

## GIS 서비스 개발자 입문



2018 공간정보아카데미 교육과정

# 목차

---

<b>1. 이 과정에서 무엇을 배울 수 있는가?</b>	<b>2</b>
1x01 수업 목표 공유	2
<b>2. 오픈소스 공간정보 서비스 스택 이해와 설치</b>	<b>3</b>
2x01 오픈소스 소프트웨어란 무엇인가?	3
2x02 오픈소스 SW 라이선스 종류와 특징	6
2x03 FOSS4G Stack 구성 이해	8
2x04 실습에 필요한 SW 설치	10
<b>3. 공간데이터 다루기</b>	<b>13</b>
3x01 세계지도 데이터 받기	13
3x02 국토지리정보원 데이터 받기	14
3x03 지적도 등 국가공간정보포털 자료 받기	18
3x04 기타 쓸만한 데이터 확보처	20
3x05 한글 코드페이지 문제 해결	23
3x06 좌표계 문제 해결	26
3x07 온맵과 GeoPackage를 통해 보는 발전방향	33
<b>4. PostGIS와 공간 SQL</b>	<b>37</b>
4x01 PostgreSQL이 공간정보를 다룰 수 있게 준비하기	37
4x02 PostGIS에 공간정보 올리기	43
4x03 공간 SQL 맛보기	49
4x04 공간 SQL 실습	56
4x05 공간자료를 효과적으로 다루는 커맨드라인 명령어	59
<b>5. GeoServer를 통한 공간정보 서비스</b>	<b>64</b>
5x01 GeoServer에 레이어 등록	64
5x02 OGC 웹서비스 이해	73
5x03 레이어 그룹과 스타일	78
5x04 uDig을 이용한 순쉬운 스타일링	86
5x05 GWC를 이용한 캐시	95
<b>6. OpenLayers를 이용한 웹 GIS</b>	<b>102</b>
6x01 OpenLayers 소개	102
6x02 서비스를 위한 Stack 구성	105
6x03 최종 목표 사이트 개발	108
6x04 개발 소스	114

## 1x01 수업 목표 공유

### ● 본 수업의 최종 목표

The True Size of... 사이트와 유사한 사이트를 처음부터 끝까지 만들어 보는 것입니다.

<https://thetruesize.com>



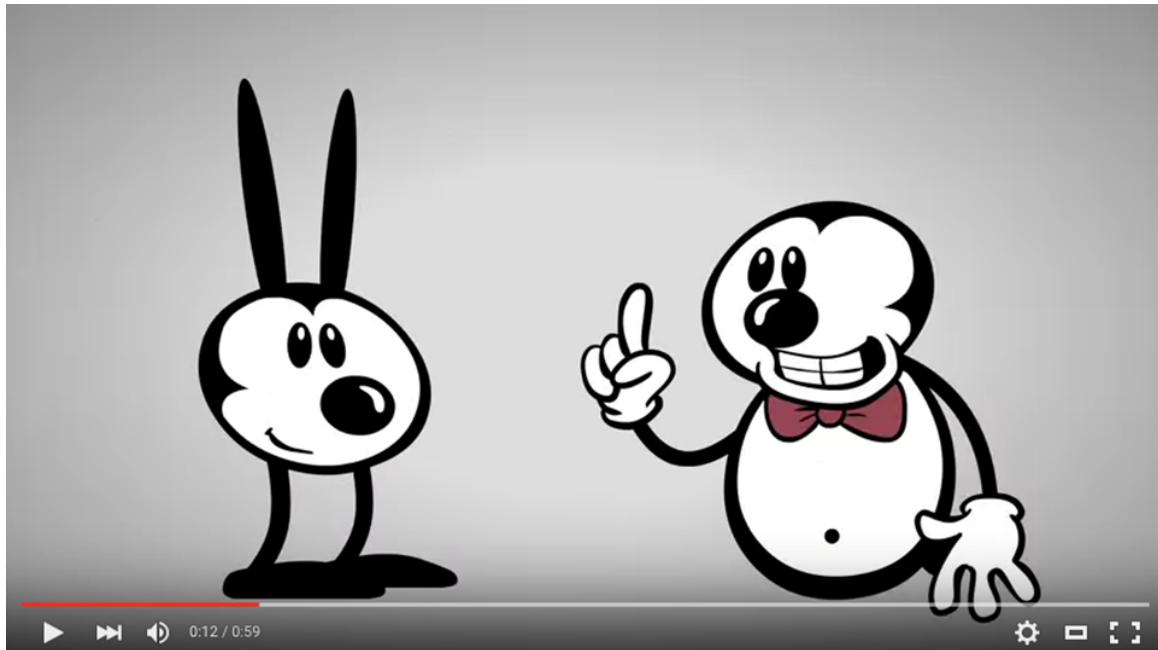
이런 사이트를 만들기 위해 아래의 많은 내용들을 공부합니다.

- 1일차
  - 오픈소스 GIS
  - 공간데이터 다루기
  - 공간 DBMS
- 2일차
  - OWS 서비스
  - 웹 GIS
  - 최종 결과물 만들기

이렇게 데이터부터 웹까지, 공간정보 서비스에 필요한 내용을 사례중심으로 전부 훑어 보는 것이 본 과정의 목표입니다. 추가로 자세한 기술적인 정보가 필요하신 분은 심화과정을 들어주세요.

## 2x01 오픈소스 소프트웨어란 무엇인가?

먼저 충격적인 동영상 하나 보고 수업을 시작하겠습니다.



링크: <https://www.youtube.com/watch?v=leTybKL1pM4>

오픈소스 운동가들은 소프트웨어를 인간세상을 발전시킬 수 있는 문화유산으로 생각하고 누구나 사용할 수 있는 공공재로 만들려 합니다.

이 철학에 기대어 제가 가장 좋아 하는 오픈소스는 '훈민정음' 입니다.



출처: [https://i.ytimg.com/vi/kBDv\\_XhgsGA/maxresdefault.jpg](https://i.ytimg.com/vi/kBDv_XhgsGA/maxresdefault.jpg)

누가 만들었는지도 분명하고, 많은 노력을 들여 만들었지만, 누구나 무료로 사용할 수 있게 공개되었고, 이를 바탕으로 많은 문화가 발전했으며, 지금 이 강의도 진행할 수 있습니다.

오픈소스 소프트웨어는 흔히 **FOSS** (Free & Open Source Software)라 불립니다.  
이 중 공간정보 오픈소스 소프트웨어는 **FOSS4G** (FOSS for Geospatial)라고 구별해 부르기도 합니다.

FOSS는 보통 다음과 같은 특징이 있습니다.

- 특정 라이선스에 따라 소프트웨어의 소스코드가 공개되어 있음
- 일반적으로 FOSS 사용자는 소프트웨어에 대한 자유로운 이용, 복사, 수정 및 재배포의 권한을 부여받음
- FOSS를 사용해 발생하는 문제는 저작자가 아닌 사용자의 책임임
- FOSS를 사용해 생산한 데이터 등은 사용자에게 권한이 있음
- FOSS의 Free는 '공짜'를 의미하는 것이 아니라, 사용자가 소스코드에 접근하고 프로그램을 사용, 수정, 재배포 할 수 있는 '자유'를 의미함
- FOSS는 개방형 표준(Open Standard)과는 다른 의미임. 하지만 일반적으로 FOSS는 국제적인 표준을 따르는 경향이 있음 (OGC와 OSGeo는 다른 조직)

오픈소스 소프트웨어의 자유는 다음과 같은 의미를 지닙니다.

- Freedom of Use
- Freedom of Copy
- Freedom of Modify
- Freedom of Redistribute

오픈소스 소프트웨어에 대한 오해들을 살펴봅시다.

- 오픈소스는 공짜?



- 오픈 소스 자체는 공짜지만,
- 오픈 소스를 제공하는 서비스(활용 및 개발, 유지보수 등)는 공짜가 아님.
- 초콜릿 만드는 레시피가 공짜라고 초콜릿도 공짜는 아님!
- 심지어 오픈소스를 돈받고 팔고 이를 보증하는 회사도 많음

- 오픈소스는 외산?



- 외산이라기보다는 지구산이라는 표현이 더 정확할 듯.
- 국내 개발자들의 참여도 많이 늘어나고 있고,
- 오픈 소스를 활용하여 국내에서 개발한 서비스들도 늘어나고 있음.
- 국산 소프트웨어라도 개발도구들은 외산이 주를 이룸(Eclipse, Microsoft Visual Studio)

- 오픈소스는 품질이 낮다?



**Low Quality**

- 과거에는 오픈 소스의 품질이 떨어지는 경우가 있었으나,
- 요즘은 많은 훌륭한 개발자들이 오픈 소스에 헌신을 하며 개발을 주도하고 있고,
- 다양한 분야에서 전문적인 오픈 소스 소프트웨어들이 상용 소프트웨어 못지 않은 품질로 출시되고 있음

- 오픈소스는 보안에 취약하다?



- 소스 코드가 공개되어 있다고 시스템 자체의 보안이 취약해지지는 않음.
- 보안이 중요한 미국 국방부에서도 오픈 소스를 활발히 사용하고 있고,
- 시스템의 보안은 소스 코드 공개의 유무보다는 하드웨어나 소프트웨어적으로 방화벽이나 보안 코드(패킷 등의 암호화)의 사용 여부 등이 더욱 중요함

- 오픈소스는 사용자 지원이 나쁘다?



- 서비스의 품질은 대체적으로 서비스 대가와 비례.
- RedHat등은 오픈 소스 제품을 유료로 판매하면서 고객들에게 상당히 높은 수준의 서비스를 제공함.
- 상용 소프트웨어 제품들도 서비스가 나쁜 경우가 종종 있음.

- 오픈소스를 사용하고 있다는 사실은 숨겨야 한다?



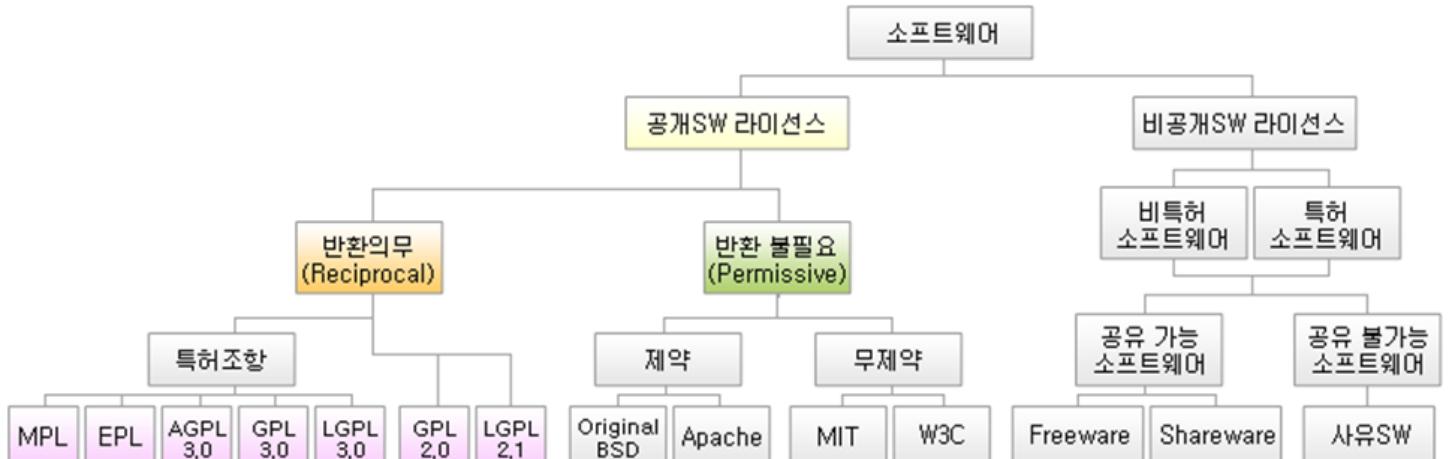
- 우리 개발자들이 해결해야 할 문제!!!

