeto tiene una variedad de parámetros que hacen posible mostrar los datos.

Un ejemplo sencillo de un *mapfile* se muestra a continuación:

```
NAME "sample"
STATUS ON
SIZE 600 400
SYMBOLSET "../etc/symbols.txt"
EXTENT -180 -90 180 90
UNITS DD
SHAPEPATH "../data"
IMAGECOLOR 255 255 255
FONTSET "../etc/fonts.txt"

#
# Start of web interface definition
#
WEB
    IMAGEPATH "/ms4w/tmp/ms_tmp/"
    IMAGEURL "/ms_tmp/"
END

#
# Start of layer definitions
#
LAYER
    NAME 'global-raster
    TYPE RASTER
    STATUS DEFAULT
    DATA bluemarble.gif
END
```

Figura 3.2. Ejemplo básico de un mapfile.

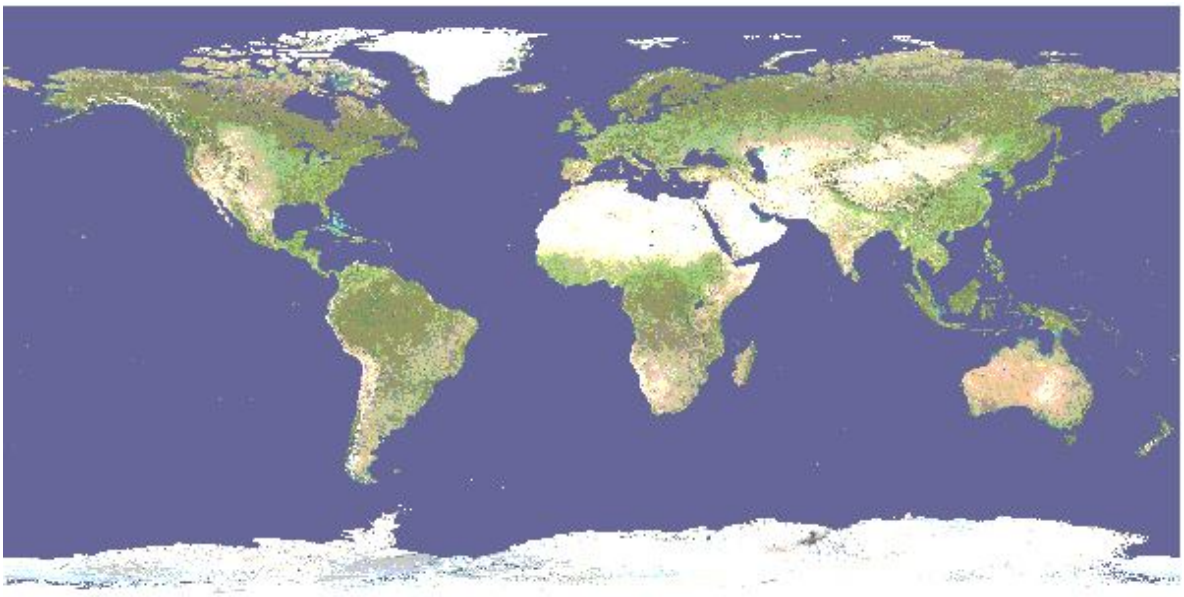


Figura 3.3. Resultado de un mapfile básico

CAPITULO 4

OBTENCIÓN DE LA CARTOGRAFÍA DIGITAL.

4.1. Introducción

La información cartográfica digital a ser publicada, específicamente la correspondiente a la C.R.P.[1], está contenida en el DVD de Geomática 2008, que fue desarrollado por la Universidad del Azuay.

El ámbito de este trabajo se centrará en los mapas temáticos contenidos en dicha cartografía.

4.2. Selección de mapas temáticos a publicar.

A continuación se detallan los mapas que serán publicados. Todos estos se encuentran en el directorio \CRP_SAM56_V3\MAPAS

- ASPECTOS:
    - Mapa de aspectos
    - Pendientes en grados
    - Pendientes en porcentaje
- CLIMA:
    - Precipitación
    - Temperatura
- COBERTURA DEL SUELO:
    - Páramo Amenazado
    - Vegetación Leñosa
- EDAFOLOGÍA:
    - Regímenes de Humedad
    - Regímenes de Temperatura
- MODELO DIGITAL DEL TERRENO

[1] C.R.P. Siglas de Cuenca del Río Paute, asumidas para el resto del trabajo.

Para una ilustración ver Anexo I.

4.3. Análisis y consolidación de la información a publicar.

Para facilitar la futura manipulación de la información utilizada en la presente monografía, se ha elaborado un cuadro, a manera de un diccionario de datos que contiene el detalle de los archivos usados, así como los nombres para las capas (mapfile) y para la base de datos espacial (postgreSQL). Ver anexo II.

4.4. Personalización de la interfaz web.

Para seguir el estándar de estilos y colores del portal Web de la Universidad del Azuay, el desarrollo de la interfaz se basará en el *Manual de uso y manejo de la página web de la Universidad del Azuay*[2], que define además de colores y estilos; los tamaños, espaciados, tipos de letras y estructura de directorios.

El diseño de la interfaz está basado en una plantilla publicada en: http://www.uazuay.edu.ec/plantilla.html. Para el código fuente ver Anexo III.

Según esta plantilla, y basado en el manual de uso de la página web de la Universidad del Azuay, se establece que un documento a ser publicado debe contener los siguientes elementos.

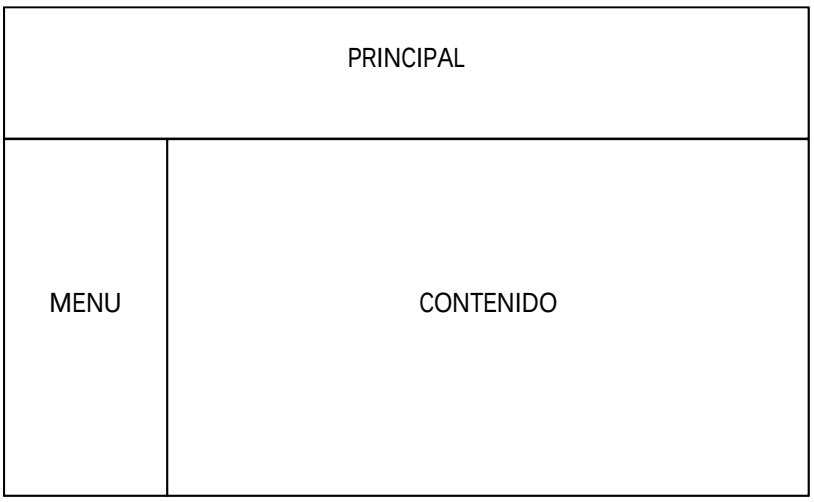


Figura 4.1. Disposición de marcos

[2] SurrealEstudio, Cuenca marzo del 2003

El frame *Principal* llama al archivo, *franja.html* que contiene la imagen que cambiara regularmente. El frame *Menu*, contiene al archivo *menu.html* que muestra el menú desplegable. Finalmente el frame *Contenido*, contiene los diferentes archivos **.html* que serán llamados desde *menu.html.*

Con estas referencias ya estudiadas, el proceso de diseño de la interfaz para la publicación de mapas quedaría de la siguiente manera.

cuerpo.html	Que contiene los frames en cuestión.
franja.html	Que contiene la franja que cambiara hasta con 9 diferentes tipos de imágenes.
menu.html	Que contiene el menú desarrollado en JavaScript y que es Opensource.
contenido.html	Contiene la información que será llamada desde el menú deplegable. (Mapas)

Tabla 4.1. Archivos que conforman nuestra interfaz web

Finalmente la interfaz web quedaría de la siguiente manera:

Figura 4.2. Interfaz web

Únicamente quedaría depurar más las herramientas de los *Openlayers* para que se presenten de manera ordenada en el espacio o "div" donde se presenta el mapa.

Para cada mapa temático de los elegidos para publicar, se creó un archivo *.html con la siguiente disposición, en donde se puede apreciar la similitud con el archivo *.mxd generado por la Universidad del Azuay que se muestra más abajo. Para los demás archivos, ver Anexo V.

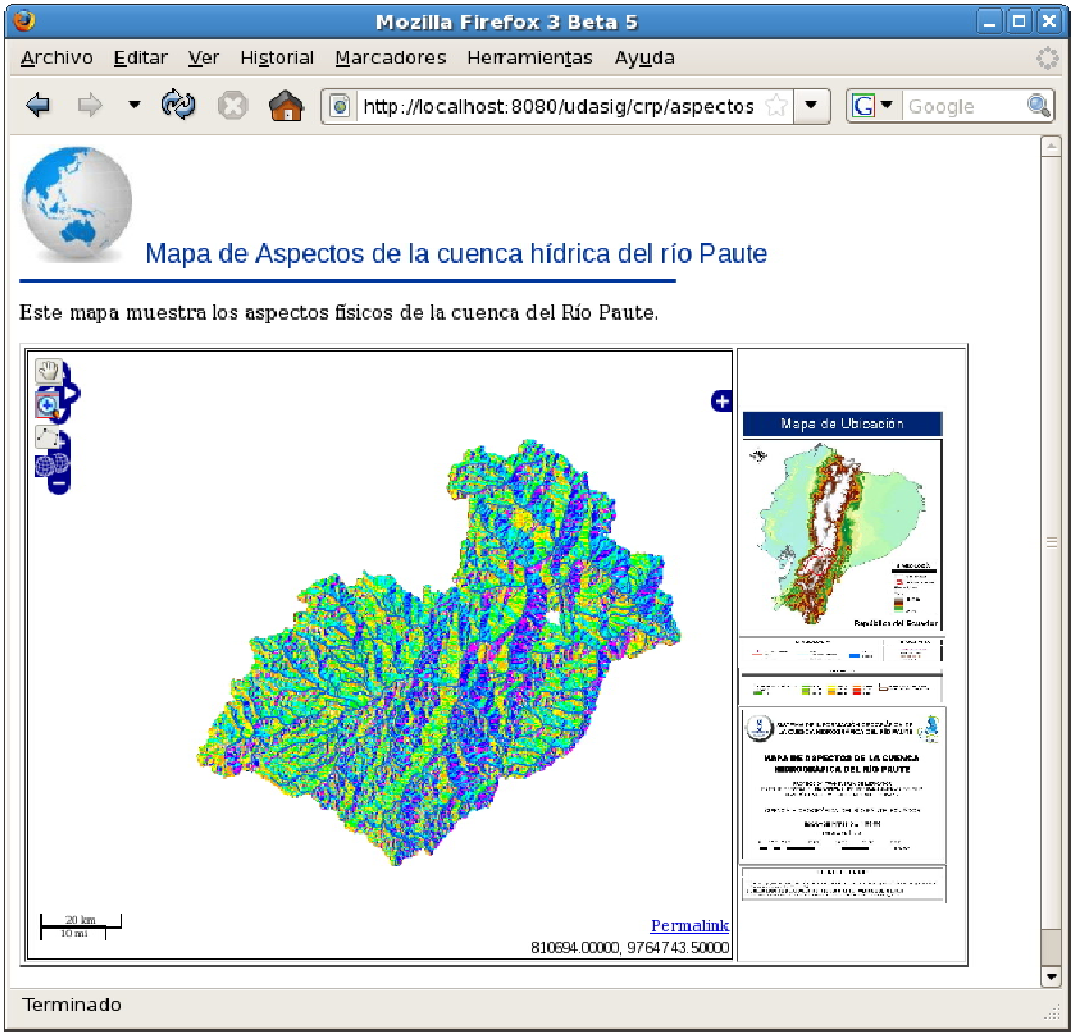


Figura 4.3: Archivo “aspectos.html”

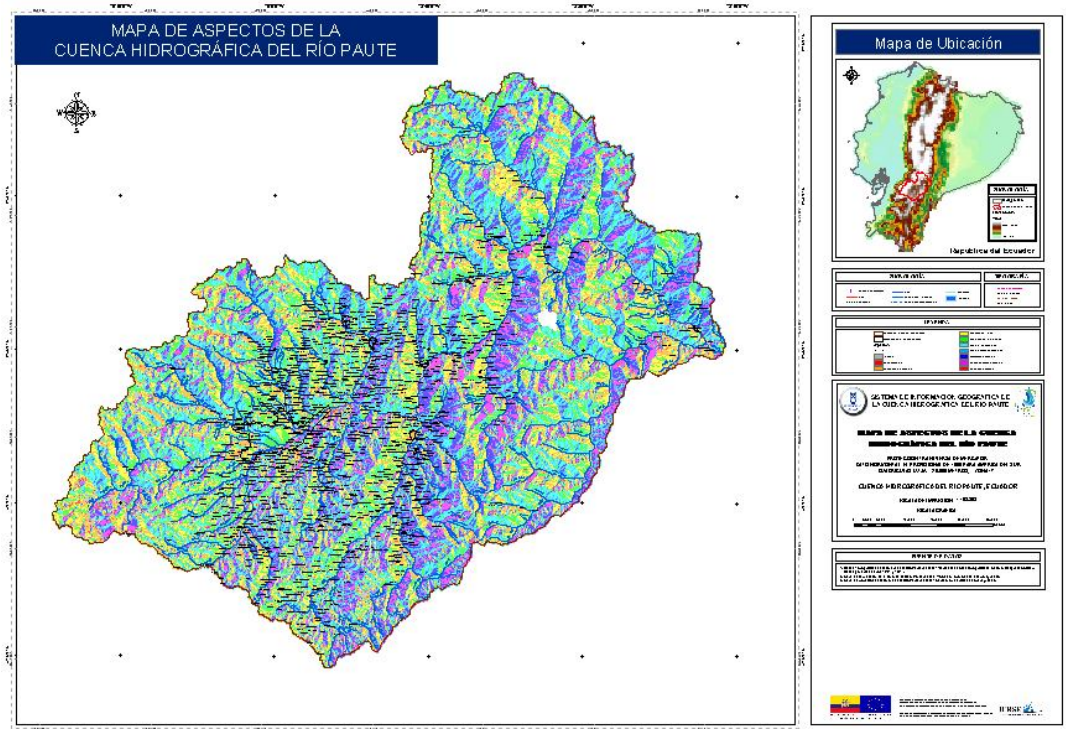


Figura 4.4: Archivo ASP aspectos.mxd

Para agregar mayor funcionalidad a la interfaz, se añadió un enlace en cada una de las imágenes que están al costado derecho del mapa, que abrirá en una nueva ventana la misma imagen pero con mejor resolución, esto para poder apreciar de mejor manera la información que contiene, por ejemplo:

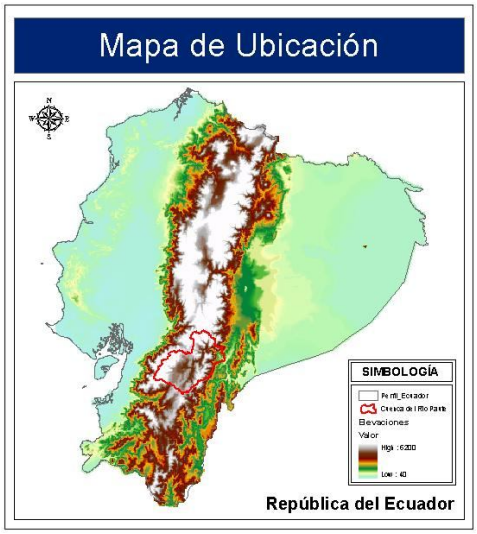


Figura 4.5. Mapa de Ubicación de la cuenca del río Paute

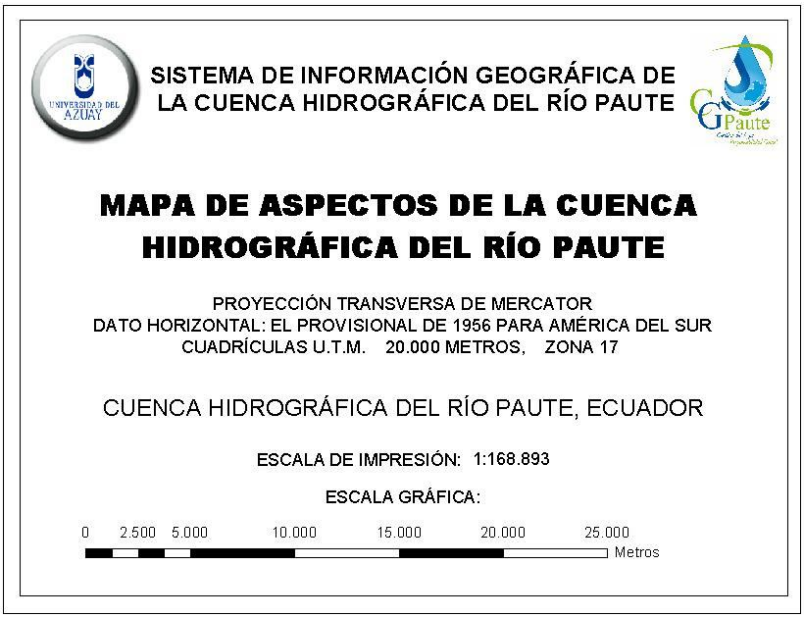


Figura 4.6. Titulo del mapa de aspectos

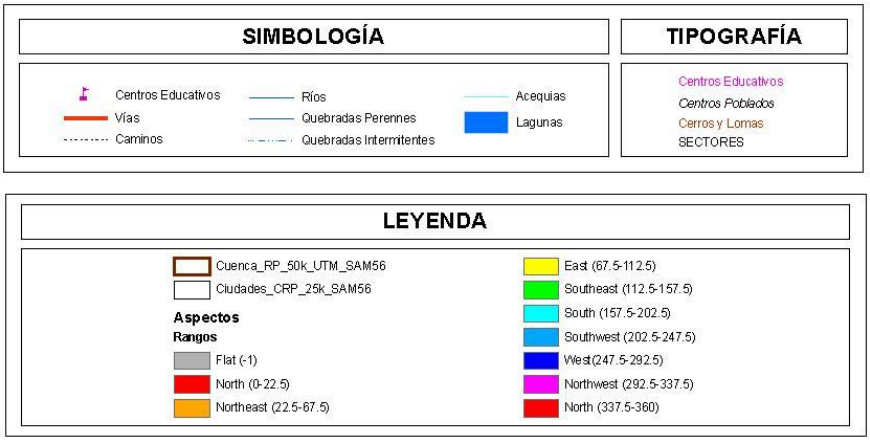


Figura 4.7. Leyenda del mapa de aspectos

Además se agregó una pequeña descripción a cada mapa en la parte superior de éste, justo debajo del Título del Mapa. Dicha descripción dará una referencia de que información contiene el mapa, así como también de que capas lo componen.

Las herramientas de Openlayers que se utilizaron son las siguientes:

	Zoom & Fractionalzoom: Opción que permiten hacer acercamientos y alejamientos, ya sea por medio del *mouse scroll* o haciendo click en el mapa y arrastrando el mouse.

	Pannig: o paneo, es una herramienta que permitirá mover el mapa libremente al arrastrarlo con el mouse.
	Maxextent: o zoom por defecto, que presentará la imagen a su disposición original.

Tabla 4.2. Herramientas de Openlayers

CAPITULO 5

PROCEDIMIENTO PARA PUBLICAR LA INFORMACIÓN

5.1. Introducción

Este capítulo se detallará la secuencia para la publicar información cartográfica que posee la Universidad del Azuay.

Esta metodología no es obligatoria y se pueden utilizar otras herramientas para obtener los mismos resultados.

5.2. Selección del mapa a publicar.

Como ejemplo se tomará al mapa temático ASP Aspectos.mxd que se encuentra dentro del directorio MAPAS de la cartografía de la Universidad.

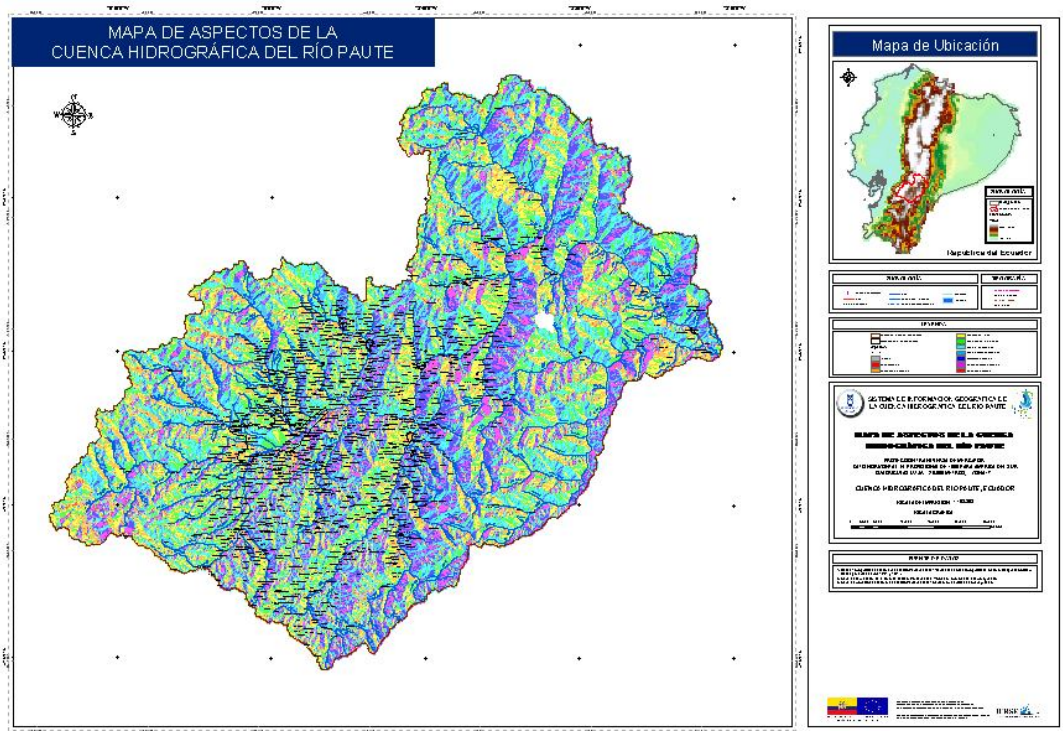


Figura 5.1. Mapa de Aspectos de la cuenca hídrica del río Paute.

5.3. Edición de campos que pueden dar conflictos.

Aunque PostgreSQL pueda soportar vocales con tildes o la letra “ñ”, se decidió modificar los atributos de los shape para que no contengan estos caracteres. Reemplazando por ejemplo las “ í ” por “ i ”, o la letra “n” por la “ñ”, como se detalla a continuación. Todo esto para evitar posibles errores posteriores.

Primero seleccionamos la capa cuyos atributos vamos a modificar, damos click derecho y seleccionamos “Open Attribute Table”

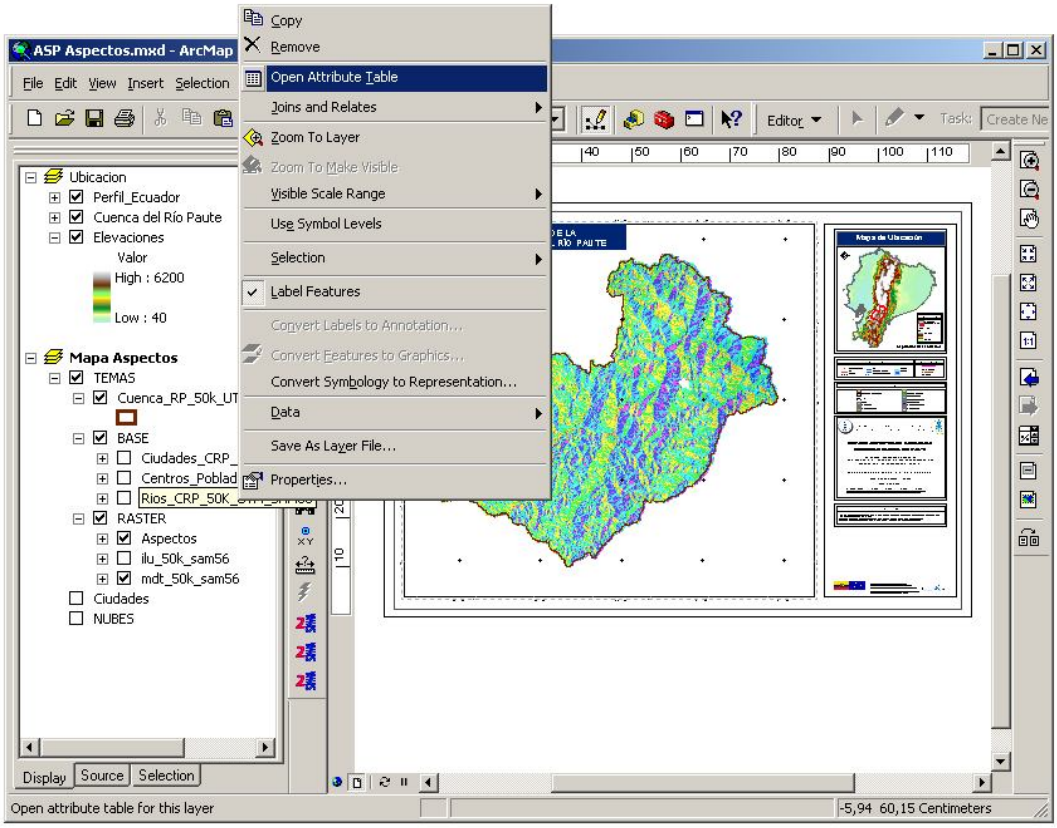


Figura 5.2. Apertura de la tabla de atributos

Con la tabla de atributos abierta, vamos a activar la opción de editar, haciendo click en la barra “Editor”, y seleccionando “Start Editing”

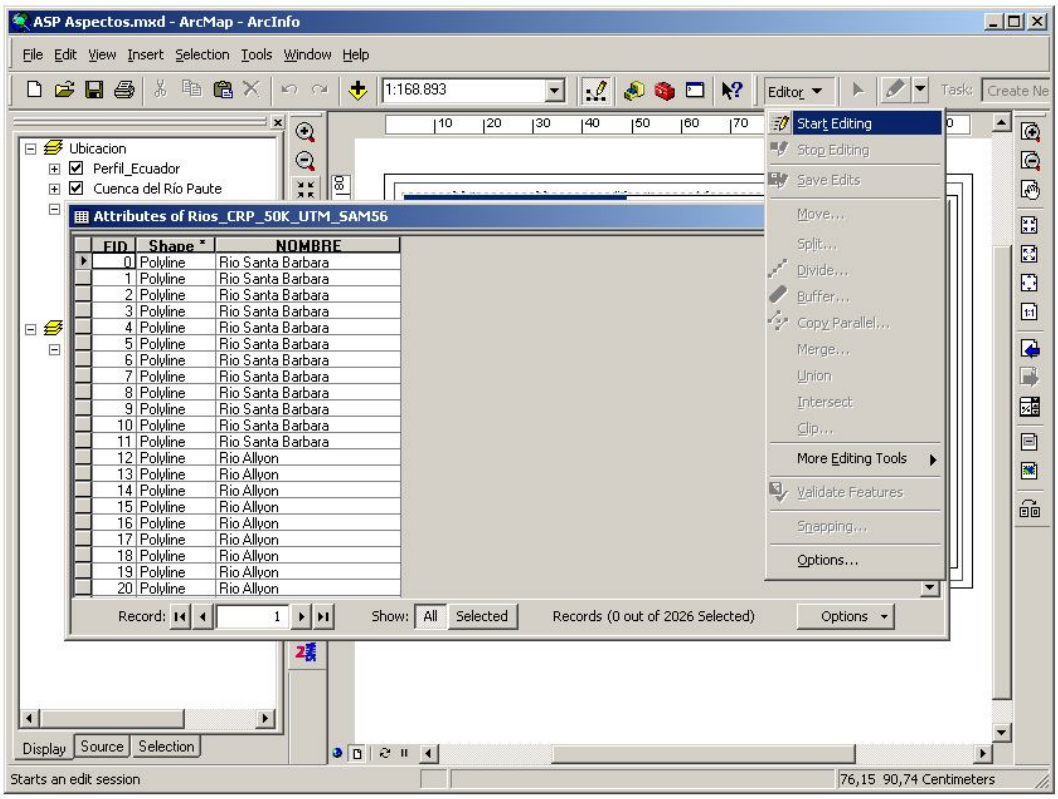


Figura 5.3.Inicio del proceso de edición

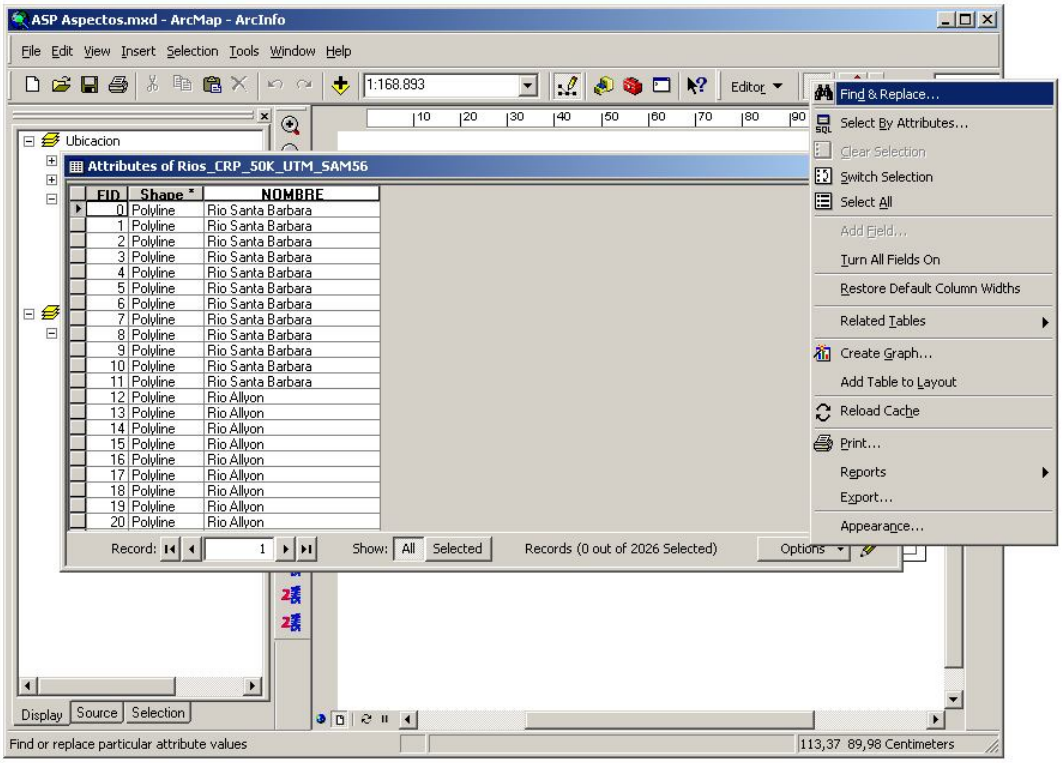


Figura 5.4. Buscar y reemplazar

Con la opción de edición habilitada, damos click en “Options” y seleccionamos “Find & Replace”, que es donde haremos los cambios requeridos.

Finalmente cerramos la ventana de “Find & Replace” y detenemos el modo edición, haciendo click en “Editor” > “Stop editing”, y aceptamos y guardamos los cambios.