## 2x02 오픈소스 SW 라이선스 종류와 특징

라이선스(license, 사용허가권)는 저작권(Copyright)과는 다른, 저작자가 저작물을 사용하는 사람들에게 특정 목적 혹은 방식으로만 사용하게 제한하는 것입니다.

소프트웨어 라이선스는 생각보다 다양하고 각 라이선스 별로 특징과 주의점이 있습니다.

### ○ SW 라이선스 분류



출처: [http://www.oss.kr/oss/images/intro/license\\_02.gif](http://www.oss.kr/oss/images/intro/license_02.gif)

- 소스코드를 공개하지 않는 소프트웨어들은 '비공개SW 라이선스'이므로 본 강의에서는 다루지 않습니다.
- 오픈소스 소프트웨어는 반드시 앞에서도 살펴본 4가지 자유인 자유로운 이용, 복사, 수정 및 재배포의 권한을 만족해야 합니다만... 조건에 따라 제한이 있을 수도 있습니다.
- 먼저 소스코드를 고친 경우 이를 공개할 의무가 '반환의무'입니다.
- 또한 반환의무가 있는 경우에도 배타적 특허를 주장할 수까지 없는 경우도 있습니다.
- 사용형태가 서비스인지, 라이브러리인지, 소스코드를 수정해 사용하는지에 따라 제한이 달라지기도 합니다.

제한을 중심으로 주요 오픈소스 라이선스를 구분해 보면 다음과 같습니다.

- 거의 아무런 제한이 없는 라이선스
  - MIT, W3C 라이선스
  - 라이선스 안내문구를 포함해 배포하여야 함
- 고쳐서 배포해도 소스코드 반환의무 없는 라이선스
  - BSD, Apache 라이선스
  - 원 소스를 고쳐 공여하는 것은 무조건 원 라이선스를 따름
- 라이브러리로 사용시 반환의무 없는 라이선스
  - LGPL, MPL, EPL 라이선스
  - 라이브러리를 바이너리로 링크하여 사용시 소스공개 안해도 됨
  - 라이브러리 소스를 수정해 사용한 경우 라이브러리 소스만 공개하면 됨
- 라이브러리로 사용하거나 소스 수정시 반환의무 있는 라이선스
  - GPL 라이선스
  - 수정하지 않고 그대로 소프트웨어로 사용시 반환의무 없음
  - 라이브러리로 링크해 프로그램 개발시에도 **전체 소스** 반환의무 발생
  - 원 소스 수정해 배포시에도 반환의무 발생
  - 원 소스 수정하거나 이용해도 배포하지 않고 서비스로 이용시 반환의무 없음
- 서비스로 사용해도 반환의무 있는 라이선스
  - AGPL 라이선스
  - 서비스로 혹은 서버로 사용시에도 반환의무 발생

- 이 경우도 그냥 배포 안하고 내부적으로 사용시 반환의무 없음

상세한 내용은 오픈소스 라이선스 종합정보시스템 사이트를 참고하세요.  
<https://olis.or.kr/license/licenseClassification.do?mapcode=010001&page=1>

우리가 많이 사용하는 오픈소스 소프트웨어들의 라이선스를 확인해 봅시다.

- **Linux**
  - GPL 라이선스
- **PostgreSQL**
  - BSD 라이선스
- **PostGIS**
  - GPL2 라이선스
- **GeoServer**
  - GPL 라이선스
- **GeoTools**
  - LGPL 라이선스
- **QGIS**
  - GPL2 라이선스
- **GDAL**
  - MIT 라이선스
- **uDig**
  - EPL 라이선스
- **OpenLayers**
  - BSD 라이선스
- **Apache Web Server (HTTPD)**
  - Apache 라이선스

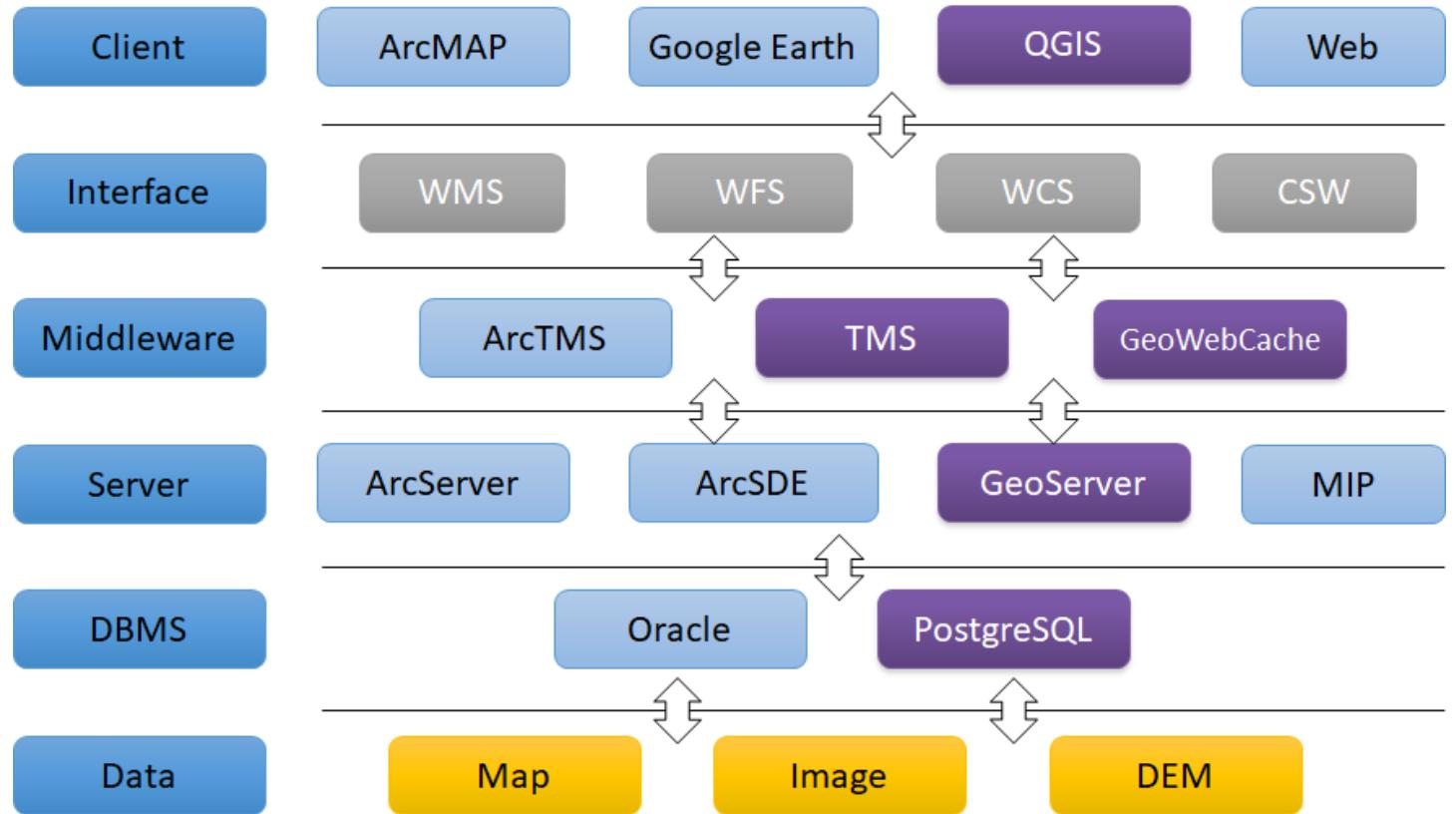
주요 공개SW 라이선스 비교

구분	무료 이용가능	배포 허용가능	소스코드 취득가능	소스코드 수정가능	2차적 저작물 재공개 의무	독점SW와 결합가능
GPL	●	●	●	●	●	X
LGPL	●	●	●	●	●	●
MPL	●	●	●	●	●	●
BSD license	●	●	●	●	X	●
Apache license	●	●	●	●	X	●

출처: [https://www.oss.kr/oss\\_license](https://www.oss.kr/oss_license)

## 2x03 FOSS4G Stack 구성 이해

GIS 서비스를 위해서는 보통 여러 계층의 여러 소프트웨어가 연결되어 구성됩니다. 이를 Stack이라 부릅니다.



위의 소프트웨어 혹은 인터페이스들 중 보통 사용자가 각 계층에서 선택하여 서비스 구성이 가능합니다.

독점적 상용 소프트웨어 만으로 서비스가 구성되기도 하고, 오픈소스 소프트웨어 만으로 서비스가 구성되기도 하고, 때로는 오픈소스와 독점 소프트웨어가 섞여 구성되기도 합니다.

각 소프트웨어는 완전히 독립적이지만, 네트워크와 이를 통한 인터페이스로 연결되어 서비스가 구성됩니다. 이런 연결은 보통 OGC라는 공간정보표준 국제단체에서 정의한 표준 인터페이스를 통해 가능하게 됩니다. 과거에는 ArcGIS 시리즈 등 독점 소프트웨어들은 이런 국제표준을 준수하지 않고 독자적인 인터페이스를 제공했지만, 이제는 대부분 국제표준 인터페이스도 지원하는 방향으로 발전되었습니다. 오픈소스의 경우에는 혼자 개발할 수 없는 경우가 일반적이라 표준인터페이스를 잘 준수하는 경우가 대부분입니다.