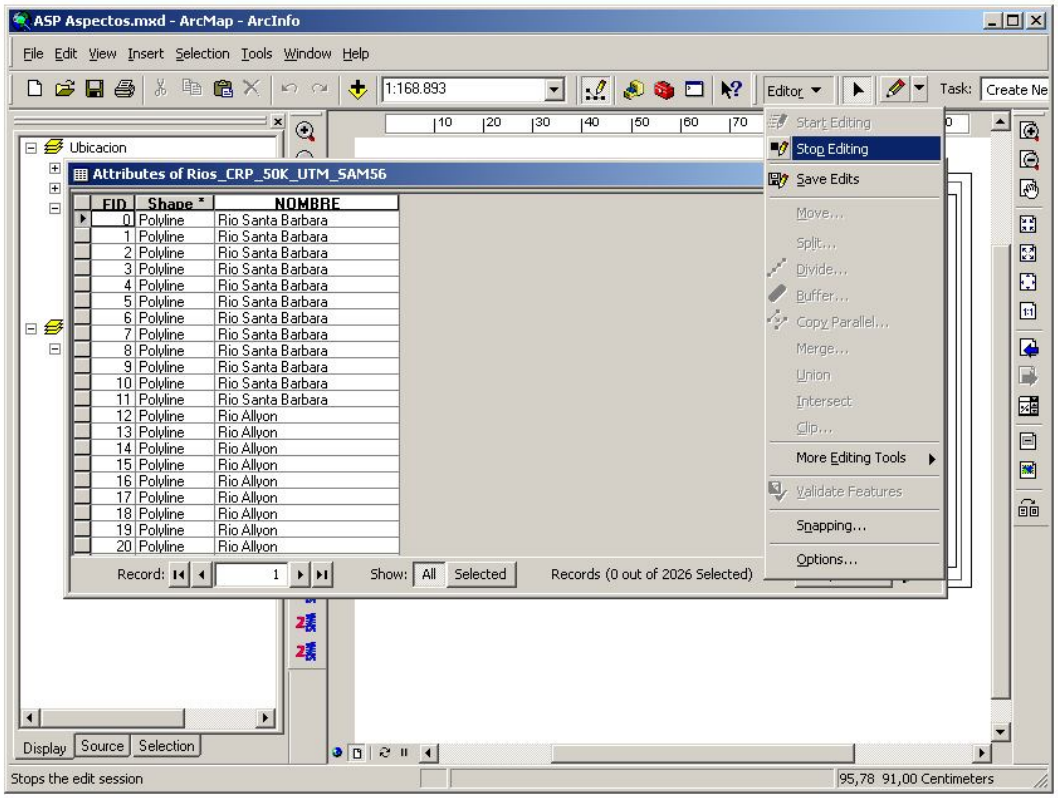


Figura 5.5. Finalización del proceso de edición.

5.4. Migración de datos ESRI (*.shp) a PostgreSQL (Linux)

Para hacer más fácil y rápida la migración a PostgreSQL, vamos a copiar todos los *shapes* a una sola carpeta (ej: C:\shape) y dicha carpeta la copiamos a algún directorio del servidor. (ej: /shape).

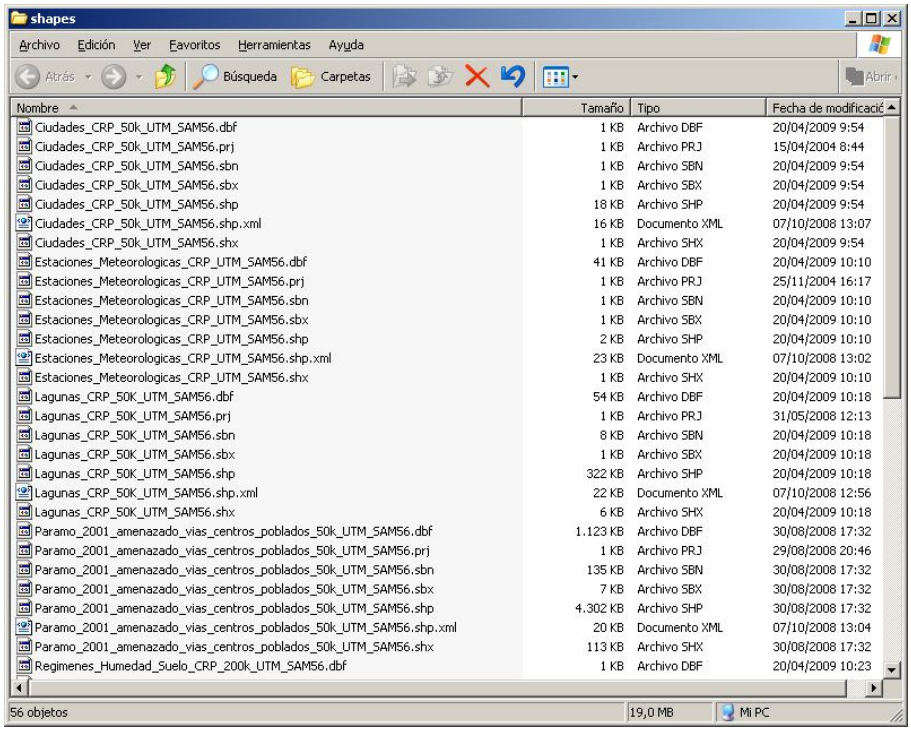


Figura 5.6. Archivos .shp agrupados en Windows

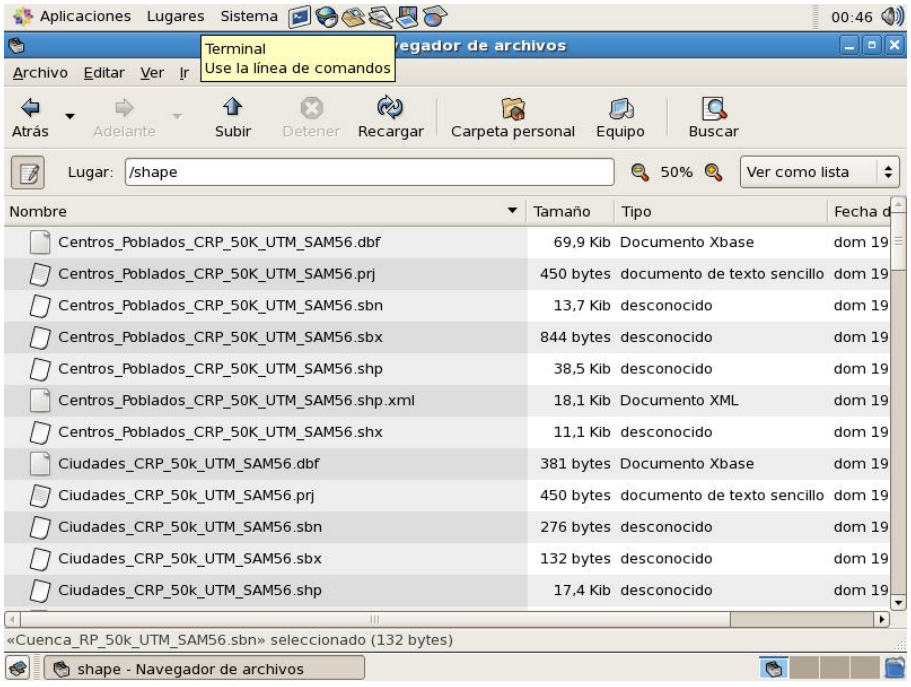


Figura 5.7. Archivos .shp agrupados en Windows

Para generar un fichero .map: Para crear un mapfile, nos valemos de una extensión de ArcMap denominada “mxd2wms”.

En la barra de menú de ArcMap seleccionamos “Tools” > “Customize..”

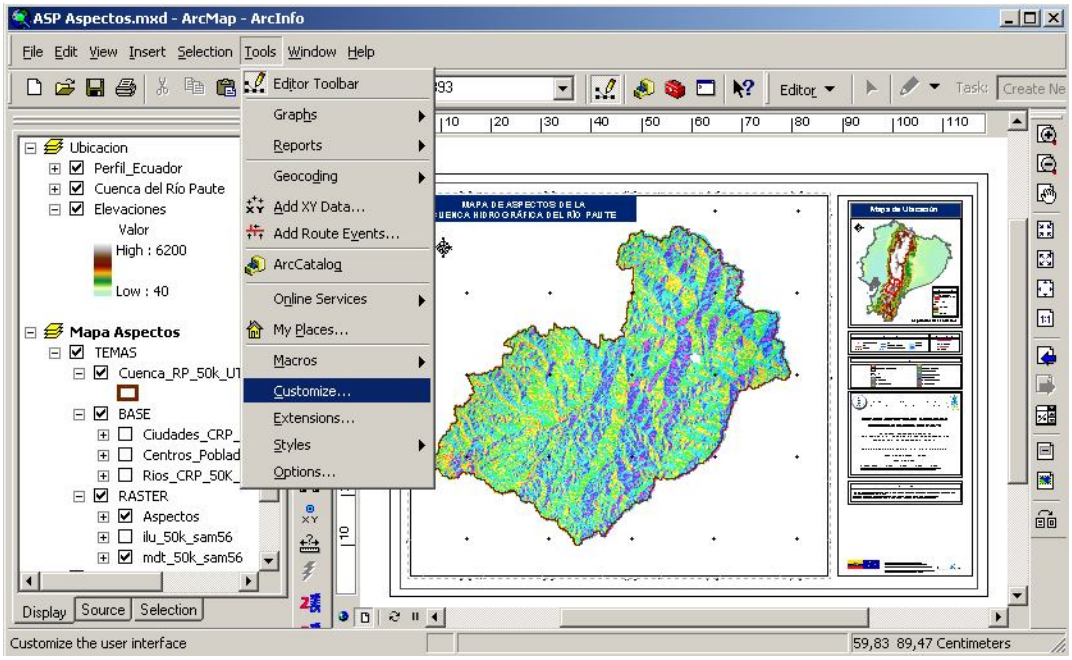


Figura 5.8. Personalización de herramientas de ArcMAP.

Dentro de la ventana Customize, seleccionamos el botón inferior “Add from file...” y escojemos la DLL de mxd2wms.

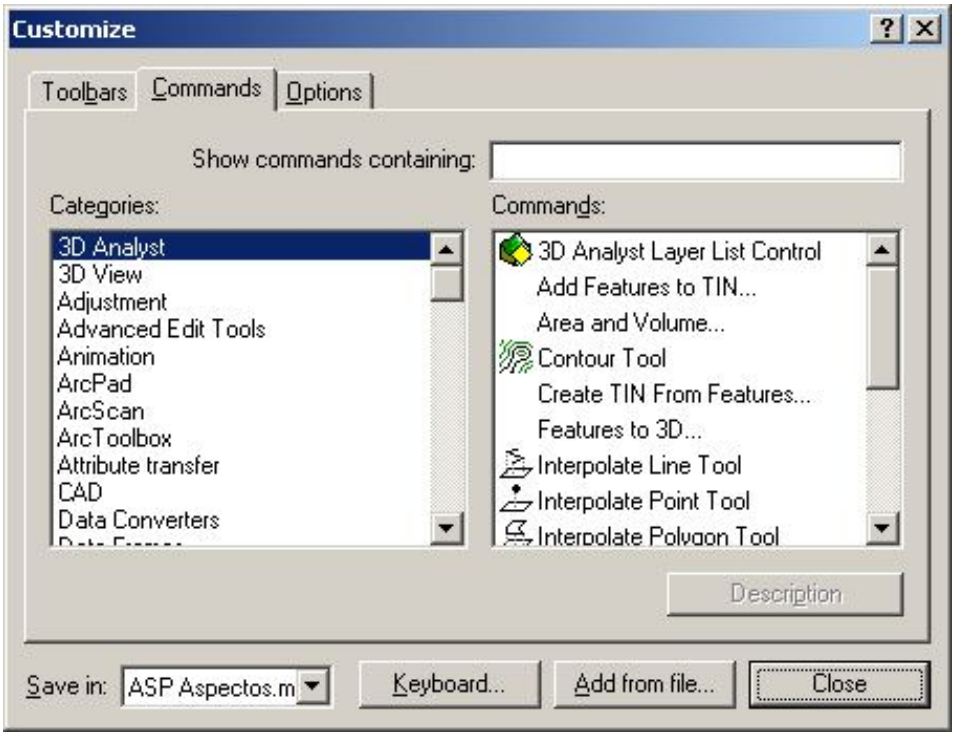


Figura 5.9. Importación desde archivo mxd2wms.dll

Con todo esto, se nos habilita un nuevo icono  , como el de la figura, que nos servirá para convertir todas nuestras capas del mapa temático en un mapfile de mapserver.

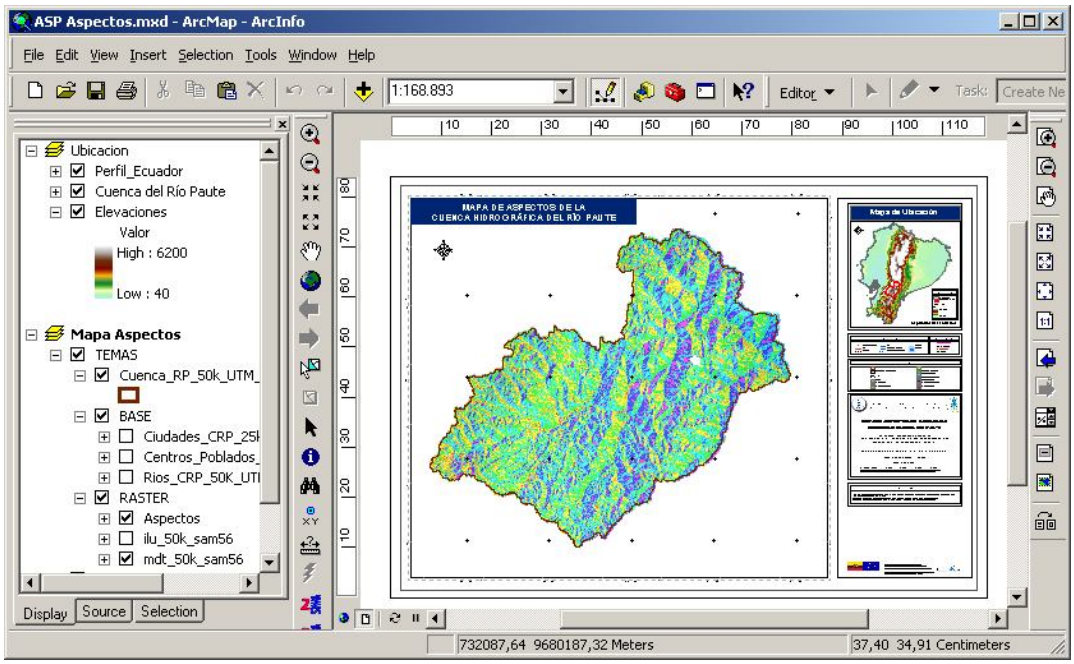


Figura 5.10. Habilitación de la opción “2wms”

Para ello damos click en el nuevo ícono y seguimos las instrucciones del asistente, para finalizar cerramos y nos aparece la pantalla “Save as…”, donde debemos especificar el tipo de archivo a *“MapServer files (*.map)”*, le damos un nombre y guardamos.