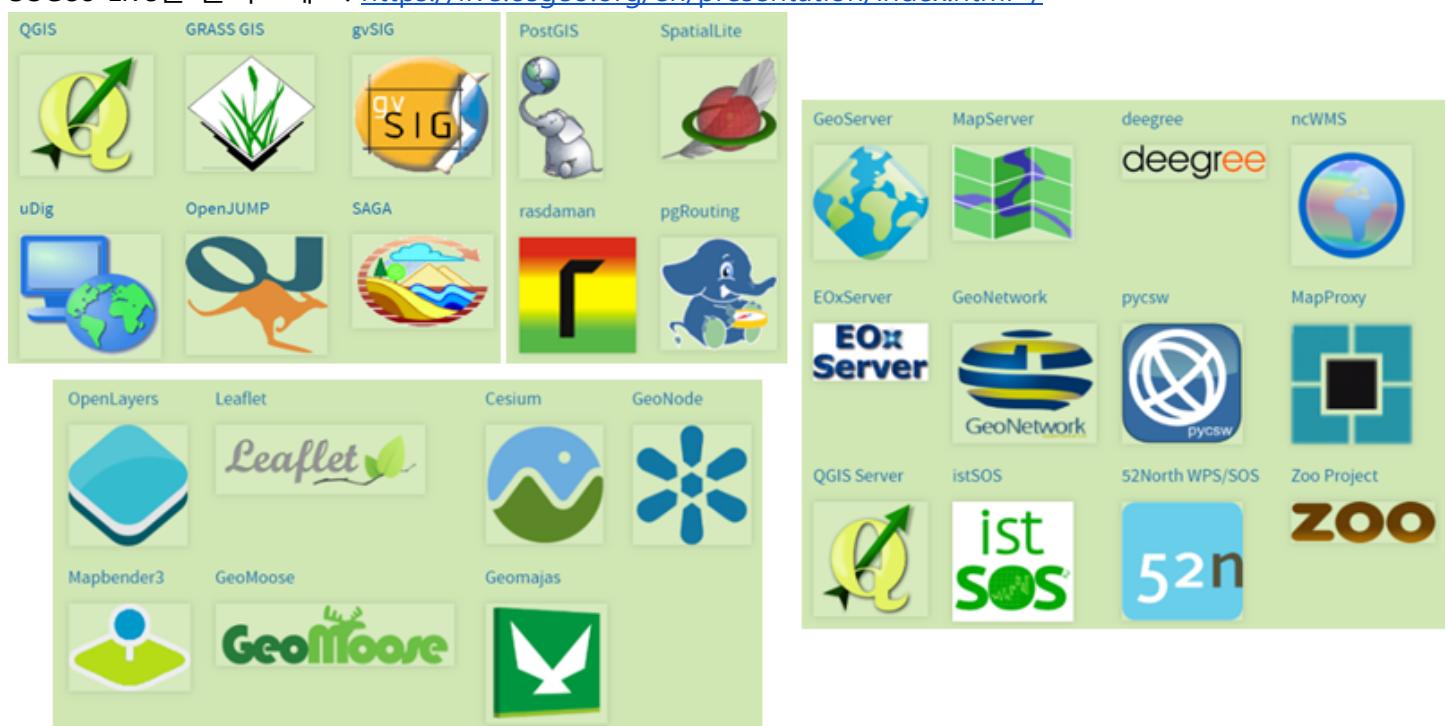
그럼, 기존 업계에서 많이 사용하던 ArcGIS 제품들에 대응되는 대표적인 오픈소스 제품은 어떤 것이 있는지 살펴보겠습니다.



오늘 수업에서 우리도 위의 제품들을 모두 사용하여 서비스를 구성해 볼 것입니다.

공간정보 오픈소스 스택에는 이 밖에도 다른 소프트웨어들이 많이 있습니다.

OSGeo Live를 둘러보세요. <https://live.osgeo.org/en/presentation/index.html#/>



## 2x04 실습에 필요한 SW 설치

- QGIS
  - 서비스용은 아니지만 데이터를 확인할 때 사용합니다.
  - <https://qgis.org/ko/site/forusers/download.html>
  - 2.18 버전을 받습니다.
  - 보통 64 비트를 사용합니다.
  - 설치관리자로 쉽게 설치할 수 있습니다.
  - 예제는 안설치해도 됩니다.
  - OS X, Linux, BSD, 안드로이드용 버전도 있습니다.

**윈도우용 다운로드**

Latest release (richest on features):

 md5	<b>QGIS 독립 설치관리자 버전 2.18 (32 비트)</b>
 md5	<b>QGIS 독립 설치관리자 버전 2.18 (64 비트)</b>

- JAVA
  - GeoServer와 uDig의 실행에 필요합니다.
  - 커맨드 라인에 `java -version` 명령을 쳤을 때 오류가 발생하면 설치해야합니다.
  - JRE 8 이상의 버전이 필요합니다.
  - <http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>
  - 반드시 Accept License Agreement를 선택해야 다운로드 됩니다.

**Java SE Runtime Environment 8u161**

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement    Decline License Agreement

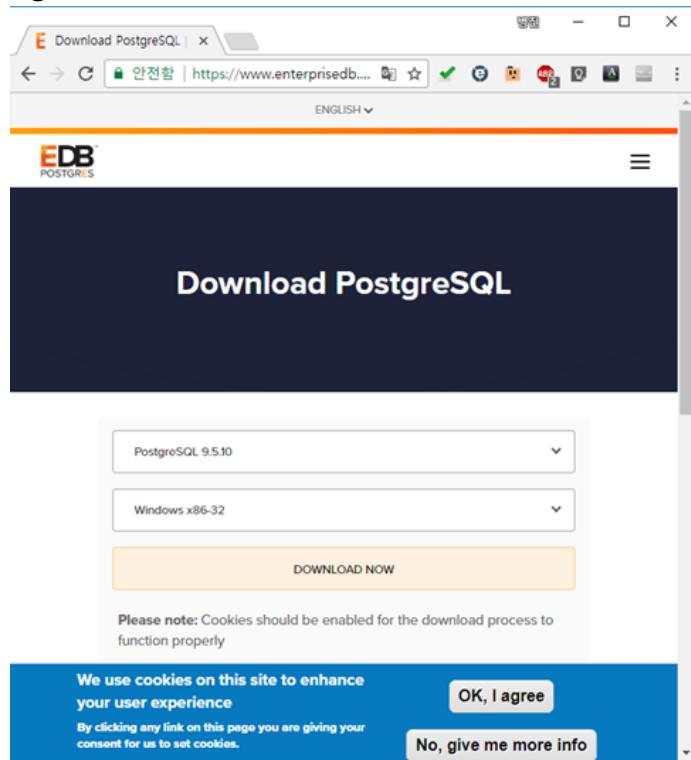
Product / File Description	File Size	Download
Linux x86	63.4 MB	<a href="#">jre-8u161-linux-i586.rpm</a>
Linux x86	79.29 MB	<a href="#">jre-8u161-linux-i586.tar.gz</a>
Linux x64	60.4 MB	<a href="#">jre-8u161-linux-x64.rpm</a>
Linux x64	76.35 MB	<a href="#">jre-8u161-linux-x64.tar.gz</a>
macOS	74.17 MB	<a href="#">jre-8u161-macosx-x64.dmg</a>
macOS	65.86 MB	<a href="#">jre-8u161-macosx-x64.tar.gz</a>
Solaris SPARC 64-bit	52.24 MB	<a href="#">jre-8u161-solaris-sparcv9.tar.gz</a>
Solaris x64	50 MB	<a href="#">jre-8u161-solaris-x64.tar.gz</a>
Windows x86 Online	1.78 MB	<a href="#">jre-8u161-windows-i586-iftw.exe</a>
Windows x86 Offline	61.35 MB	<a href="#">jre-8u161-windows-i586.exe</a>
Windows x86	64.56 MB	<a href="#">jre-8u161-windows-i586.tar.gz</a>
Windows x64 Offline	68.02 MB	<a href="#">jre-8u161-windows-x64.exe</a>
Windows x64	68.58 MB	<a href="#">jre-8u161-windows-x64.tar.gz</a>

- uDig
  - QGIS와 비슷한 데스크탑 툴이지만 스타일 편집시 사용합니다.
  - <http://udig.refractions.net/download/>
  - x86\_64 Installer를 설치합니다.

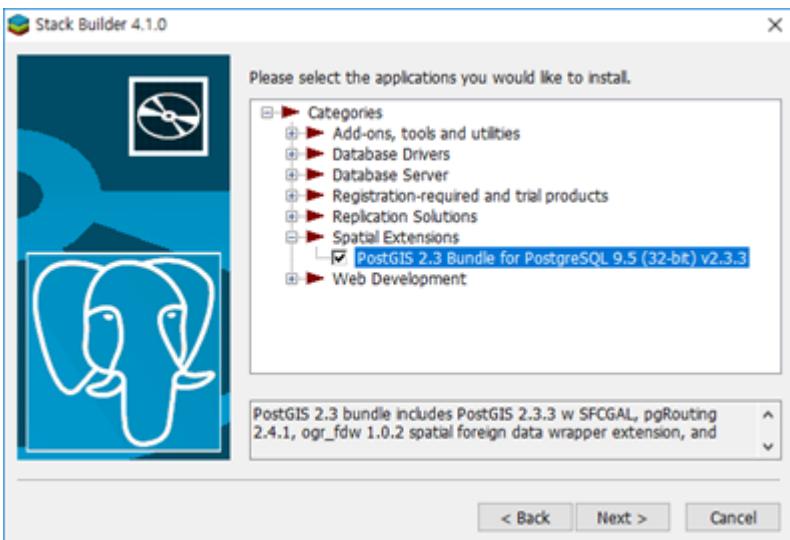
## Release 2.0.0.RC1



- PostgreSQL
  - 공간정보를 저장하는 PostGIS의 기반이 되는 DBMS입니다.
  - <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads#windows>
  - PostgreSQL 9.5, Windows x86-64 버전을 받아 설치합니다.
  - Superuser(postgres) 암호로 postgres를 입력합니다. 실습을 위해 통일합니다.
  - BigSQL 배포본으로 설치시 PostGIS가 설치 어려우니 EDB 배포본으로 설치하는 것이 좋습니다.



- PostGIS
  - PostgreSQL의 버전에 맞는 버전 설치가 필수입니다.
  - PostgreSQL 설치 마지막 단계의 Stack Builder로 설치 가능합니다.
  - Spatial Extension 밑의 PostGIS 2.3을 선택해 설치합니다.
  - 샘플 DB 만들기 옵션을 선택하는 것이 좋습니다.



- GeoServer
  - 공간정보를 온라인으로 서비스하기 위해 사용합니다.
  - 2.11 버전을 설치합니다.
  - <http://geoserver.org/release/maintain/>
  - Windows Installer를 선택합니다.



- Notepad++
  - 텍스트 설정파일을 편집하고 웹코딩도 합니다.
  - <https://notepad-plus-plus.org/download/v7.5.4.html>
  - 32-Bit x86 버전을 선택합니다.



- Apache Web Server
  - 상용 환경과 비슷하게 서비스 구성을 위해 사용합니다.
  - <https://www.apachelounge.com/download/>

## 3x01 세계지도 데이터 받기

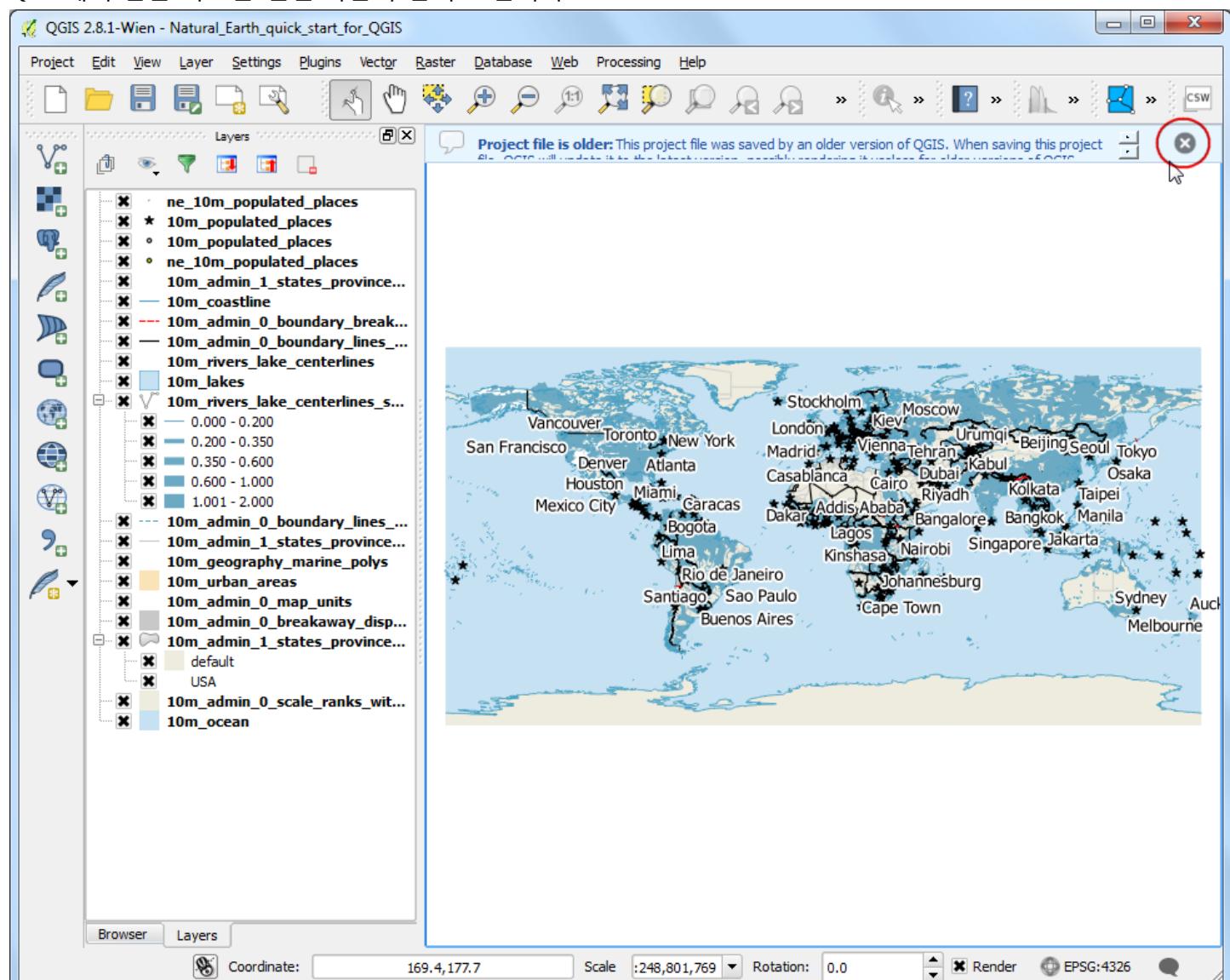
양질의 공간데이터를 받을 수 있는 사이트가 많이 있습니다.

세계지도를 먼저 받아보겠습니다.

오픈된 세계지도 자료 중 가장 유명한 것은 Natural Earth(<http://www.naturalearthdata.com/>)입니다.  
벡터와 래스터로 행정구역, 도로, 수계 등 쓸만한 전세계 데이터를 받을 수 있습니다.

오늘 우리는 이 Natural Earth 데이터를 가지고 QGIS에서 스타일링까지 해둔 자료를 바로 받겠습니다.  
[http://naciscdn.org/naturalearth/packages/Natural\\_Earth\\_quick\\_start.zip](http://naciscdn.org/naturalearth/packages/Natural_Earth_quick_start.zip)  
자료가 커 시간이 오래 걸리니 미리 받아두세요.

QGIS에서 받은 자료를 열면 다음과 같이 보입니다.



출처: [http://www.qgistutorials.com/ko/docs/making\\_a\\_map.html](http://www.qgistutorials.com/ko/docs/making_a_map.html)

## 3x02 국토지리정보원 데이터 받기

오픈데이터를 제공하는 국토정보플랫폼에서 내가 원하는 지역 수치지도를 다운로드 받아보도록 하겠습니다. 일단, 아래 공식 홈페이지에 접속합니다.

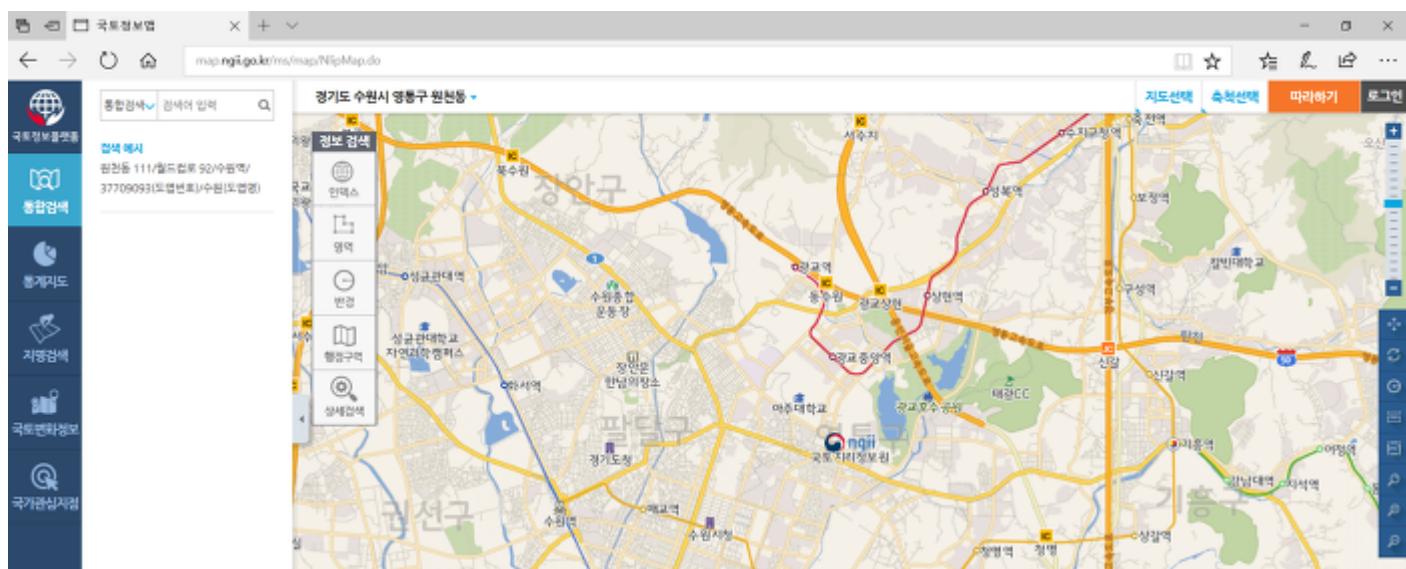
국토정보플랫폼 공식 홈페이지: <http://map.ngii.go.kr/mn/mainPage.do>

**정보 다운로드**  
수치지도, 항공사진, 국가기준점 등  
국토지리정보원에서 제작한 공간정보  
온라인 다운로드 서비스

**국토 정보**  
지명, GNSS측량, 지명정보 등  
국토정보플랫폼을 통해 알아가는  
국토 정보

**업무지원서비스**  
측량기기성능검사, 공공측량관리,  
측량표지조사보고, 적격심사 등  
업무를 위한 지원서비스

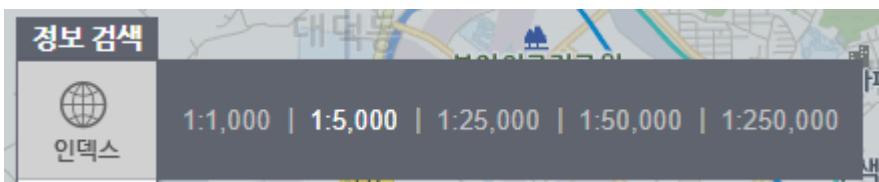
'정보 다운로드'를 선택하면 아래와 같이 국토정보맵이 실행됩니다.



자료를 받기 위해 로그인을 합니다.  
만약 아이디가 없으신 분은 가입을 하셔야 합니다.

## '세종시'를 검색해 봤습니다.

'정보 검색 > 인덱스'에서 1:5,000 축척을 선택하고,



세종특별자치시청이 위치한 도엽을 선택하면,  
화면 좌측에 해당 도엽의 수치지도, 항공사진, 정사영상, 공개DEM 조회 결과가 표시됩니다.

수치지도는 수치지형도 (DXF 파일), 수치지형도Ver2.0 (NGI 파일), 연속수치지형도 (SHP 파일), 기본공간정보(SHP 파일), 토지특성도(SHP 파일), 토지이용현황도(SHP파일) 연안해역기본도(SHP파일)이 있습니다.

여기서는 수치지형도 (DXF 파일), 연속수치지형도(SHP 파일), 2017년도 온맵을 다운로드 받겠습니다.

장바구니에 담고 가보면 대략 다음처럼 보입니다.

총 3 매의 내역이 있습니다

#### 수치지도(1매)

	번호	자료분류	축척	도엽명	도엽번호	형태	갱신일자	갱신사유	지도	금액(원)	비고
<input type="checkbox"/>	1	수치지도V1.0	1:5,000	대전002	36710002	DXF	-	-		0	<button>지도받기</button>

#### 연속수치지도(1매)

	번호	신청방법	축척	형태	파일크기(byte)	면적(km <sup>2</sup> )	자료선택	자료확인	지도	금액(원)	비고
<input type="checkbox"/>	1	사각형	1:5,000	-	-	-				0	<button>바로받기</button>

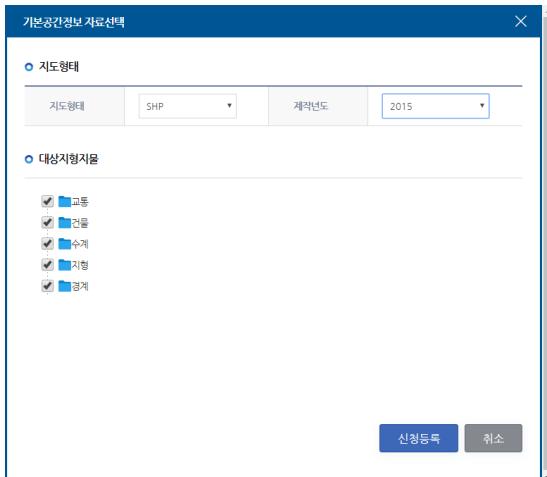
#### 온맵(1매)

	번호	자료분류	자료상세정보	축척	공급년도	형태	자료선택	지도	금액(원)	비고
<input type="checkbox"/>	1	도엽(정사영상)	36710002(대전002)	1:5,000	2017	PDF			0	<button>지도받기</button>

※ 온맵 사용자정의 지도는 자료선택 완료 시 신청 할 수 있습니다.

[온라인신청하기](#) [오프라인신청하기](#) [선택삭제](#)

연속수치지도는 [자료선택]을 눌러 받을 자료 종류 선택하고 확인요청을 하면 시간이 지난 후 공급됩니다.



받을 자료를 다 선택하고 [온라인신청하기]를 누르시고 신청정보를 장성하면 받을 수 있습니다.

각 수치지도 유형의 차이는 다음 그림을 참고하시면 되겠습니다.