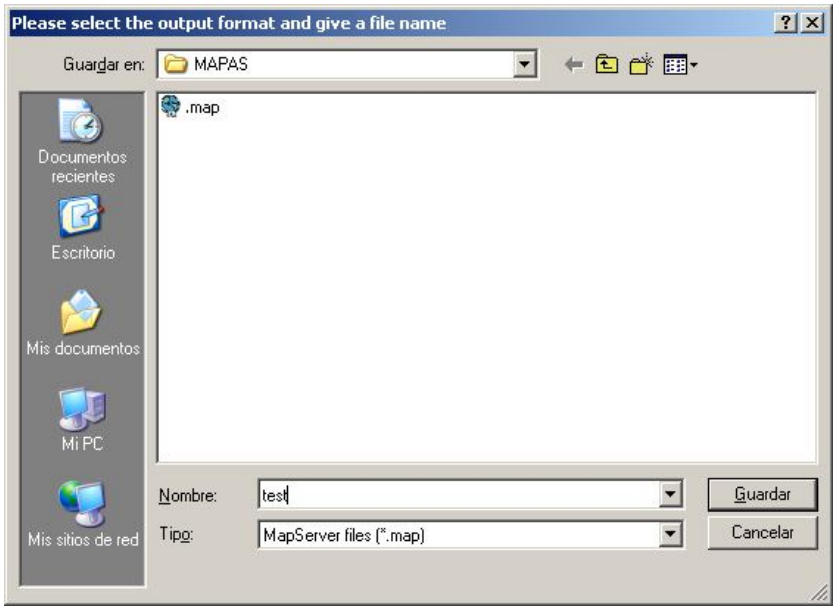


Figura 5.11. Guardado del mapfile

5.5. Configuraciones del servidor de mapas.

Adecuación del CGI de mapserver. Cuando tenemos varios mapfile en nuestro servidor HTTP, será necesario especificar diferentes CGI mapserver, esto lo hacemos simplemente clonando el archivo “mapserv” que se encontrara dentro de la carpeta CGI-BIN del servidor web, en este caso bajo el directorio www del Apache.

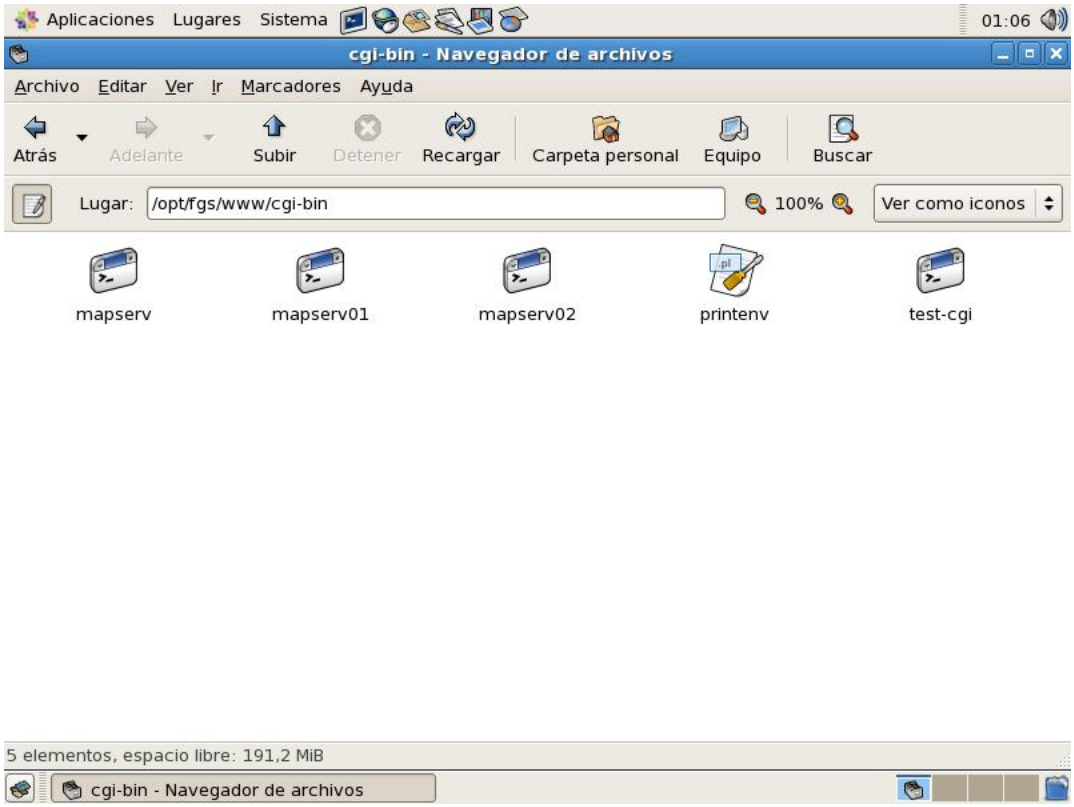


Figura 5.12. Contenido del directorio CGI-BIN

Configuración del Apache (httpd.conf): Tenemos que hacer que Apache trabaje con Mapserver, para ello editamos el archivo de configuración “httpd.conf” y agregamos las siguientes líneas.

```
# "$FGS_HOME/www/cgi-bin" should be changed to whatever your ScriptAliased
# CGI directory exists, if you have that configured.
<Directory "$FGS_HOME/www/cgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
</Directory>
```

```
# "$FGS_HOME/www/cgi-bin" should be changed to whatever your ScriptAliased
# CGI directory exists, if you have that configured.
<Directory "$FGS_HOME/www/cgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
    SetEnvIf Request_URI "/cgi-bin/mapserv01"
             MS_MAPFILE=/opt/fgs/www/htdocs/mapfiles/mapfile01.map
    SetEnvIf Request_URI "/cgi-bin/mapserv02"
             MS_MAPFILE=/opt/fgs/www/htdocs/mapfiles/mapfile02.map
</Directory>
```

Figura 5.13. Edicion del archivo httpd.conf

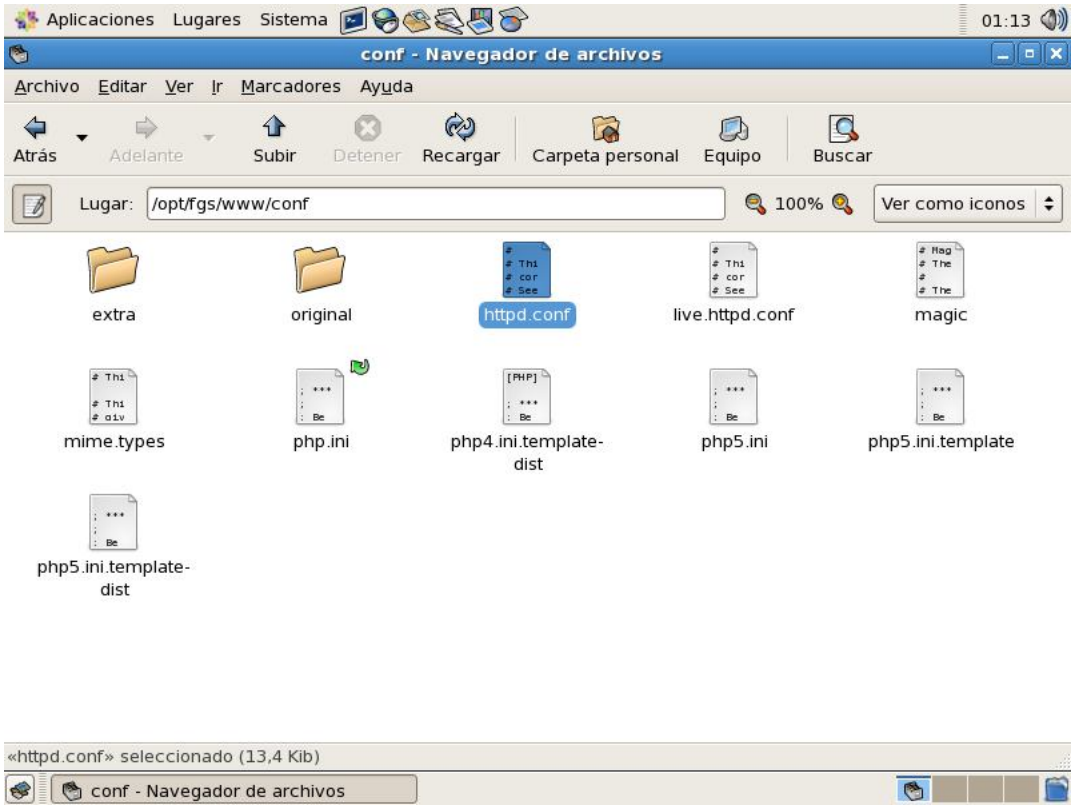


Figura 5.14. Ubicación del archivo httpd.conf

Edición del mapfile (Linux): En vista que el mapfile fue generado desde Windows, va a ser necesario editarlo para su correcto funcionamiento en el ambiente Linux. Por ejemplo:

- Reemplazamos “C:/” por “\” y todos los “/” por “\”.

- Y revisamos que los directorios y paths coincidan con la ubicación de los archivos en Linux.

```
DATA 'C:\CRP_SAM56_V3\RASTER\mdt_50k_sam56.img'
```

cambiamos por

```
DATA '/CRP_SAM56_V3/RASTER/mdt_50k_sam56.img'
```

5.6. Conciliación de la interfaz en el servidor HTTP (Openlayers)

Para probar que todo funcione con normalidad, realizamos una prueba de una de las páginas HTML creadas.