이미지 출처: [전자도서] 국가지도의 이해와 활용(2016) |  
<http://www.ngii.go.kr/kor/board/view.do?rbsIdx=31&idx=634>

특성 요소	수치지도 1.0	수치지도 2.0	연속수치지도	온맵
모양 예시				
구조	<ul style="list-style-type: none"><li>• 도형정보만 포함</li><li>• 문자와 기호로 속성정보를 대체</li></ul>	<ul style="list-style-type: none"><li>• 도형정보와 속성정보 모두 포함</li></ul>	<ul style="list-style-type: none"><li>• 도형정보와 속성정보 모두 포함</li></ul>	<ul style="list-style-type: none"><li>• 도형정보만 포함</li><li>• 문자와 기호로 속성정보를 대체</li><li>• 배경영상 포함</li></ul>
묘사 특성	<ul style="list-style-type: none"><li>• 주로 점과 선을 이용하여 지형자물을 묘사</li></ul>	<ul style="list-style-type: none"><li>• 점, 선, 다각형을 모두 이용하여 지형자물을 묘사</li></ul>	<ul style="list-style-type: none"><li>• 점, 선, 다각형을 모두 이용하여 지형자물을 묘사</li></ul>	<ul style="list-style-type: none"><li>• 점, 선, 다각형을 모두 이용하여 지형자물을 묘사</li><li>• 하이브리드 지도 표현</li></ul>
포맷	<ul style="list-style-type: none"><li>• DXF 파일</li></ul>	<ul style="list-style-type: none"><li>• NGI 파일</li><li>• NDA 파일</li><li>• ESRI Shape 파일</li></ul>	<ul style="list-style-type: none"><li>• NGI 파일</li><li>• NDA 파일</li><li>• ESRI Shape 파일</li><li>• Geodatabase</li></ul>	<ul style="list-style-type: none"><li>• PDF 파일</li></ul>
제공 · 제작 단위	<ul style="list-style-type: none"><li>• 측척별 도엽 단위로 제작 · 제공됨</li></ul>	<ul style="list-style-type: none"><li>• 측척별 도엽 단위로 제작 · 제공됨</li></ul>	<ul style="list-style-type: none"><li>• 연속화된 수치지도(도엽 단위로 구분되지 않음)</li><li>• 사용자 지정 영역 단위로 제공</li></ul>	<ul style="list-style-type: none"><li>• 측척별 도엽 단위로 제작 · 제공됨</li></ul>
제작 측척	<ul style="list-style-type: none"><li>• 1:1,000</li><li>• 1:5,000</li><li>• 1:25,000</li></ul>	<ul style="list-style-type: none"><li>• 1:1,000</li><li>• 1:5,000</li></ul>	<ul style="list-style-type: none"><li>• 1:5,000</li><li>• 1:25,000</li></ul>	<ul style="list-style-type: none"><li>• 1:5,000</li><li>• 1:25,000</li><li>• 1:50,000</li><li>• 1:250,000</li></ul>

### 3x03 지적도 등 국가공간정보포털 자료 받기

국가공간정보포털에서도 많은 양질의 공간자료를 받으실 수 있습니다.  
국가중점개방데이터가 대표적이며 이 중에서도 연속지적도, 용도지역지구정보 등이 많이 이용됩니다.

이번 실습에서는 연속지적도를 받아보겠습니다.

먼저 국가공간정보포털에 접속합니다.

<http://www.nsdi.go.kr/?menuno=2679>

그리고 아이콘이 모여있는 중앙의 국가중점개방데이터 아이콘을 누릅니다.

리스트의 가장 하단으로 가 부동산 개방데이터(15년)의 연속지적도정보의 [SHP] 버튼을 누릅니다.

▶ 부동산 개방데이터(15년) - 11건

GIS건물통합정보	공유지연명정보	대지권등록정보	법정구역정보	연속지적도형정보
API SHP	API CSV	API CSV	API SHP	API SHP
용도지역지구정보	지적도근점정보	지적삼각보조점정보	지적삼각점정보	토지등급정보
API SHP	API SHP	API SHP	API SHP	API CSV
토지임야정보				
API CSV				

시도에서 세종특별자치시를 선택하고 [조회]를 누릅니다.  
검색되어 나온 리스트에서 가장 최신 데이터의 [SHP] 버튼을 눌러 자료를 받습니다.

국토정보 서비스

국가중점 데이터

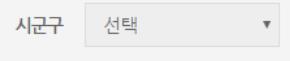
## 부동산 개방데이터

### 연속지적도형정보

전산화된 지적도 및 임야도의 도면상 경계점을 연결하여 연속된 형태로 작성한 도면정보(측량자료로 사용하지 못하는 참조용 도면정보)

업데이트 주기	전체-월간, 변동-일간	다운로드 횟수	66153
최근 생성일	2018-01-30	QGIS 사용가이드	 <a href="#">설치 바로가기</a>  <a href="#">DOWNLOAD</a>
등록일	2015-12-24	컬럼 정의서 	공간정보 파일데이터의 항목 정의  <a href="#">DOWNLOAD</a>

목록

기준일자  ~  시도  세종특별자치시  시군구 선택 

구분  전체 데이터셋 명  조회

구분	시도	데이터셋명	기준일자	파일크기	다운로드
전체데이터	세종특별자치시	연속지적도형정보	2018-01-06	24,821 KB	 <a href="#">SHP</a>

국가중점개방데이터들은 대부분 이처럼 로그인 없이 받을 수 있습니다.  
하지만, 국가공간정보포털의 다른 자료들은 가입하고 로그인 해야만 받을 수 있는 경우가 많습니다.

## 3x04 기타 쓸만한 데이터 확보처

전세계 위성영상도 쉽게 받을 수 있습니다.

NASA에서 Blue Marble이라는 이름으로 위성영상과 지형등의 자료로 각 월별 영상을 잘 만들어 두었습니다.  
<https://neo.sci.gsfc.nasa.gov/view.php?datasetId=BlueMarbleNG-TB>



File Type으로 GeoTIFF를 선택하고 0.1 degrees 영상을 다운 받습니다.

기타 사이트는 목록만 보도록 하겠습니다.

### 국내

#### 국가공간정보유통센터

- 국가공간정보유통센터([사이트 바로가기](#))
- 개인 또는 기관으로 가입 필수
- 구매 또는 무료 공간정보 신청 가능
- 도로명 전자지도(새주소 공간데이터), 해양주제도, 국가공간정보(공간정보오픈플랫폼에서 서비스하는 주제도 일부), 보행자용 DB, 민간 공간정보, 공공데이터 등을 신청하여 무료로 활용 가능

#### 공공데이터포털

- 공공데이터포털([사이트 바로가기](#))
- 가장 많은 종류의 데이터를 제공하며, 대부분의 공공기관 데이터 공개 지원
- 가입 후 파일, Open API 등을 이용해 활용 가능

#### 국가통계포털

- 국가통계포털([사이트 바로가기](#))
- 국내외 통계 데이터를 행정구역단위로 엑셀, CSV, 텍스트 등으로 다운로드

#### 통계지리정보

- 통계지리정보([사이트 바로가기](#))
- 가입 필수

- 집계구, 통계지역경계(시도/시군구/읍면동/도시화지역, 도시권경계 등), 센서스지도(하천, 건물, 도로, 철도, 등고 등) 신청 후 다운로드

## 건축데이터 민간개방 시스템

- 건축데이터 민간개방 시스템([사이트 바로가기](#))
- 가입 필수
- 건축물 인허가정보, 건축물대장정보 제공

## 도로명주소 안내시스템

- 도로명주소 안내시스템([사이트 바로가기](#))
- 공간데이터는 국가공간정보유통센터를 통해서 제공
- 매칭테이블 및 주소변환서비스, API 등은 이 사이트를 통해 제공

## 국가교통DB센터

- 국가교통DB센터([사이트 바로가기](#))
- 가입 필수
- 도로네트워크, 철도 등 교통주제도, OD 등 자료 신청 후 수령

## 표준노드링크관리시스템

- 표준노드링크관리시스템([사이트 바로가기](#))
- 가입은 필요없으며, 표준노드링크관리시스템에서 생산한 도로네트워크 다운로드

## 공간정보오픈플랫폼

- 공간정보오픈플랫폼([사이트 바로가기](#))
- 가입 및 인증키 필수
- OGC 표준 스펙(WMS, WFS) 또는 Open API를 통해 웹에서 사용 가능

## 지방행정데이터개방

- 지방행정데이터개방([사이트 바로가기](#))
- 지방자치단체가 보유, 관리하는 생활관련 데이터의 개방
- 일반음식점, 관광업소 등 우선 개방, 이후 의료, 보건 등 48분야 530여종 확대 개방 예정

## 공동주택관리정보시스템

- 공동주택관리정보시스템([사이트 바로가기](#))
- 공동주택 단지정보 및 공동주택 가격정보

## 부동산공시가격알리미

- 부동산공시가격알리미([사이트 바로가기](#))
- 표준지공시지가, 개별공시지가 검색
- 개별, 표준, 공동주택가격 검색

## 토양지하수 정보시스템

- 토양지하수 정보시스템([사이트 바로가기](#))
- 토양측정망, 지하수 수질측정망, 골프장 농약사용량 주소정보로 제공

## 지방자치단체 공공데이터 포털

- 서울 열린 데이터 광장([사이트 바로가기](#))
- 부산 공공데이터포털([사이트 바로가기](#))
- 강원 공공데이터([사이트 바로가기](#))
- 경기도 공공데이터 공개 포털([사이트 바로가기](#))
- 경상북도 공공데이터포털시스템([사이트 바로가기](#))
- 전북 3.0 포털([사이트 바로가기](#))
- 제주특별자치도 공공데이터포털([사이트 바로가기](#))
- 수원 공공데이터포털([사이트 바로가기](#))

# 전세계

## OpenStreetMap(OSM)

- OpenStreetMap([사이트 바로가기](#))
- 가장 인기있는 참여형 지도제작 플랫폼 및 데이터
- ODbL 라이선스에 따라 사용
- Open API 또는 데이터 직접 다운로드가 가능하며, QGIS에서 직접 Import도 가능

## EarthExplorer

- USGS EarthExplorer([사이트 바로가기](#))
- 다양한 전세계 위성영상을 받을 수 있음

## ASTER GDEM

- ASTER GDEM([사이트 바로가기](#))
- 전세계 30m 해상도 DEM 제공
- 가입 후 도엽별로 선택하고 신청 후 인증 - [사이트 바로가기](#)

## SRTM(Shuttle Radar Topography Mission)

- SRTM(Shuttle Radar Topography Mission)([사이트 바로가기](#))
- 30m(일부지역) ~ 500m 등 다양한 해상도의 전세계 DEM 제공

## NASA's Socioeconomic Data and Applications Center (SEDAC)

- SEDAC([사이트 바로가기](#))
- Socioeconomic data (agriculture, climate, conservation, governance, hazards, health, infrastructure, land use, marine and coastal, population, poverty, remote sensing, sustainability, urban and water)

## UNEP Environmental Data Explorer

- UNEP([사이트 바로가기](#))
- Freshwater, population, forests, emissions, climate, disasters, health and GDP spatial and non-spatial data.

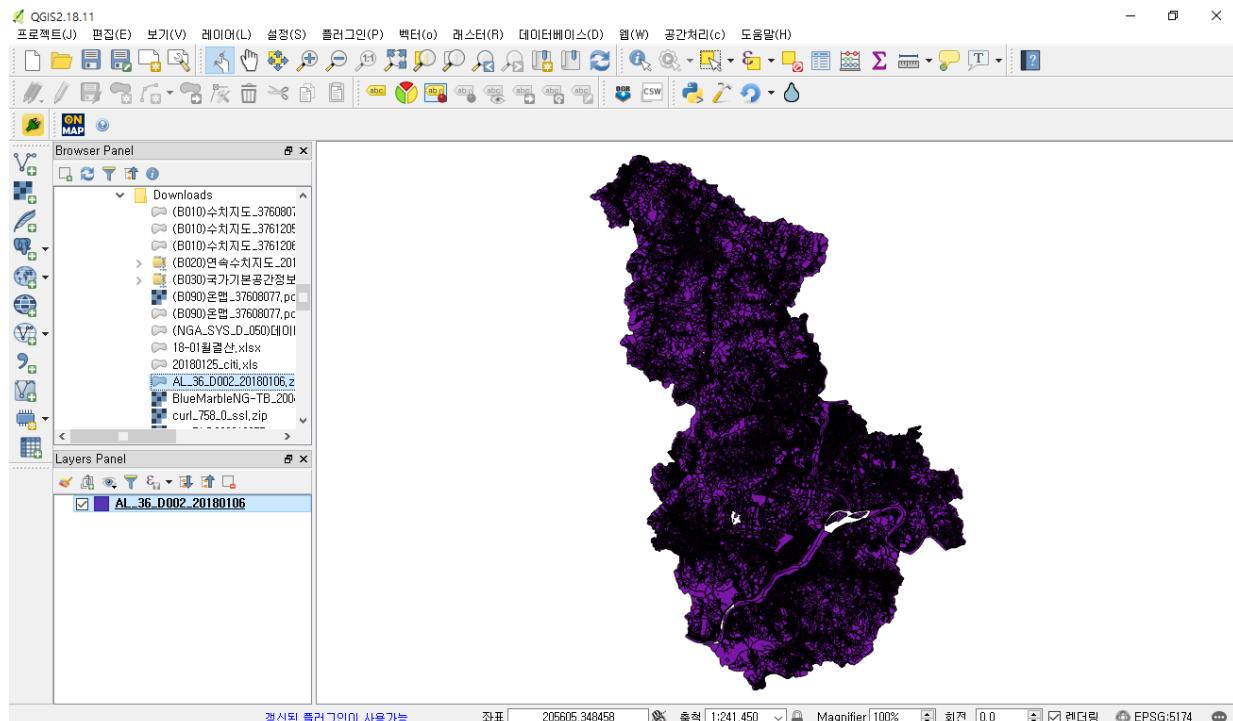
출처: <http://geeps.krihs.re.kr/?wiki=%EB%85%A5%EC%97%94%EC%9A%A9%EC%8A%A4%ED%8A%B8%EC%84%9C%EC%8A%A4%ED%8A%B8%EC%84%9C>

## 3x05 한글 코드페이지 문제 해결

이제 한국 개발자라면 누구나 겪어야 하는 한글 코드페이지 문제를 해결해 봅시다.  
QGIS에서 아까 받은 세종시의 지적도를 열어보겠습니다.

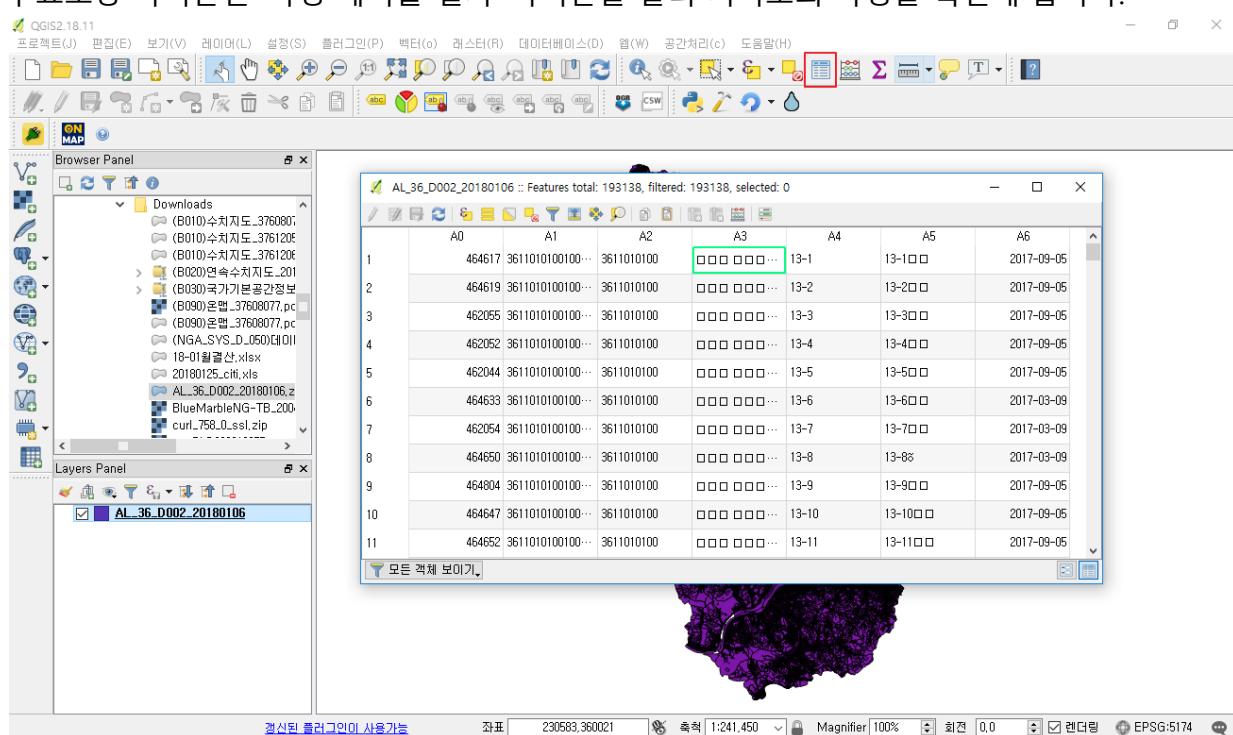
QGIS를 열고 Browser Panel에서 앞에서 다운로드 받은 AL\_36\_D002\_20180106.zip 파일을 찾아 더블클릭합니다.

그러면 자동으로 Layers Panel에 지적도가 들어오며 열립니다. zip 파일 안에 하나의 레이어만 들어온 경우 압축을 풀지 않아도 이렇게 잘 열립니다.



이제 한글 문제를 확인해 보겠습니다.

툴바에서 표모양 아이콘인 '속성 테이블 열기' 아이콘을 눌러 지적도의 속성을 확인해 봅시다.

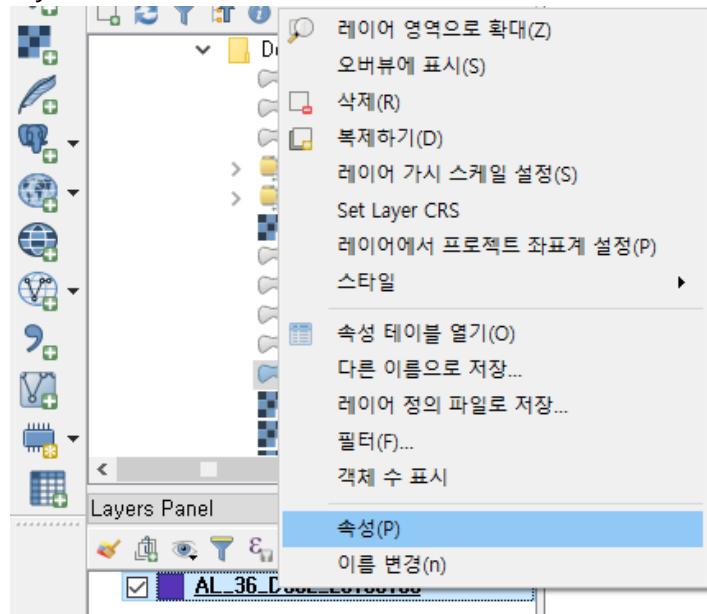


속성 테이블을 보면 A3, A5열에 알아볼 수 없는 사각형이 잔뜩 보입니다.  
한글이 정상적으로 보이지 않고 깨져 보이는 것입니다. 이런 현상 때문에 일부 사용자는 QGIS는 한글 문제가 있어 사용할 수 없다고 까지 말하기도 합니다.

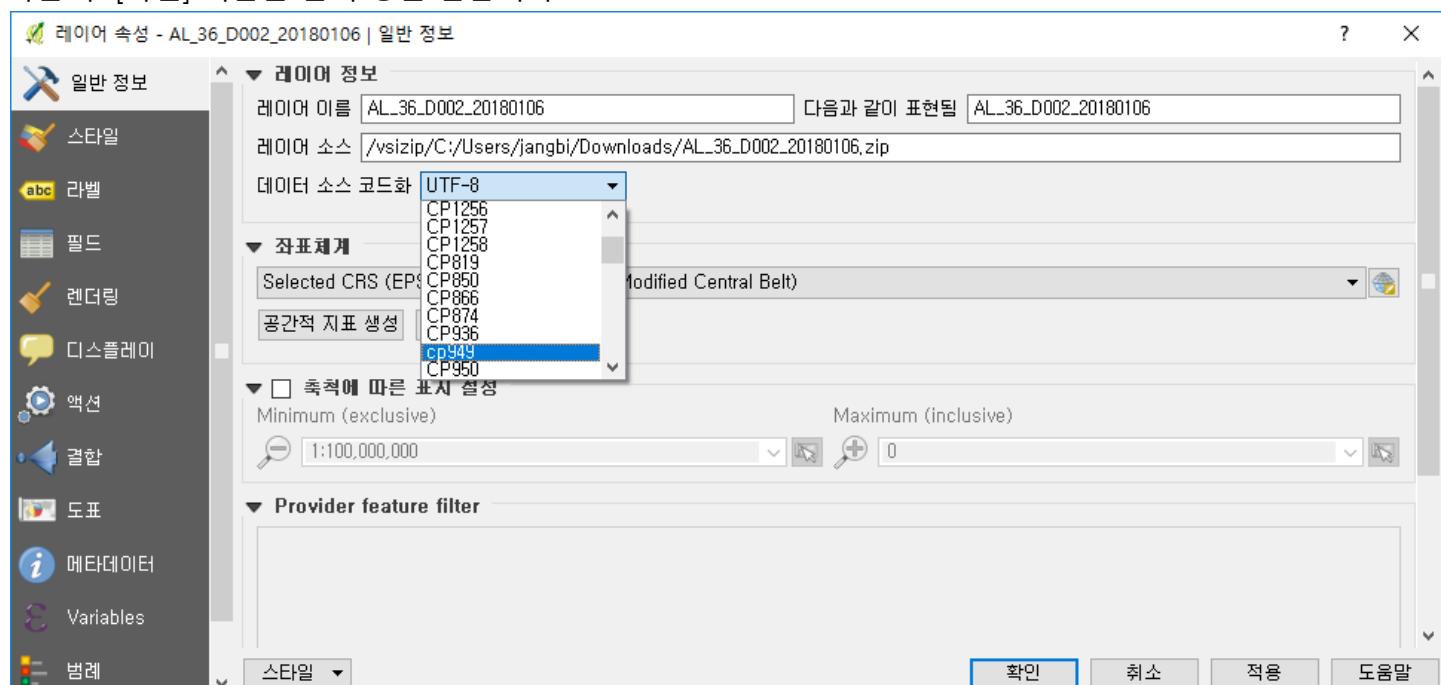
하지만, 조금만 아시면 문제 없이 사용 가능하고, 심지어는 QGIS는 국제표준을 잘 따르고 있고 우리가 받은 지적도 자료가 국제표준과 다른 코드페이지를 사용하고 있어서 문제가 되는 것입니다.