Figura 5.15. Interfaz web

BIBLIOGRAFÍA

http://www.maestrosdelweb.com

http://openlayers.org/

http://mapserver.org/

http://www.osgeo.org/

http://www.postgresql.org/

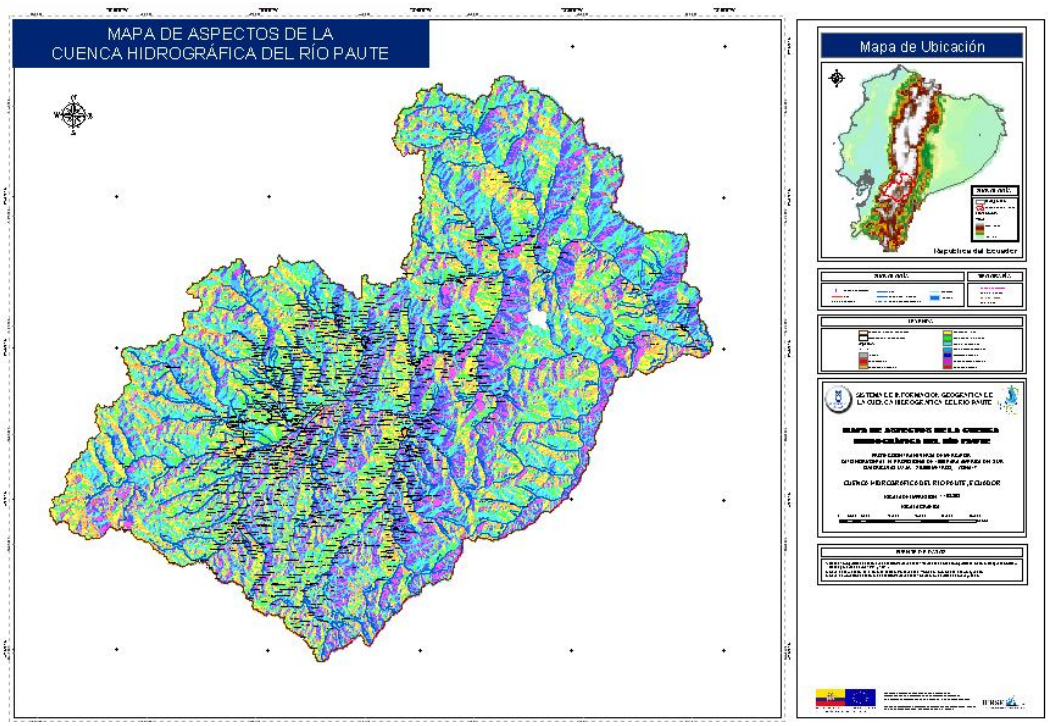
http://postgis.refractions.net/

http://postgis.refractions.net/documentation/postgis-spanish.pdf

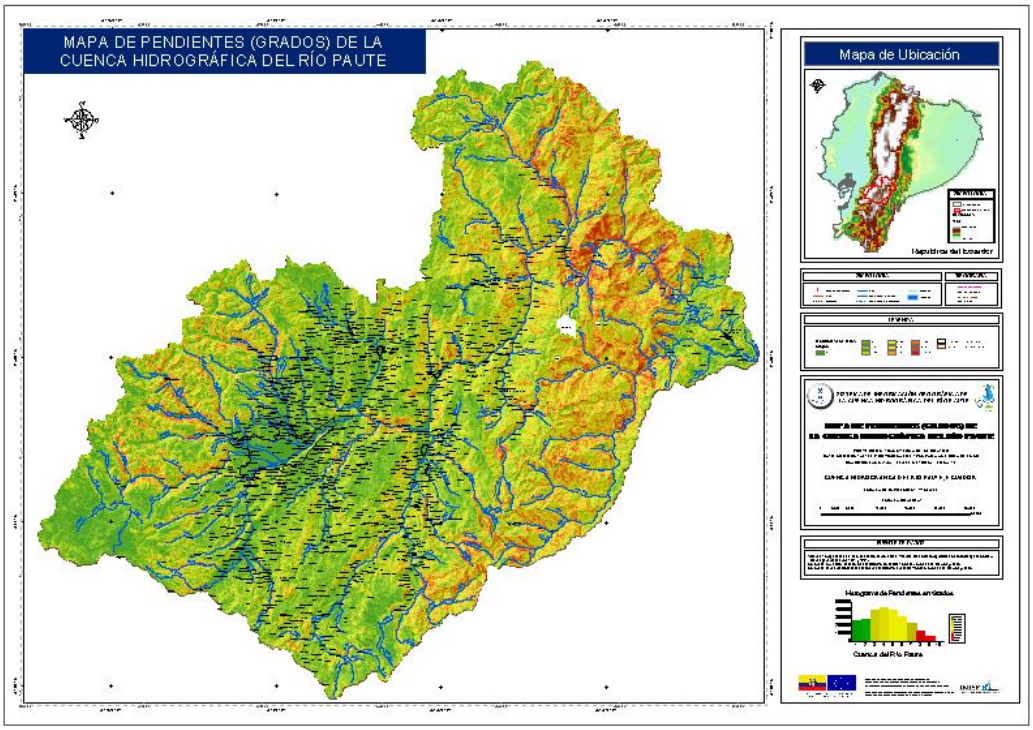
http://www.gvsig.gva.es/

ANEXO I

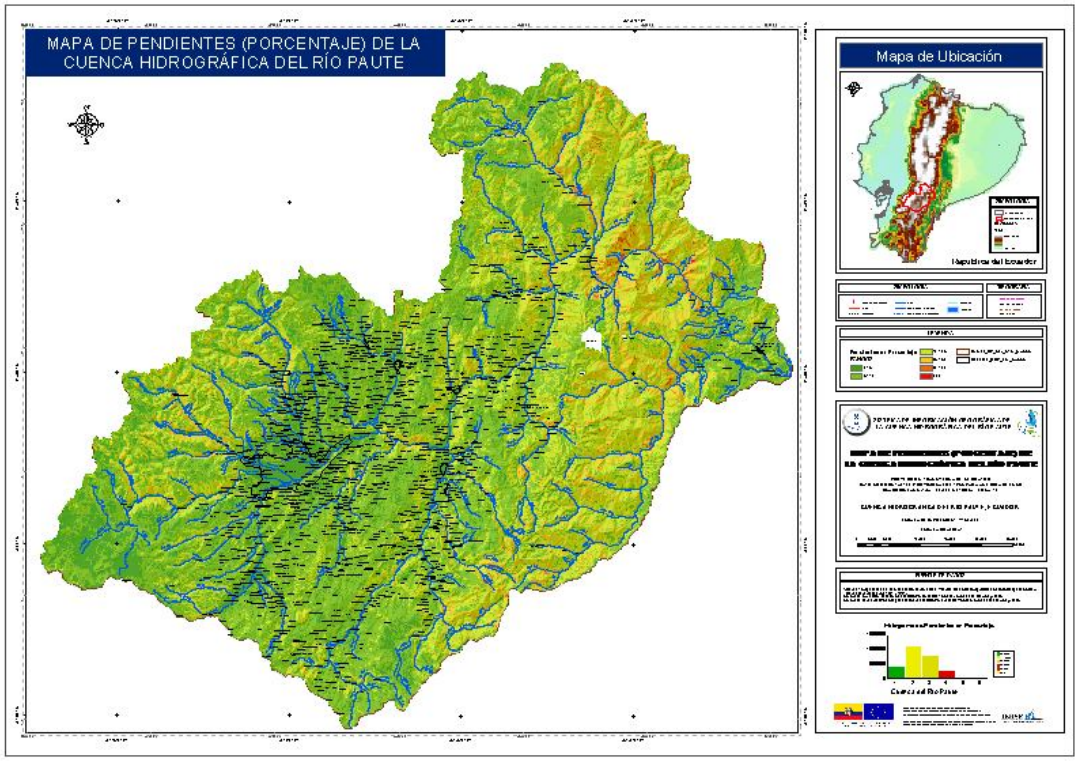
MAPA DE ASPECTOS



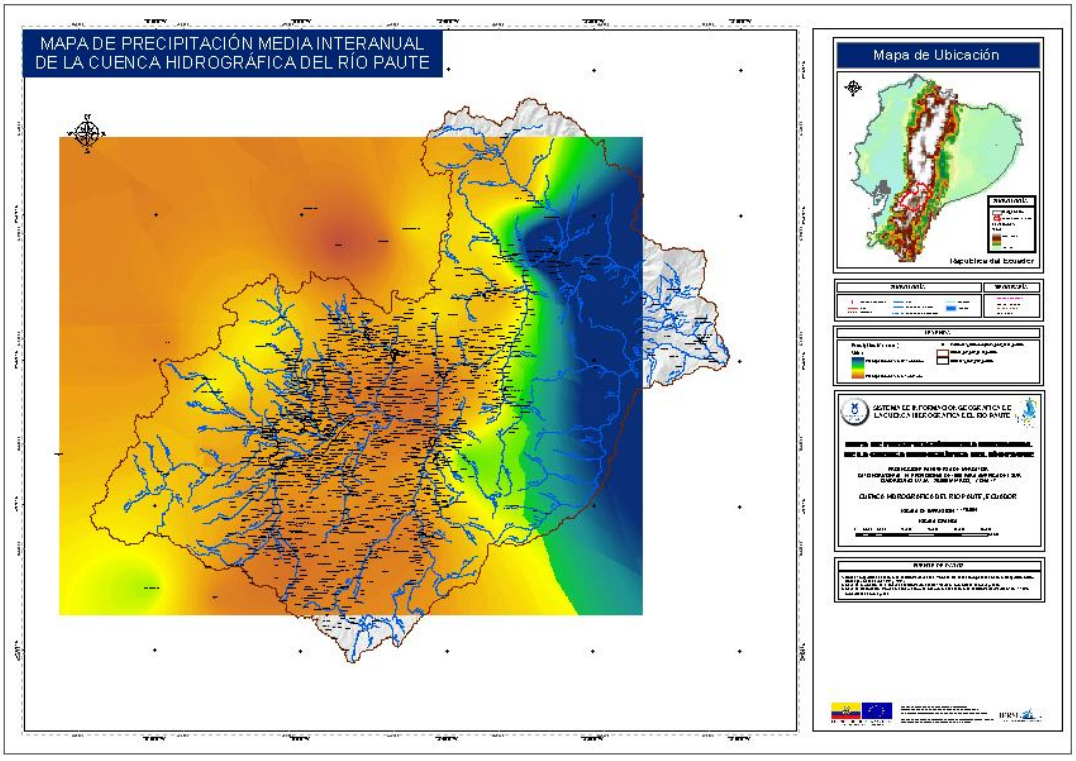
PENDIENTES EN GRADOS



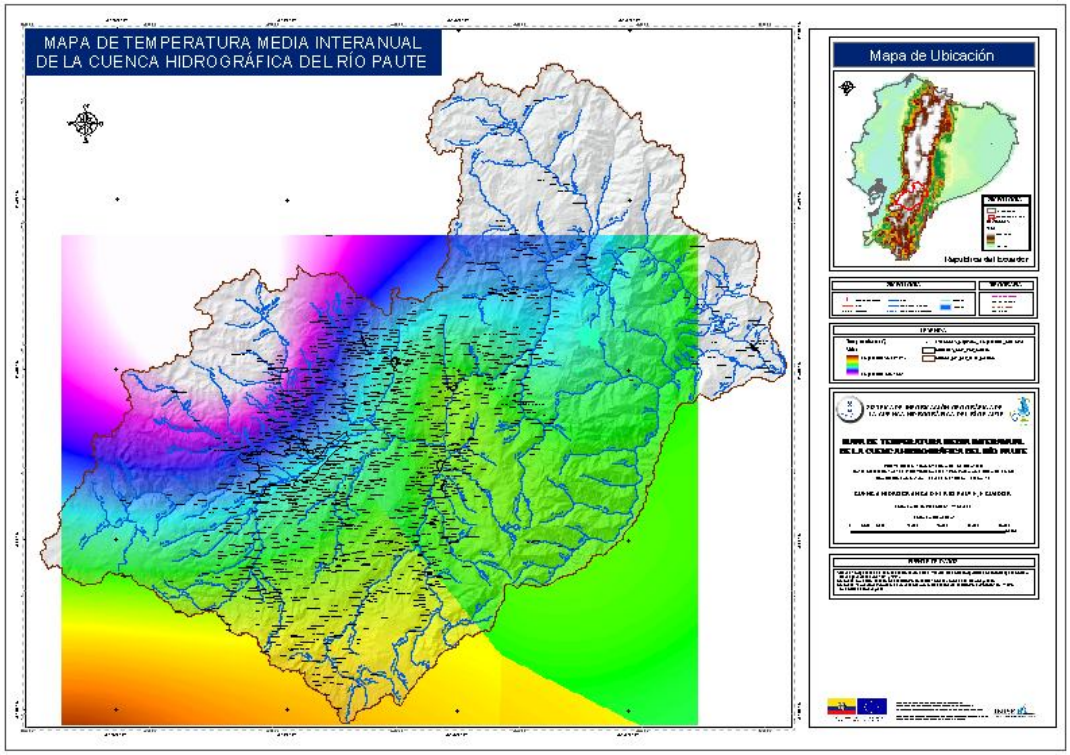
PENDIENTES EN PORCENTAJE



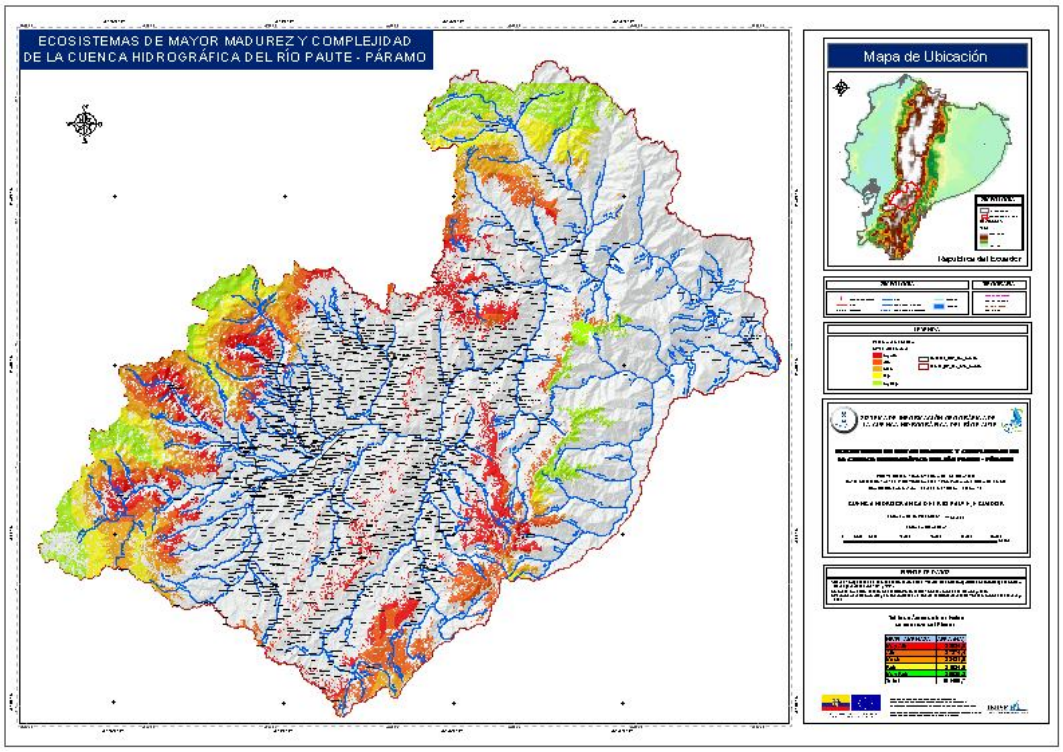
PRECIPITACION



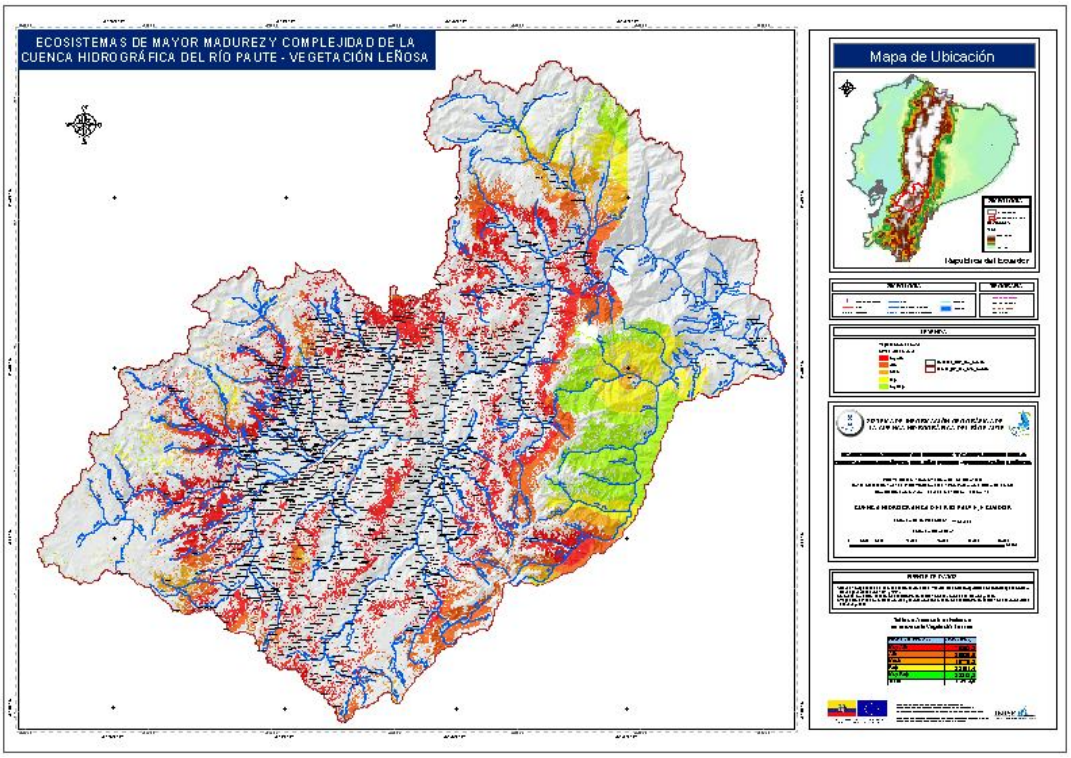
TEMPERATURA



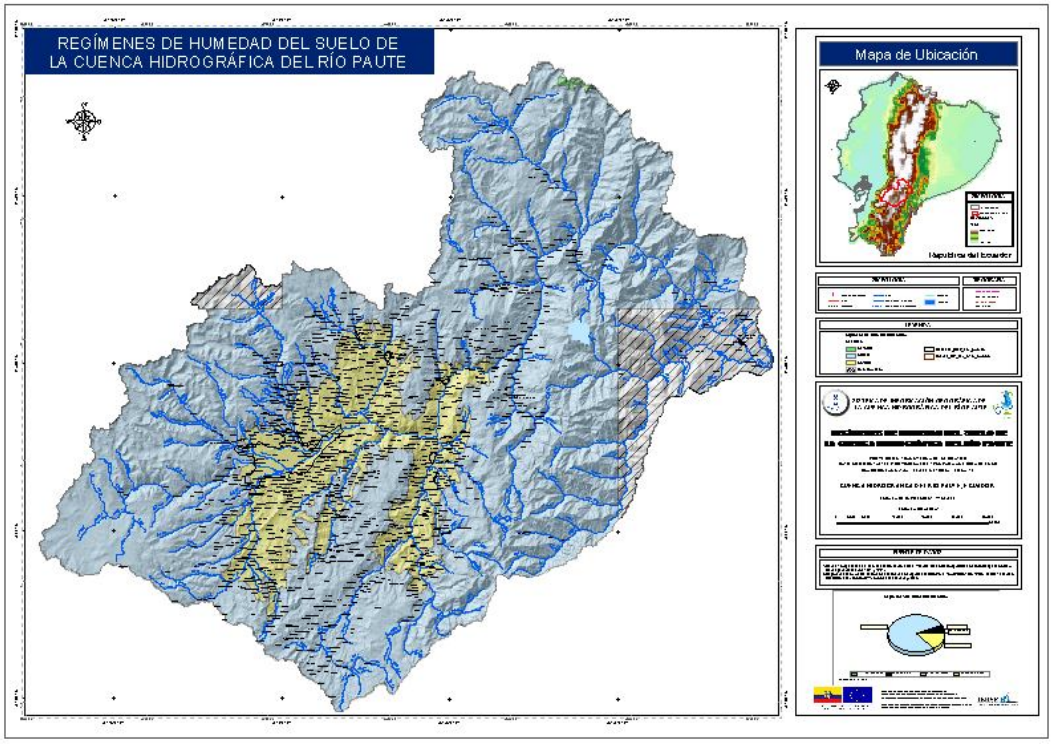
COBERTURA DEL SUELO: Páramo Amenazado



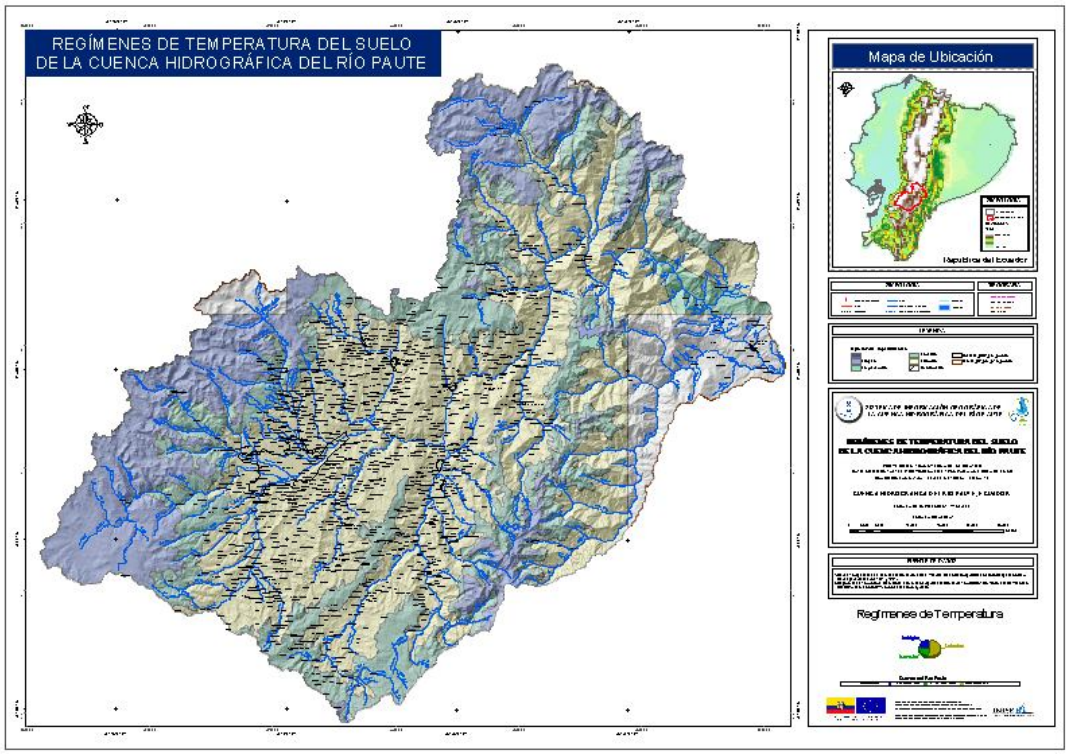
COBERTURA DEL SUELO: Vegetación Leñosa



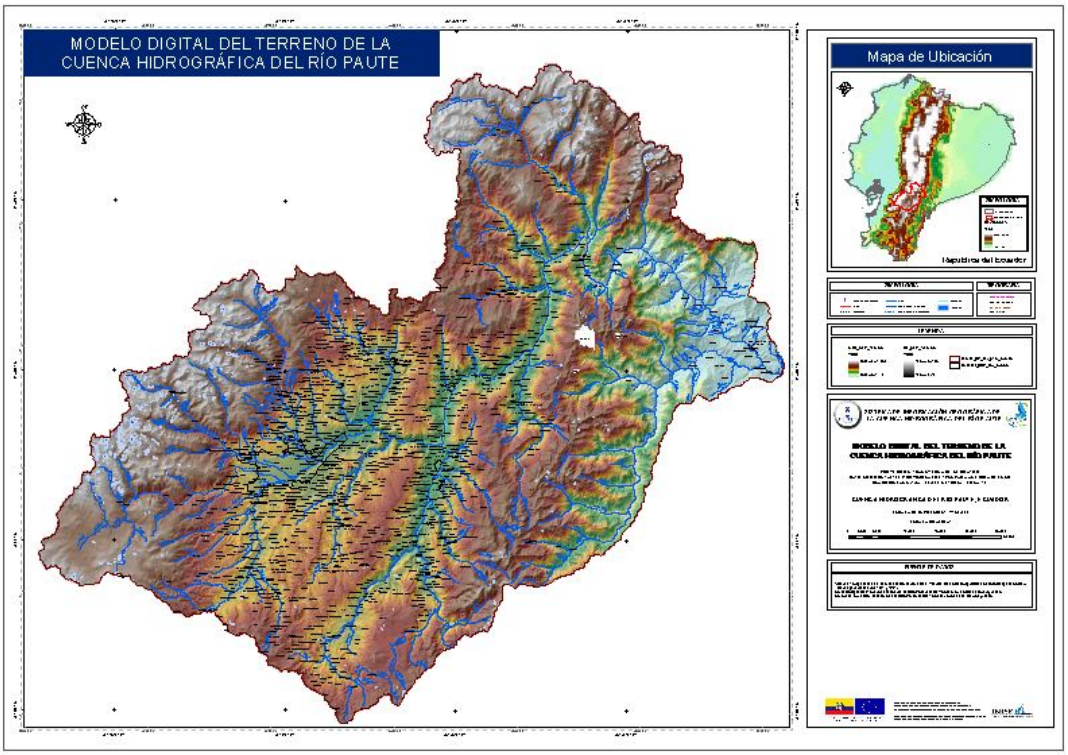
EDAFOLOGÍA: Regímenes de Humedad



EDAFOLOGÍA: Regímenes de Temperatura



MODELO DIGITAL DEL TERRENO



ANEXO II

MAPA DE ASPECTOS: ASP Aspectos.mxd		
FOLDER	**LAYER WMS**	**TABLE POSTGIS**
\GRAFICO\HIDROGRAFIA\Cuenca_RP_50k_UTM_SAM56.shp	cuenca_rp50k	cuenca_rp50k
\GRAFICO\DPA\Ciudades_CRP_50k_UTM_SAM56.shp	ciudades_25k	ciudades_25k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50K\Rios_CRP_50K_UTM_SAM56.shp	rios_crp50k	rios_crp50k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50k\Centros_Poblados_CRP_50K_UTM_SAM56.shp	centros_poblados50k	centros_poblados50k
\RASTER\asp_50k_sam56.img	asp_50k	n/a
\RASTER\ilu_50k_sam56.img	ilu_50k	n/a
\RASTER\mdt_50k_sam56.img	mdt_50k	n/a
MAPA DE PENDIENTES GRADOS: ASP Pendiente - Grados.mxd		
FOLDER	**LAYER WMS**	**TABLE POSTGIS**
\GRAFICO\HIDROGRAFIA\Cuenca_RP_50k_UTM_SAM56.shp	cuenca_rp50k	cuenca_rp50k
\GRAFICO\DPA\Ciudades_CRP_50k_UTM_SAM56.shp	ciudades_25k	ciudades_25k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50K\Rios_CRP_50K_UTM_SAM56.shp	rios_crp50k	rios_crp50k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50k\Centros_Poblados_CRP_50K_UTM_SAM56.shp	centros_poblados50k	centros_poblados50k
\RASTER\ilu_50k_sam56.img	ilu_50k	n/a
\RASTER\pgr_50k_sam56.img	pgr_50k	n/a
MAPA DE PENDIENTES PORCENTAJE: ASP Pendiente - Porcentaje.mxd		
FOLDER	LAYER WMS	TABLE POSTGIS
\GRAFICO\HIDROGRAFIA\Cuenca_RP_50k_UTM_SAM56.shp	cuenca_rp50k	cuenca_rp50k
\GRAFICO\DPA\Ciudades_CRP_50k_UTM_SAM56.shp	ciudades_25k	ciudades_25k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50K\Rios_CRP_50K_UTM_SAM56.shp	rios_crp50k	rios_crp50k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50k\Centros_Poblados_CRP_50K_UTM_SAM56.shp	centros_poblados50k	centros_poblados50k
\RASTER\ilu_50k_sam56.img	ilu_50k	n/a
\RASTER\ppo_50k_sam56.img	ppo_50k	n/a
MAPA DE CLIMA, PRECIPITACION MEDIA: CLI Precipitación.mxd		
FOLDER	LAYER WMS	TABLE POSTGIS
\GRAFICO\HIDROGRAFIA\Cuenca_RP_50k_UTM_SAM56.shp	cuenca_rp50k	cuenca_rp50k
\GRAFICO\DPA\Ciudades_CRP_50k_UTM_SAM56.shp	ciudades_25k	ciudades_25k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50K\Rios_CRP_50K_UTM_SAM56.shp	rios_crp50k	rios_crp50k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50k\Centros_Poblados_CRP_50K_UTM_SAM56.shp	centros_poblados50k	centros_poblados50k
\GRAFICO\HIDROGRAFIA\Estaciones_Meteorologicas_CRP_UTM_SAM56	est_met	est_met
\GRAFICO\CLIMA\pre_promd_s56	precipitación_mm	n/a
\RASTER\ilu_50k_sam56.img	ilu_50k	n/a
\RASTER\mdt_50k_sam56.img	mdt_50k	n/a
MAPA DE CLIMA, TEMPERATURA: CLI Temperatura.mxd		
FOLDER	LAYER WMS	TABLE POSTGIS
\GRAFICO\HIDROGRAFIA\Cuenca_RP_50k_UTM_SAM56.shp	cuenca_rp50k	cuenca_rp50k

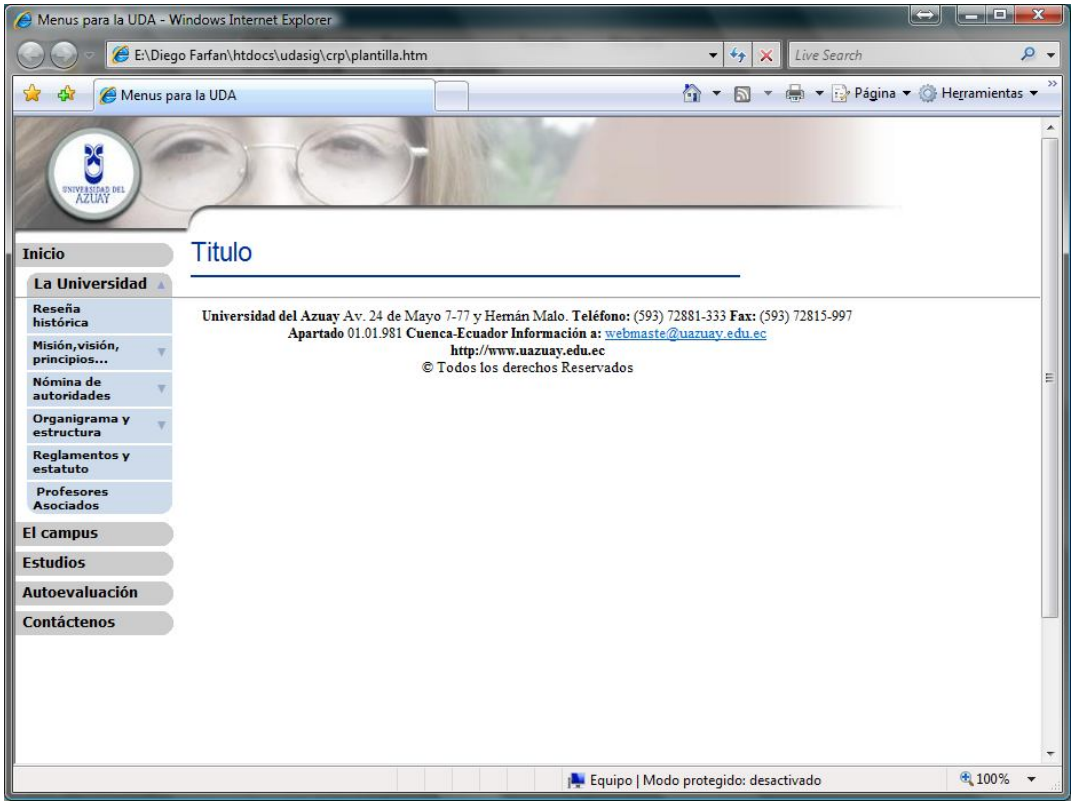
\GRAFICO\DPA\Ciudades_CRP_50k_UTM_SAM56.shp	ciudades_25k	ciudades_25k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50K\Rios_CRP_50K_UTM_SAM56.shp	rios_crp50k	rios_crp50k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50k\Centros_Poblados_CRP_50K_UTM_SAM56.shp	centros_poblados50k	centros_poblados50k
\GRAFICO\CLIMA\Estaciones_Registro_Temperatura_DIFORA_CRP_UTM_SAM56	est_temp	est_temp
\GRAFICO\CLIMA\temp_prmd_s56	temp_prmd_s56	n/a
\RASTER\ilu_50k_sam56.img	ilu_50k	n/a
MAPA DE COBERTURA DE SUELO, VEGETACIÓN LENOSA: COB Ecosistemas de Mayor Madurez - Vegetación Leñosa.mxd		
FOLDER	**LAYER WMS**	**TABLE POSTGIS**
\GRAFICO\HIDROGRAFIA\Cuenca_RP_50k_UTM_SAM56.shp	cuenca_rp50k	cuenca_rp50k
\GRAFICO\DPA\Ciudades_CRP_50k_UTM_SAM56.shp	ciudades_25k	ciudades_25k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50K\Rios_CRP_50K_UTM_SAM56.shp	rios_crp50k	rios_crp50k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50k\Centros_Poblados_CRP_50K_UTM_SAM56.shp	centros_poblados50k	centros_poblados50k
\GRAFICO\COBERTURA_SUELO\Ecosistemas_mayor_madurez_complejidad\Vegetación Leñosa_2001_amenazada_vias_centros_poblados_50k_UTM_SAM56.shp	veg_len	veg_len
\RASTER\ilu_50k_sam56.img	ilu_50k	n/a
MAPA DE COBERTURA DE SUELO, PARAMO AMENAZADO: COB Ecosistemas de Mayor Madurez - Páramo.mxd		
FOLDER	**LAYER WMS**	**TABLE POSTGIS**
\GRAFICO\HIDROGRAFIA\Cuenca_RP_50k_UTM_SAM56.shp	cuenca_rp50k	cuenca_rp50k
\GRAFICO\DPA\Ciudades_CRP_50k_UTM_SAM56.shp	ciudades_25k	ciudades_25k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50K\Rios_CRP_50K_UTM_SAM56.shp	rios_crp50k	rios_crp50k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50k\Centros_Poblados_CRP_50K_UTM_SAM56.shp	centros_poblados50k	centros_poblados50k
\GRAFICO\COBERTURA_SUELO\Ecosistemas_mayor_madurez_complejidad\Paramo_2001...	param_ame	param_ame
\RASTER\ilu_50k_sam56.img	ilu_50k	n/a
MAPA DE MODELO DIGITAL DEL TERRENO: MDT - Cuenca del Paute.mxd		
FOLDER	**LAYER WMS**	**TABLE POSTGIS**
\GRAFICO\HIDROGRAFIA\Cuenca_RP_50k_UTM_SAM56.shp	cuenca_rp50k	cuenca_rp50k
\GRAFICO\DPA\Ciudades_CRP_50k_UTM_SAM56.shp	ciudades_25k	ciudades_25k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50K\Rios_CRP_50K_UTM_SAM56.shp	rios_crp50k	rios_crp50k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50k\Centros_Poblados_CRP_50K_UTM_SAM56.shp	centros_poblados50k	centros_poblados50k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50k\Lagunas_CRP_50K_UTM_SAM56.shp	lagunas_50k	lagunas_50k
\RASTER\ilu_50k_sam56.img	ilu_50k	n/a
\RASTER\mdt_50k_sam56.img	mdt_50k	n/a
MAPA DE EDAFOLOGIA: EDA Regímenes de Humedad.mxd		
FOLDER	**LAYER WMS**	**TABLE POSTGIS**
\GRAFICO\HIDROGRAFIA\Cuenca_RP_50k_UTM_SAM56.shp	cuenca_rp50k	cuenca_rp50k
\GRAFICO\DPA\Ciudades_CRP_50k_UTM_SAM56.shp	ciudades_25k	ciudades_25k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50K\Rios_CRP_50K_UTM_SAM56.shp	rios_crp50k	rios_crp50k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50k\Centros_Poblados_CRP_50K_UTM_SAM56.shp	centros_poblados50k	centros_poblados50k
\GRAFICO\EDAFOLOGIA\Regimenes_Humedad_Suelo_CRP_200k_UTM_SAM56.shp	reg_hum	reg_hum

\RASTER\ilu_50k_sam56.img	ilu_50k	n/a

MAPA DE EDAFOLOGIA: EDA Regímenes de Temperatura del Suelo.mxd		
FOLDER	**LAYER WMS**	**TABLE POSTGIS**
\GRAFICO\HIDROGRAFIA\Cuenca_RP_50k_UTM_SAM56.shp	cuenca_rp50k	cuenca_rp50k
\GRAFICO\DPA\Ciudades_CRP_50k_UTM_SAM56.shp	ciudades_25k	ciudades_25k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50K\Rios_CRP_50K_UTM_SAM56.shp	rios_crp50k	rios_crp50k
\GRAFICO\CARTOGRAFIA_BASE\ESCALA_50k\Centros_Poblados_CRP_50K_UTM_SAM56.shp	centros_poblados50k	centros_poblados50k
\GRAFICO\EDAFOLOGIA\Regimenes_Temperatura_Suelo_CRP_200k_UTM_SAM56.shp	regim_temperatura	regim_temperatura
\RASTER\ilu_50k_sam56.img	ilu_50k	n/a

ANEXO III

PLANTILLA HTML



ANEXO IV

FUNCIONES OPENGIS[1]

1. AddGeometryColumn(varchar,varchar,varchar,integer,varchar,integer)

Sintaxis:

```
AddGeometryColumn(<nombre_db>,<nombre_tabla>,<nombre_columna>,<srid
>,<type>,<dimension>)
```

Añade una columna geométrica a una tabla existente. SRID debe ser un valor entero que referencia una valor en la tabla *SPATIAL_REF_SYS*. El tipo debe ser una cadena en mayúsculas que indica el tipo de geometría, como, '*POLYGON*' o '*MULTILINESTRING*'.