이제 해결해 봅시다.

Layer Panel에서 AL\_36\_D002\_20180106 레이어를 오른쪽 마우스로 선택하고 [속성] 메뉴를 선택합니다.



레이어 속성 창에서 일반정보 탭을 선택하고 데이터 소스 코드화(?) 항목에서 UTF-8로 된 부분을 cp949로 바꾼 후 [확인] 버튼을 눌러 창을 닫습니다.



이제 다시 '속성 테이블 열기' 버튼을 누르면 한글이 정상적으로 보입니다.



The screenshot shows a QGIS interface with a table containing 11 rows of data. The columns are labeled A0 through A6. The first column is a row index from 1 to 11. The second column, A0, contains numerical values. The third column, A1, contains long strings of characters starting with '3611010100100...'. The fourth column, A2, contains shorter strings like '3611010100'. The fifth column, A3, contains Korean text such as '세종특별자치...'. The sixth column, A4, contains strings like '13-1'. The seventh column, A5, contains strings like '13-1전'. The eighth column, A6, contains dates like '2017-09-05'. A green box highlights the value '464617' in the A0 column of the first row. The bottom of the window has a toolbar with icons for '모든 객체 보기' (View all objects) and other GIS functions.

	A0	A1	A2	A3	A4	A5	A6
1	464617	3611010100100...	3611010100	세종특별자치...	13-1	13-1전	2017-09-05
2	464619	3611010100100...	3611010100	세종특별자치...	13-2	13-2전	2017-09-05
3	462055	3611010100100...	3611010100	세종특별자치...	13-3	13-3전	2017-09-05
4	462052	3611010100100...	3611010100	세종특별자치...	13-4	13-4전	2017-09-05
5	462044	3611010100100...	3611010100	세종특별자치...	13-5	13-5전	2017-09-05
6	464633	3611010100100...	3611010100	세종특별자치...	13-6	13-6전	2017-03-09
7	462054	3611010100100...	3611010100	세종특별자치...	13-7	13-7전	2017-03-09
8	464650	3611010100100...	3611010100	세종특별자치...	13-8	13-8전	2017-03-09
9	464804	3611010100100...	3611010100	세종특별자치...	13-9	13-9전	2017-09-05
10	464647	3611010100100...	3611010100	세종특별자치...	13-10	13-10전	2017-09-05
11	464652	3611010100100...	3611010100	세종특별자치...	13-11	13-11전	2017-09-05

지금 한 작업이 저장된 자료에 맞게 코드페이지를 변경해 한글이 잘 보이게 한 것입니다.  
'데이터 소스 코드화'라는 용어가 '코드페이지 설정'의 다른(?)용어입니다.

최근 국제표준은 모든 자료교환시 문자셋 즉 코드페이지에 의한 문제를 없애기 위해 UTF-8을 쓰도록 권장하고 있습니다.

지적도의 올바른 코드페이지 였던 CP949도 역시 윈도우 OS에서 사용중인 우리나라 표준이고 국제표준이기도 하지만, 최근에는 이런 문자셋의 문제를 줄이기 위해 UTF-8을 사용할 것을 권장하고 있습니다.

맥 컴퓨터의 OS인 OS X, Linux 등에서는 UTF-8이 기본 문자셋이며, 웹문서도 UTF-8로 저장하는 것이 표준입니다.

QGIS 뿐 아니라 최근 버전의 ArcGIS에서도 자료 저장시 기본적으로 UTF-8을 사용하도록 되어 있습니다.

이렇게 한글이 들어간 자료를 다룰 때에는 코드페이지를 잘 확인하는 것이 매우 중요합니다.  
문자셋과 코드페이지는 거의 비슷한 개념이며, 쉽게 이야기 하자면 문자를 바이너리로 저장하는 규약을 의미합니다. 이 규약이 너무 여러 가지라 우리가 괴로운 것이지요.

이것 하나만 기억합시다.

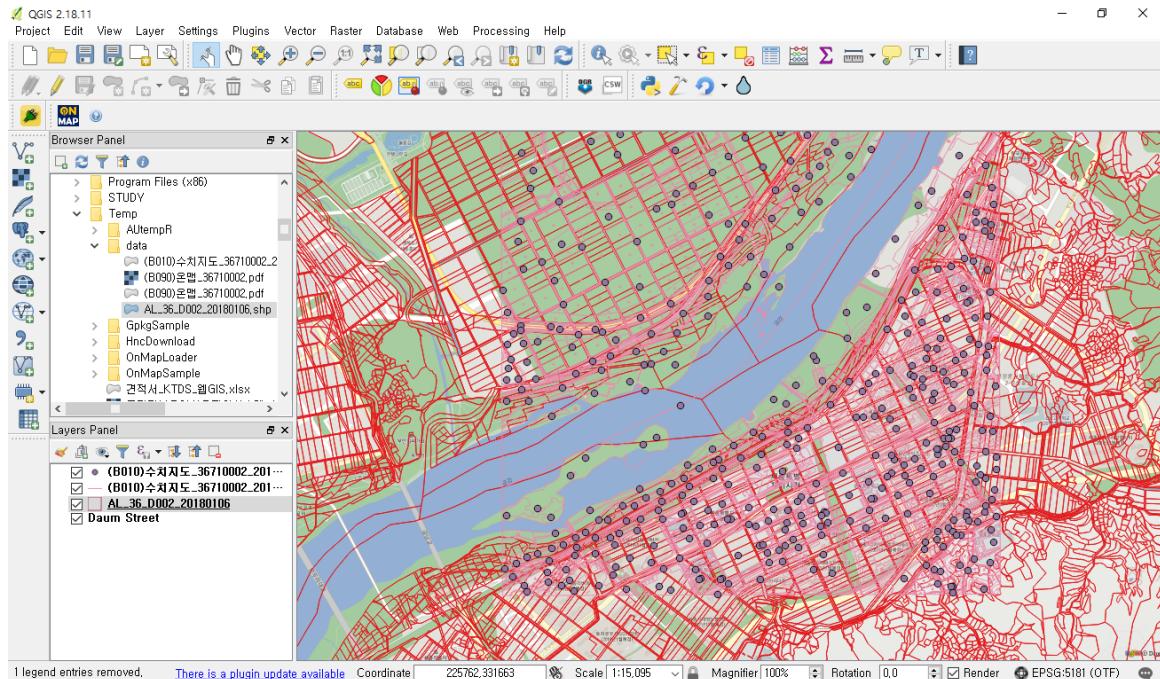
한글이 있는 자료는 꼭 한글이 정상적으로 보이는지 확인해야 하고, 정상적으로 안보이면 CP949로 바꾸어 보고, 그래도 안보이면 UTF-8로 바꿔보면 됩니다.

만약 이 두가지로 바꿔봐도 정상적으로 보이는 경우가 없다면, 자료를 구한 곳에 다시 달라고 하세요. 한글이 깨진 것이고, 복구가 안되는 상황입니다.

## 3x06 좌표계 문제 해결

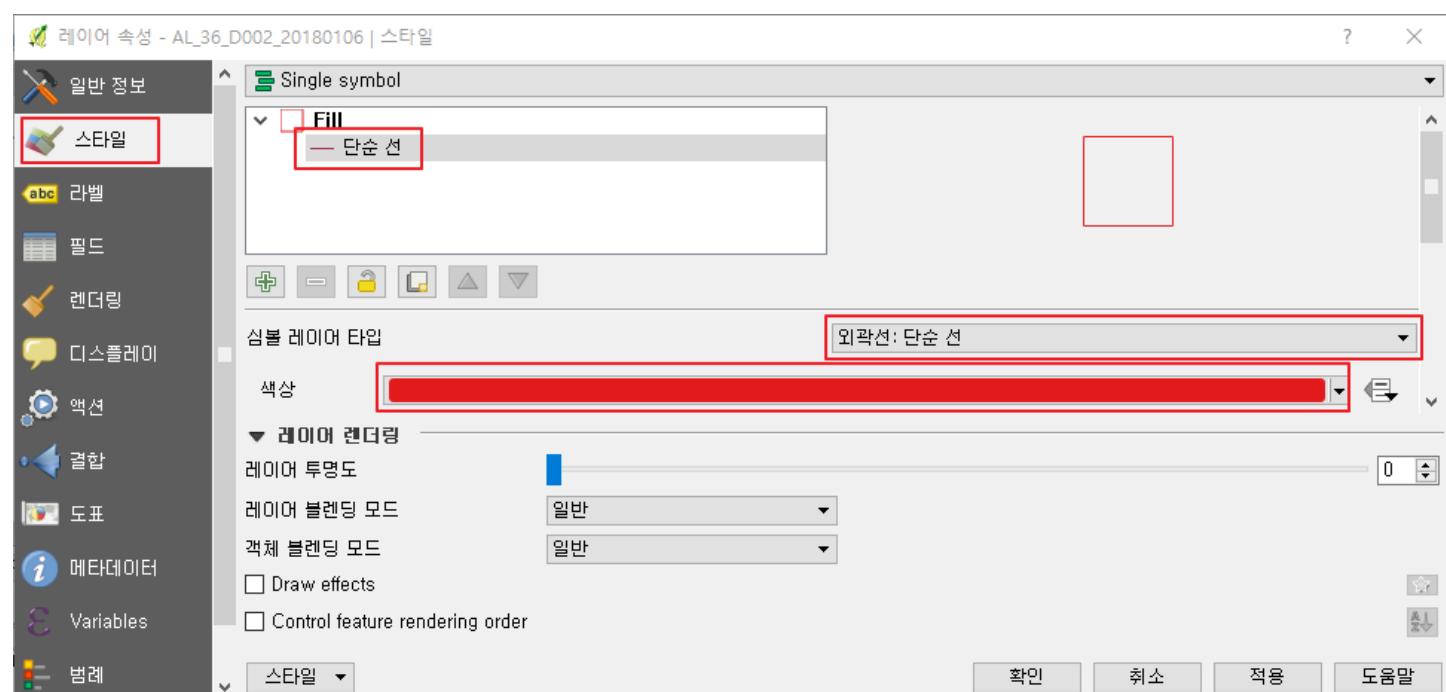
이제 공간정보를 다루는 개발자를 가장 괴롭히는 좌표계 문제를 해결해 봅시다.

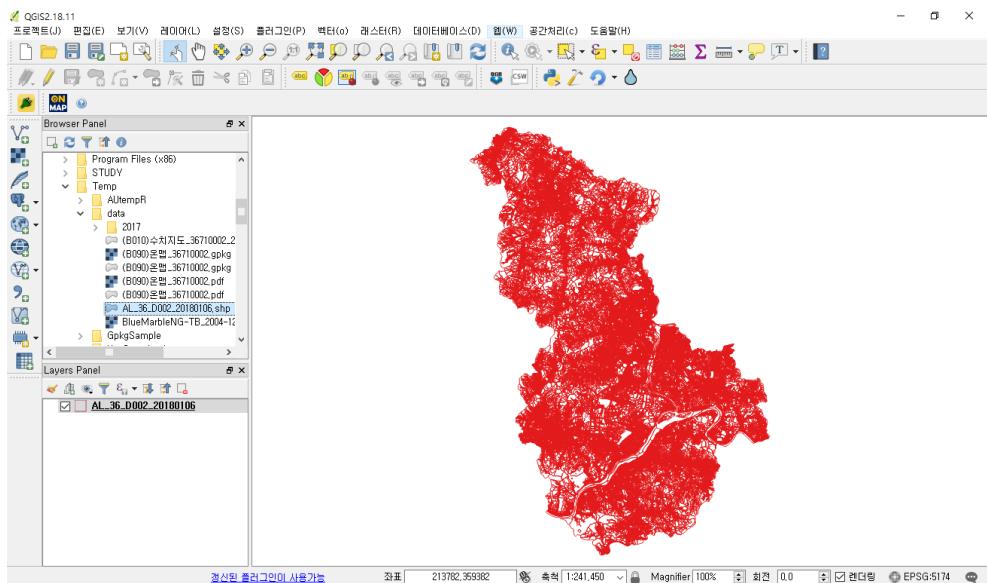
우리가 받은 세종시의 여러가지 자료들을 아래 그림처럼 잘 맞게 겹치는 것이 목표입니다.