2. DropGeometryColumn(varchar,varchar,varchar)

```
Sintaxis:DropGeometryColumn(<nombre_db>,<nombre_Tabla>,<nombre_colum
na>)
```

Elimina una columna geométrica de una tabla espacial.

3. AsBinary(geometry)

Devuelve la geometría pasándola a formato *well-known-binary* de OGC, usando la codificación *endian* del servidor donde se ejecuta la base de datos.

4. Dimension(geometry)

Devuelve 2 si la geometría es de 2D y 3 si es 3D.

5. Envelope(geometry)

Retorna un *POLYGON* que representa la caja circunscrita de la geometría.

6. GeometryType(geometry)

Retorna el tipo de geometría como una cadena. Ejemplo:

'*LINESTRING*','*POLYGON*','*MULTIPOINT*',etc.

7. X(geometry)

Encuentra y devuelve la coordenada X del primer punto de *geometry*. Devuelve NULL si no hay puntos.

8. Y(geometry)

Encuentra y devuelve la coordenada Y del primer punto de *geometry*. Devuelve NULL si no hay puntos.

9. Z(geometry)

[1] (Tomado del Manual PostGIS de Paul Ramsey)

Encuentra y devuelve la coordenada Z del primer punto de geometry. Devuelve NULL si no hay puntos.

10. NumPoints(geometry)

Busca y devuelve el número de puntos del primer *linestring* en la *geometry*. Devuelve NULL sino hay *linestring.*

11. PointN(geometry,integer)

Devuelve el punto enésimo en el primer *linestring* de *geometry*. Y NULL si no hay ningún *linestring*.

12.EsxteriorRing(geometry)

Devuelve el circulo exterior del primer *polygon* en la *geometry*. Y nulo sino hay polígonos.

13.NumInteriorRings(geometry)

Devuelve el numero de círculos interiores del primer polígono de la geometría. Y NULL si no hay ningún polígono.

14. InteriorRingN(geometry,integer)

Devuelve el enésimo circulo interior del primer polígono en la *geometry*. Y NULL si no hay polígonos.

15.IsClosed(geometry)

Devuelve cierto si punto final = punto inicial de la geometría.

16. NumGeometries(geometry)

Si *geometry* es una *GEOMETRYCOLLECTION* devuelve el numero de geometrías que la componen. En caso contrario devuelve NULL.

17. Distance(geometry,geometry)

Devuelve la distancia cartesiana entre dos geometrías en unidades proyectadas.

18. AsText(geometry)

Devuelve una representación textual de una geometría. Ejemplo: POLYGON(0 0,0 1,1 1,1 0,0 0)

19. SRID(geometry)

Devuelve un numero entero que es el identificador del sistema de referencia espacial de una geometría.

20. GeometryFromText(varchar,integer)

Sintaxis: GeometryFromText(<geom>,<SRID>) convierte un objeto de la representación textual a un objeto geometría.

21.GeomFromText(varchar,integer)

Igual que *GeometryFromText* .

22.SetSRID(geometry)

Establece el valor del *SRID* de una geometría al entero dado. Usado en la construcción de cajas circunscritas para consultas.

23.EndPoint(geometry)

Devuelve un objeto punto que representa el ultimo punto en la geometría.

24.StartPoint(geometry)

Devuelve un objeto punto que representa el primer punto en la geometría.

24.Centroid(geometry)

Devuelve un punto que representa el *centroide* de la geometría.

Otras funciones:

1. A<&B: Devuelve verdadero si la caja que circunscribe a A superpone o está a la derecha de la de B.

2. A&>B: Devuelve verdadero si la caja que circunscribe a A superpone o está a la izquierda de la de B.

3. A<<B: Devuelve verdadero si la caja que circunscribe a A esta estrictamente a la derecha de la de B.

4. A>>B: Devuelve verdadero si la caja que circunscribe a A esta estrictamente a la izquierda de la de B.

5. A~=B: Es equivalente al operador "igual que". Compara las 2 geometrías característica por característica y si todas coinciden devuelve verdadero.

6. A~B: Devuelve verdadero si la caja circunscrita de A esta contendida en la de B.

7. A&&B: Si la caja circunscrita de A se superpone a la de B devuelve Verdadero.

8. area2d(geometry): Devuelve el área de una geometría si es un *POLYGON* o *MULTIPOLYGON*.

9. asbinary(geometry,'NDR'): Devuelve la geometría en el formato binario de OGC, usando codificacion littleendian.

Se usa para sacar datos de la bd sin convertirlos a texto.

10.asbinary(geometry,'XDR'): Devuelve la geometría en el formato binario de *OGC*, usando codificación *bigendian*. Se usa para sacar información de la base datos sin convertirla a texto.

11.box3d(geometry): Devuelve una *BOX3D* que representa la máxima extensión de la geometría.

12.collect(geometry): Devuelve un objeto *GEOMETRYCOLLECTION* a partir de un conjunto de geometrías. Es una función de Agregación en la terminología de PostgreSQL.

Ejemplo:

SELECT COLLECT(GEOM) FROM GEOTABLE GROUP BY ATTRCOLUMN

Devuelve una colección separada para cada valor distinto de *ARRTCOLUMN*.

13.distance_spheroid(point,point,spheroid): Devuelve la distancia lineal entre 2 latitud/longitud puntos dados de un spheroid.

14.extent(geometry): Es una función de Agregación en la terminología de PostgreSQL. Trabaja sobre una lista de datos, de la misma manera que las funciones sum() y mean().

Ejemplo:

SELECT EXTENT(GEOM) FROM GEOMTABLE

Devuelve una BOX3D dando la máxima extensión a todas las características de la tabla.

SELECT EXTENT(GEOM) FROM GEOMTABLE GROUP BY CATEGORY

Devuelve un resultado extendido para cada categoria.

15. find_srid(varchar,varchar,varchar): Sintaxis:

find_srid(<db/esquema>,<tabla>,<columna>)

Devuelve el *SRID* de una columna dada buscando esta en la tabla *GEOMETRY_COLUMNS*. Si la columna geométrica no ha sido añadida con la función AddGeometryColumns() no funcionará.

16.force_collection(geometry): Convierte una geometría en una *GEOMETRYCOLLECTION*. Esto se hace para simplificar la representación WKB .

17. force_2d(geometry): Fuerza la geometría a 2D así la representación de salida tendrá solo las coordenadas X e Y. Se usa porque OGC solo especifica geometrías 2D.

18.force_3d(geometry): Fuerza la geometría a 3D. Así la todas las representaciones tendrán 3 coordenadas X,Y y Z.

19.length2d(geometry): Devuelve la longitud 3d de la geometría si es una *linestring* o *multilinestring*.

20.length3d(geometry): Devuelve la longitud 2d de la geometría si es una *linestring* o *multilinestring*.

21.length_spheroid(geometry,spheroid): Calcula la longitud de una geometría en un elipsoide. Se usa si las coordenadas de la geometría están en latitud/longitud. El elipsoide es un tipo separado de la base de datos y se puede construir como:

SPHEROID [<NAME>,<SEMI-MAJOR AXIS>,<INVERSE FLTTENING>]

Ejemplo:

SPHEROID [“GRS_1980”,6378137, 298,257222101]

Ejemplo de cálculo:

SELECT

LENGTH_SPHEROID(geometry_column,'SPHEROID[“GRS_1980”,6378137,298.257222101]')FROM geometry_table;

22. length3d_spheroid(geometry,spheroid)

Calcula la longitud de una geometría en un elipsoide, teniendo en cuenta la elevación.

23. max_distance(linestring,linestring)

Devuelve la distancia más larga entre dos *linestring*.

24. mem_size(geometry)

Retorna el tamaño en bytes de la geometría.

25. npoints(geometry)

Devuelve el número de puntos en la geometría.

26. nrings(geometry)

Si la geometría es un polígono o multipolígono devuelve el numero de círculos de la geometría.

27. num_sub_objects(geometry)

Devuelve el número de objetos almacenados en la geometría. Se usa en Multigeometrías y *GEOMETRYCOLLECTIONs*.

28. perimeter2d(geometry)

Devuelve el perímetro 2d de la geometría, si esa geometría es un polígono o un multipolígono.

29. perimeter3d(geometry)

Devuelve el perímetro 3d de la geometría, si esa geometría es un polígono o un multipolígono.

30.point_inside_circle(geometry,float,float,float)

Sintaxis: point_inside_circle(<geometry>,<circle_center_x>,<circle_center_y>,<radius>)

Devuelve verdadero si la geometría es un punto y está dentro del circulo.

31. postgis_version()

Devuelve la versión de las funciones postgis instaladas en la bases de datos.

32. summary(geometry)

Devuelve un resumen en texto del contenido de esa geometría.

33. transform(geometry,integer)

Devuelve una nueva geometría con sus coordenadas transformadas la SRID dadapor el parámetro integer. SRID debe existir en la tabla SPATIAL_REF_SYS.

34. translate(geometry,float8,float8,float8)

Traslada la geometría a la nueva localización usando los valores pasados como desplazamientos X,Y,Z.

35. truly_inside(geometryA,geometryB)

Devuelve verdadero si alguna parte de B esta dentro de la caja circunscrita de A.

36. xmin(box3d) ymin(box3d) xmin(box3d)

Devuelve la mínima coordenada de la caja circunscrita.

37. xmax(box3d) ymax(box3d) zmax(box3d)

Devuelve la máxima coordenada de la caja circunscrita.

ANEXO V

ARCHIVOS .HTML FINALES

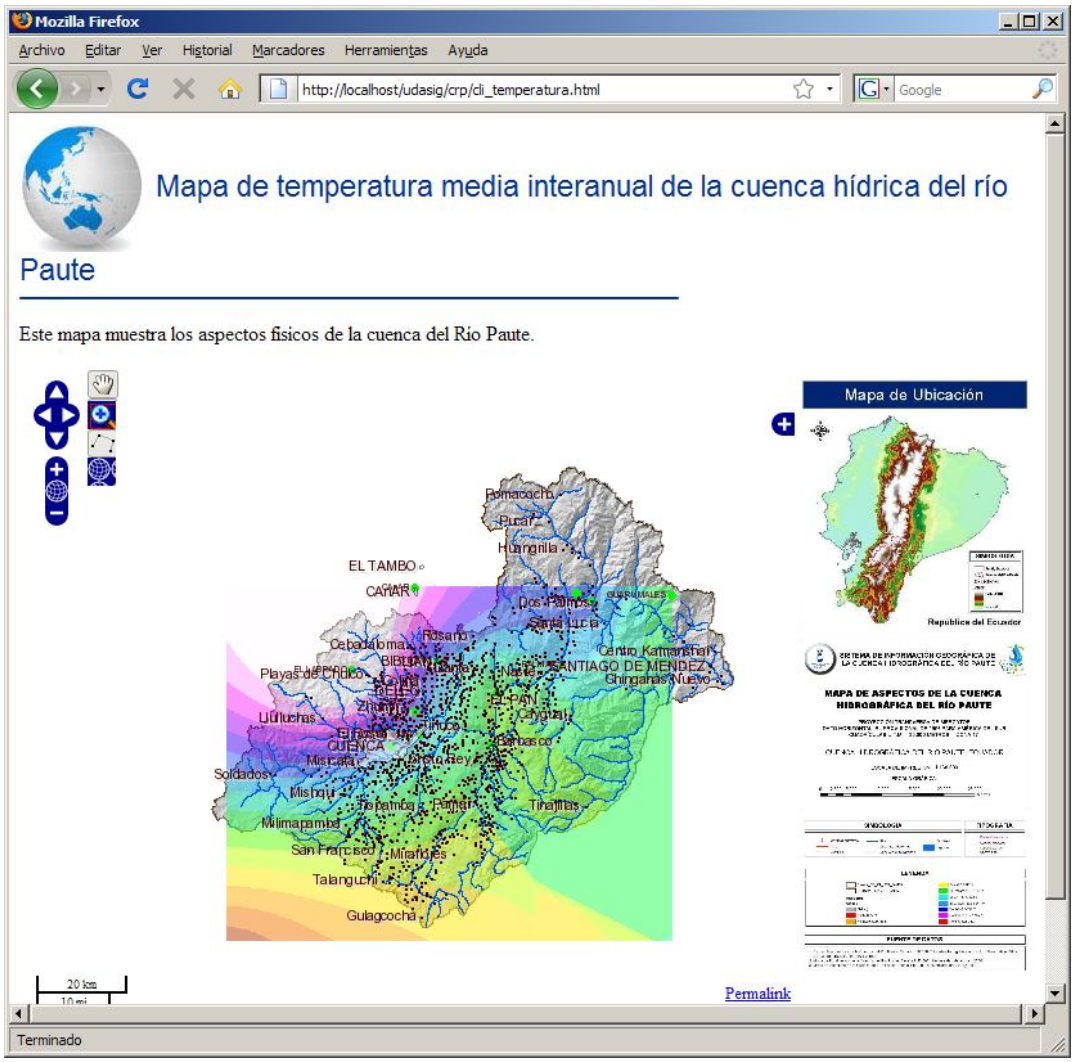
MAPA DE ASPECTOS

MAPA DE PENDIENTES EN GRADOS

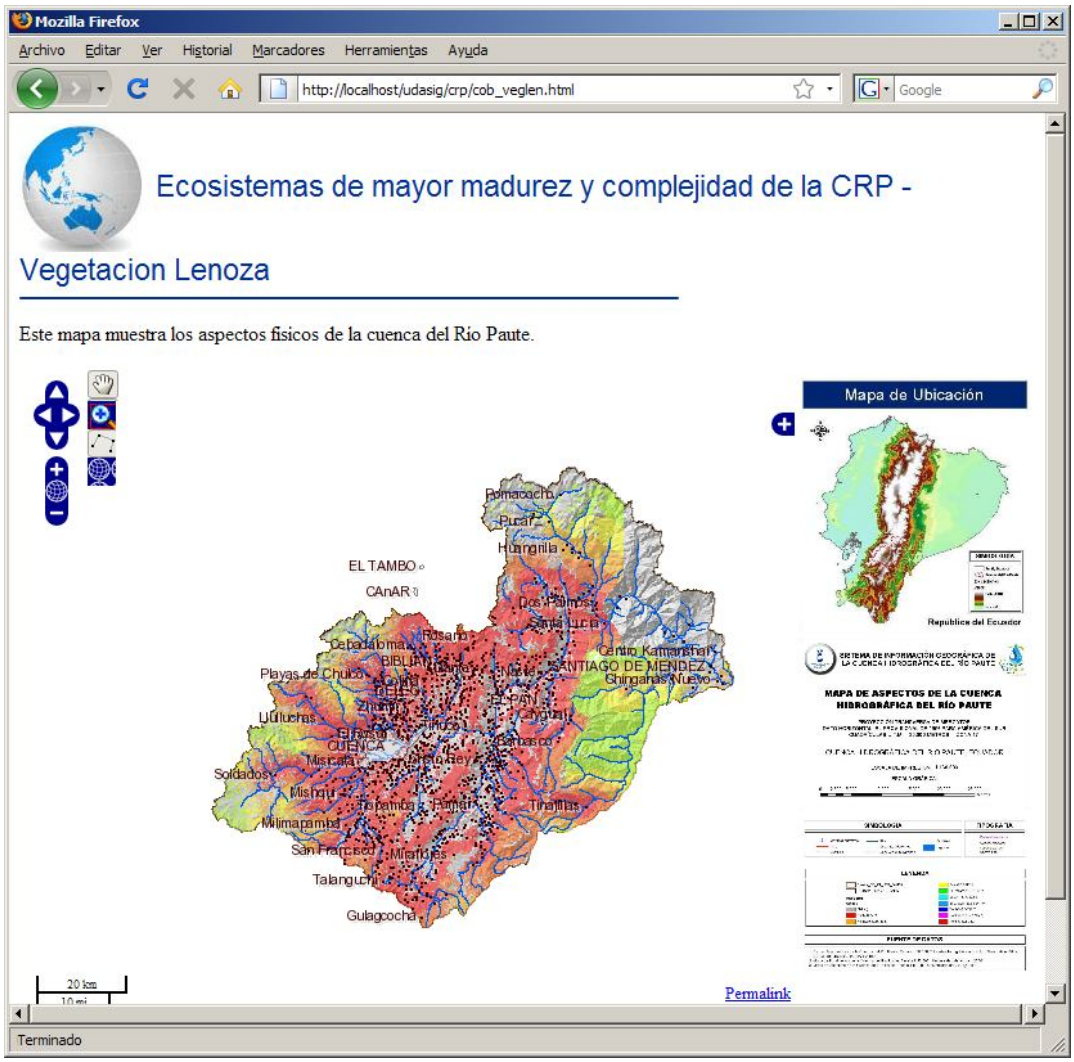
MAPA DE PENDIENTES EN PORCENTAJE

MAPA DE PRECIPITACION MEDIA INTERANUAL DE LA CUENCA HÍDRICA DEL RÍO PAUTE

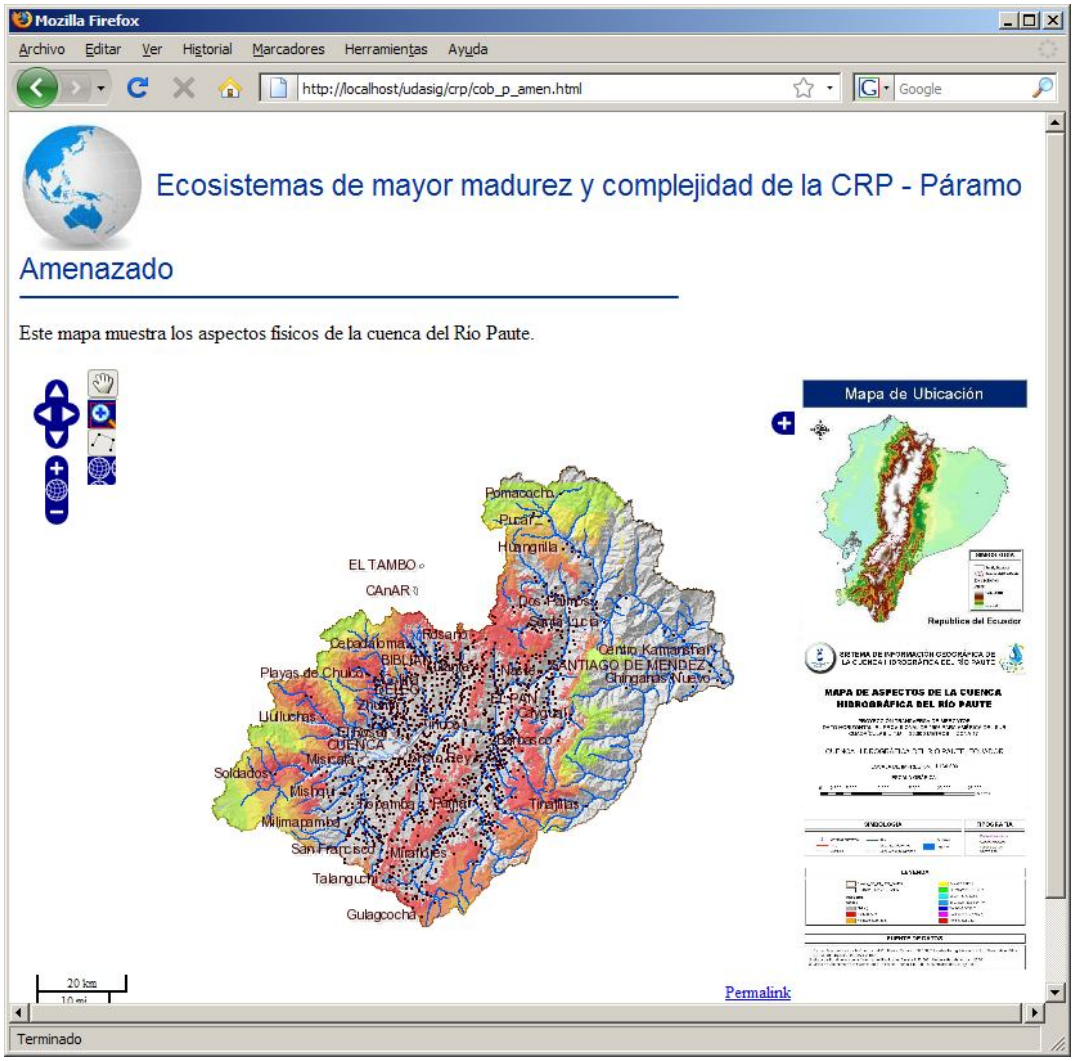
MAPA DE TEMPERATURA MEDIA INTERANUAL DE LA CUENCA HÍDRICA DEL RÍO PAUTE



ECOSISTEMAS DE MAYOR MADUREZ Y COMPLEJIDAD DE LA CRP - VEGETACION LENOZA

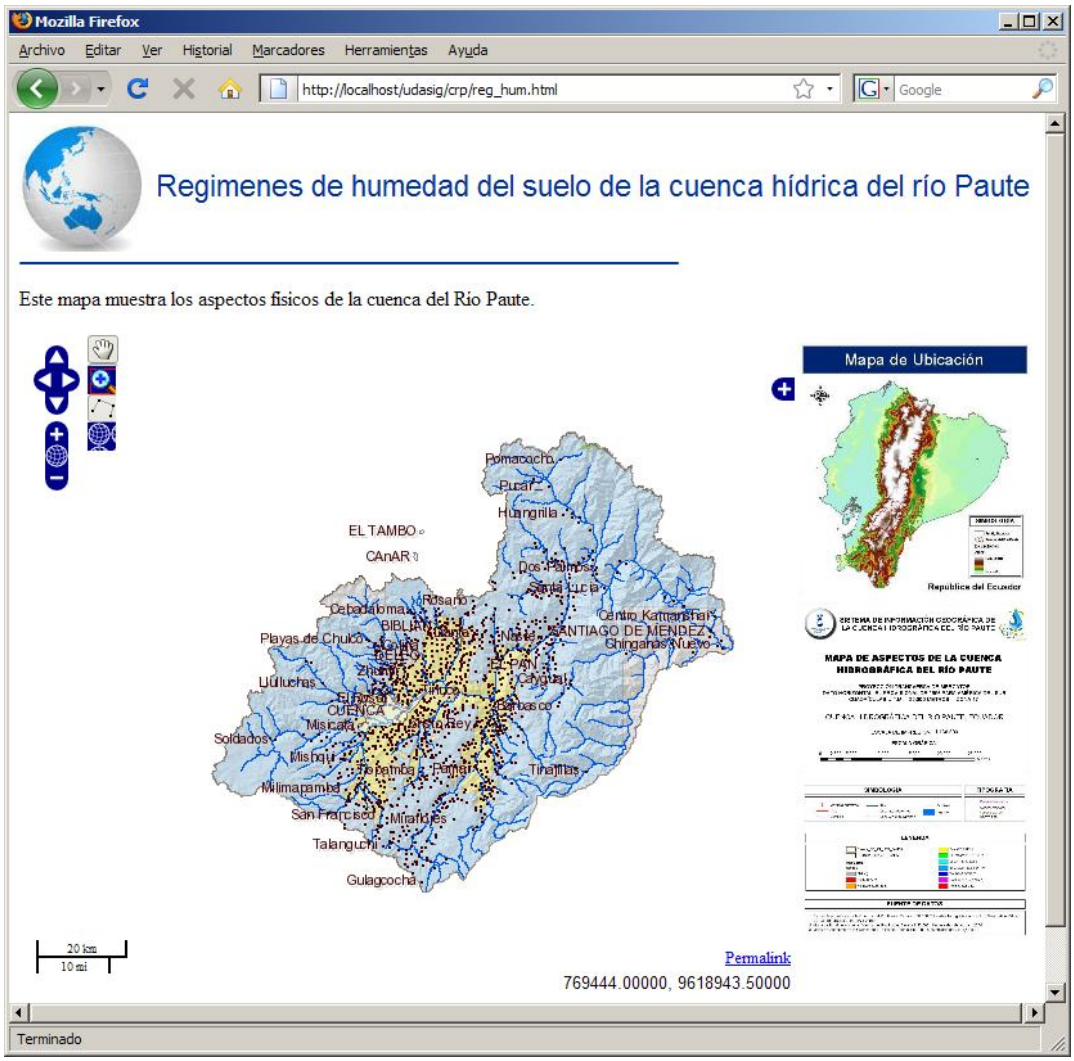


ECOSISTEMAS DE MAYOR MADUREZ Y COMPLEJIDAD DE LA CRP - PÁRAMO AMENAZADO



MODELO DIGITAL DEL TERRENO DE LA CUENCA HÍDRICA DEL RÍO PAUTE

REGIMENES DE HUMEDAD DEL SUELO DE LA CUENCA HÍDRICA DEL RÍO PAUTE



REGÍMENES DE TEMPERATURA DEL SUELO DE LA CUENCA HÍDRICA DEL RÍO PAUTE

ANEXO VI

A continuación se muestra el encabezado del mapfile utilizado en la publicación de mapas temáticos de la cuenca del Río Paute.

```
MAP
       NAME tematicos
       STATUS ON
       SIZE 800 400
       EXTENT 678106.8561 9633526.9535 782805.8241 9747053.0365
       UNITS DD
       SHAPEPATH "data"
       IMAGETYPE PNG
       FONTSET "./font/fonts.txt"
       SYMBOL
              NAME 'circle'
              TYPE ELLIPSE
              POINTS 5 5 END
              FILLED TRUE
       END
       SYMBOL
              NAME 'estacion'
              TYPE ELLIPSE
              POINTS 10 10 END
              FILLED TRUE
       END
       PROJECTION
              "init=epsg:24877"
       END
       SCALEBAR
              STATUS on
              POSITION lc
              STYLE 0
              INTERVALS 3
              SIZE 129 3
              IMAGECOLOR  255 255 255
              LABEL
                     COLOR 0 0 0
                     SIZE 1
              END  # end label
              COLOR  0 0 0
              UNITS kilometers
       END   # end scalebar

       OUTPUTFORMAT
              NAME "png"
              MIMETYPE "image/png"
              DRIVER "GD/PNG"
              EXTENSION "png"
              IMAGEMODE RGB  #PC256
              TRANSPARENT true
       END #end outputformat
```

El siguiente es un fragmento de código que maneja la capa ‘’cuenca_rp50k”, que corresponde a un archivo .shp de ESRI.

```
LAYER
      CONNECTIONTYPE postgis
      CONNECTION "host=localhost
                  dbname=monografia
                  user=postgres password=postgres
                  port=5432"
            DATA "the_geom from cuenca_rp50k"
      NAME 'cuenca_rp50k'
      PROJECTION
            "init=epsg:24877"
      END #end projection
      TYPE polygon
      STATUS ON
      TOLERANCE 8 #default is 3 for raster. 0 for vector
      TEMPLATE "query.html"
            CLASS
                  COLOR 255 255 255
                  OUTLINECOLOR 128 64 0
                  SIZE 2
            END #end CLASS
END #end layer
```

Este en cambio maneja los centros_poblados50k

```
LAYER
      CONNECTIONTYPE postgis
      CONNECTION   "host=localhost
                  dbname=monografia
                  user=postgres
                  password=postgres
                  port=5432”
      DATA "the_geom from centros_poblados50k"
      NAME 'centros_poblados50k'
      PROJECTION
            "init=epsg:24877"
      END #end projection
      TYPE point
      STATUS ON
      TOLERANCE 8 #default is 3 for raster. 0 for vector
      TEMPLATE "query.html"
            CLASS
                  SYMBOL 'circle'
                  #ANTIALIAS false # not sure about this one here!
                  COLOR 64 0 0
                  SIZE 2
            END #end style
      GROUP "centros"
END #end layer
```

Y las respectivas etiquetas:

```
LAYER # States label layer begins here
      CONNECTIONTYPE postgis
      CONNECTION "host=localhost
                  dbname=monografia
                  user=postgres
                  password=postgres
                  port=5432"
      DATA "the_geom from centros_poblados50k"
      NAME         'nom_centros'
      STATUS       ON
      TYPE         ANNOTATION
      CLASSITEM    "nombre"
      LABELITEM    "nombre"
      CLASS
      # EXPRESSION 'nombre'
      STYLE
        COLOR      -1 -1 -1
      END
      LABEL
        COLOR 64 0 0
             SHADOWCOLOR 218 218 218
        SHADOWSIZE 2 2
        TYPE TRUETYPE
        FONT arial
        SIZE 8
        ANTIALIAS TRUE
        POSITION CL
        PARTIALS FALSE
        MINDISTANCE 300
        BUFFER 4
      END # end of label
    END # end of class
      GROUP "centros"
END # States label layer ends here
```

A continuación un layer con fuente raster:

```
LAYER
                NAME 'asp_50k'
                GROUP 'Aspectos'
                DATA '/opt/fgs/www/htdocs/cursoIDE/data/raster/asp_50k_sam56.img'
                PROJECTION
                        "init=epsg:24877"
                END #end projection

                #TRANSPARENCY 50
                TYPE raster
                OFFSITE 255 255 255 # transparency color for raster layer
                STATUS ON
                TOLERANCE 8 #default is 3 for raster. 0 for vector

                CLASS
                        NAME 'Flat (-1)'
                        EXPRESSION ([pixel] >= -1 AND [pixel] < 0)
                        COLOR 176 176 176
                END #end class
                CLASS
                        NAME 'North (0-22.5)'
                        EXPRESSION ([pixel] >= 0 AND [pixel] < 22.5)
                        COLOR 254 255 255
                END #end class
                CLASS
                        NAME 'Northeast (22.5-67.5)'
                        EXPRESSION ([pixel] >= 22.5 AND [pixel] < 67.5)
                        COLOR 255 166 0
                END #end class
                CLASS
                        NAME 'East (67.5-112.5)'
                        EXPRESSION ([pixel] >= 67.5 AND [pixel] < 112.5)
                        COLOR 255 255 0
                END #end class
                CLASS
                        NAME 'Southeast (112.5-157.5)'
                        EXPRESSION ([pixel] >= 112.5 AND [pixel] < 157.5)
                        COLOR 0 255 0
                END #end class
                CLASS
                        NAME 'South (157.5-202.5)'
                        EXPRESSION ([pixel] >= 157.5 AND [pixel] < 202.5)
                        COLOR 0 255 255
                END #end class
                CLASS
                        NAME 'Southwest (202.5-247.5)'
                        EXPRESSION ([pixel] >= 202.5 AND [pixel] < 247.5)
                        COLOR 0 166 255
                END #end class
                CLASS
                        NAME 'West(247.5-292.5)'
                        EXPRESSION ([pixel] >= 247.5 AND [pixel] < 292.5)
                        COLOR 0 0 255
                END #end class
                CLASS
                        NAME 'Northwest (292.5-337.5)'
                        EXPRESSION ([pixel] >= 292.5 AND [pixel] < 337.5)
                        COLOR 255 0 255
                END #end class
                CLASS
                        NAME 'North (337.5-360)'
                        EXPRESSION ([pixel] >= 337.5 AND [pixel] < 359.999969482422)
                        COLOR 254 255 255
                END #end class
                CLASS
                        NAME 'Other'
                        COLOR 255 255 255
                END #end class
END #end layer
```