먼저 앞에서 한글 문제를 해결했던 세종시 지적도를 좀 더 지적도처럼 표현해 보겠습니다.  
지적도는 보통 붉은 색 선으로 많이 표현합니다.

Layers Panel에서 AL\_36\_D002\_20180106 레이어를 오른쪽 클릭해 [속성] 메뉴를 선택합니다.  
스타일 탭을 선택하고, Fill 아래의 단순 채우기를 선택 후, 심볼 레이어 타입을 '외곽선: 단순 선'으로 바꿉니다.  
마지막으로 선의 색상을 붉은 색으로 바꾸고 [확인]을 선택하시면 됩니다.





이제 지적도가 속이 빈 붉은 선으로 보입니다. 하지만, 단지 지적도만 띄워서는 위치가 잘 맞는지 확인이 어렵습니다.

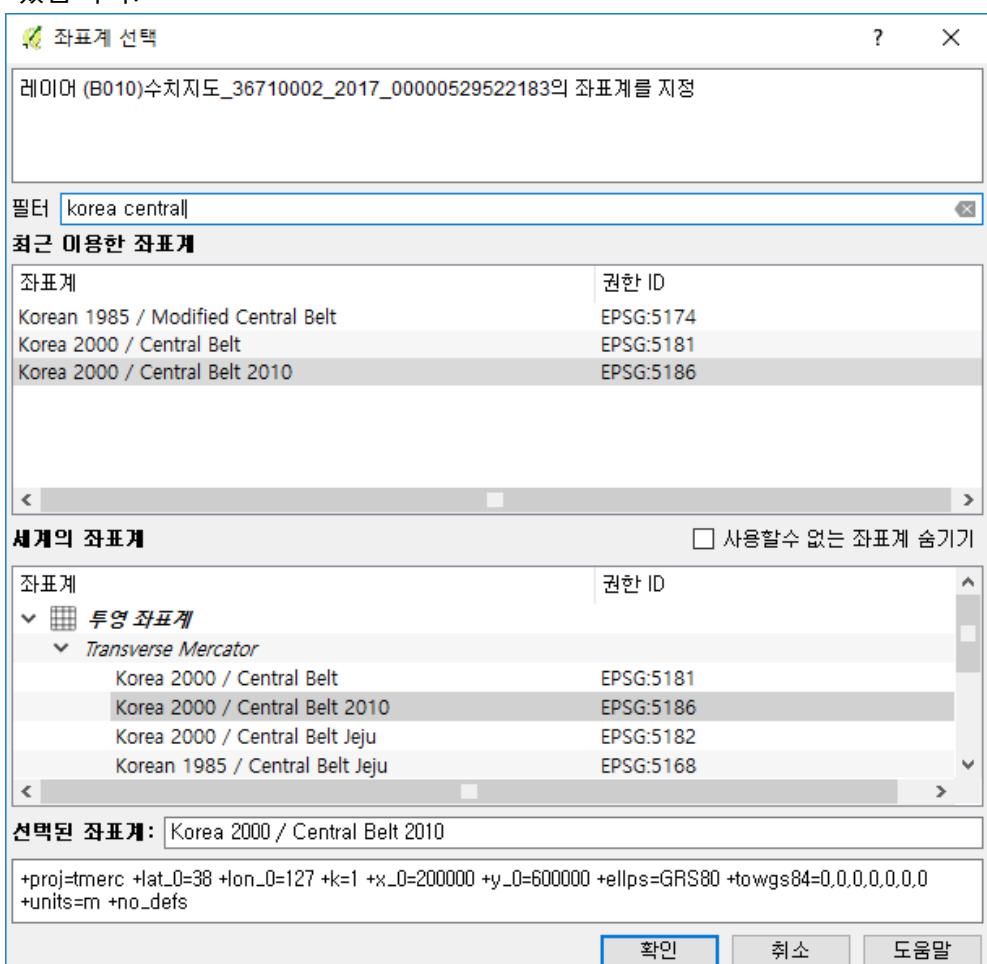
지형도도 한번 같이 띄워서 확인해 보도록 하겠습니다.

국토정보플랫폼에서 받은 (B010)수치지도\_36710002\_2017\_00000529522183.dxf 파일을 같이 열어보록 하겠습니다.

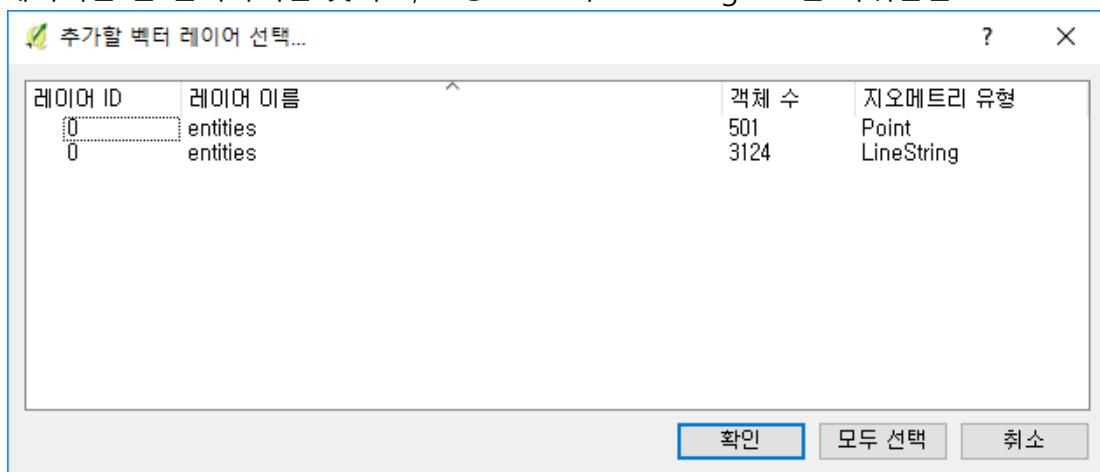
Browser Panel에서 이 파일을 찾아 더블클릭하면 됩니다.

DXF에는 좌표계 정보가 없어 좌표계를 물어보는데, 국토지리정보원 지형도의 중부지방은 EPSG:5186 좌표계를 사용합니다.

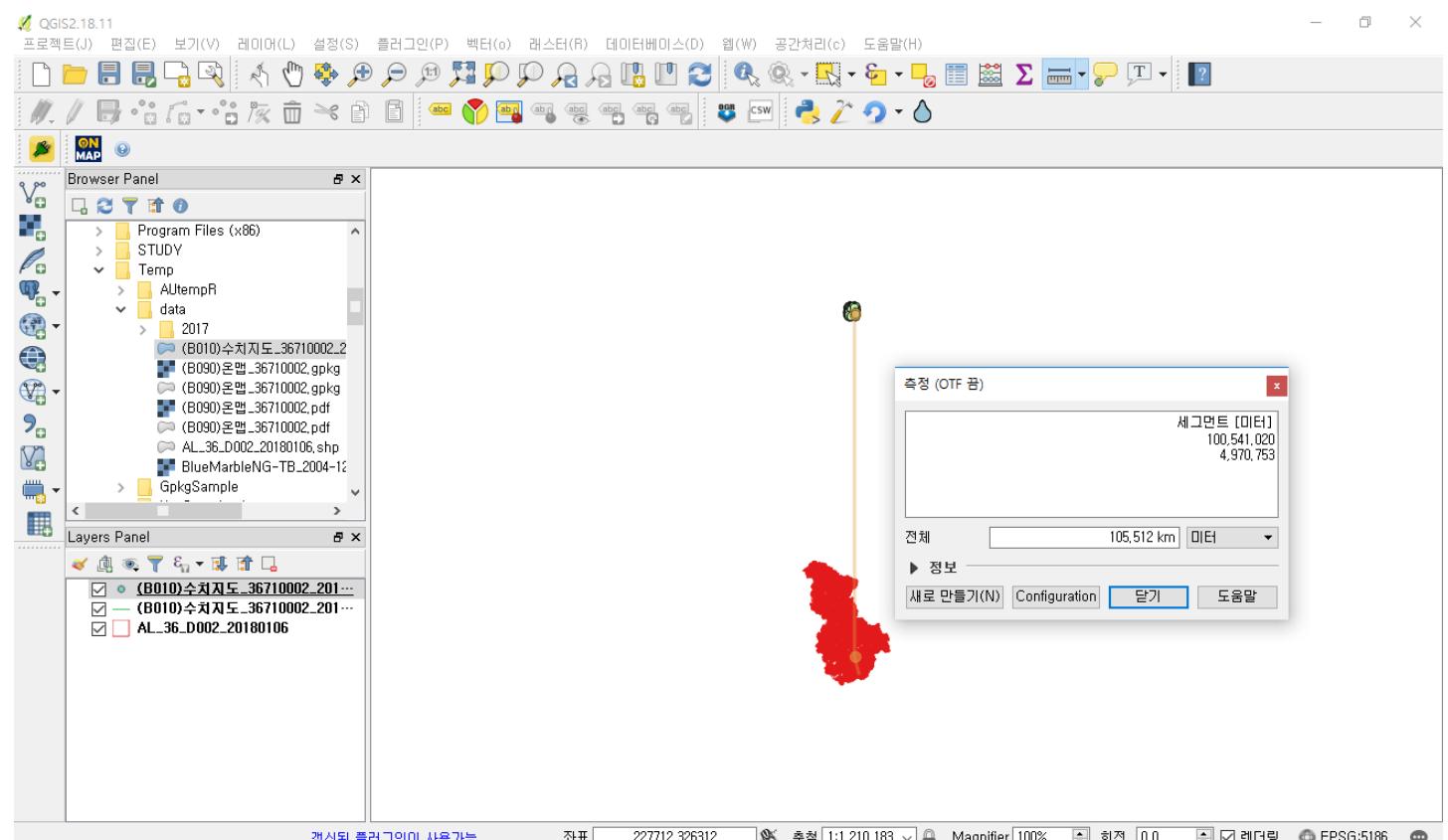
좌표계 선택 화면의 필터에 'korea central'을 입력하면 중부원점 좌표계들이 보이는데 이 중 선택하시면 좀 쉽게 지정하실 수 있습니다.



좌표계 선택이 끝나면 읽어들일 레이어를 선택하는 화면이 나옵니다.  
DXF 파일의 레이어를 잘 인식하지는 못하고, 그냥 Point와 LineString으로만 나누는군요.

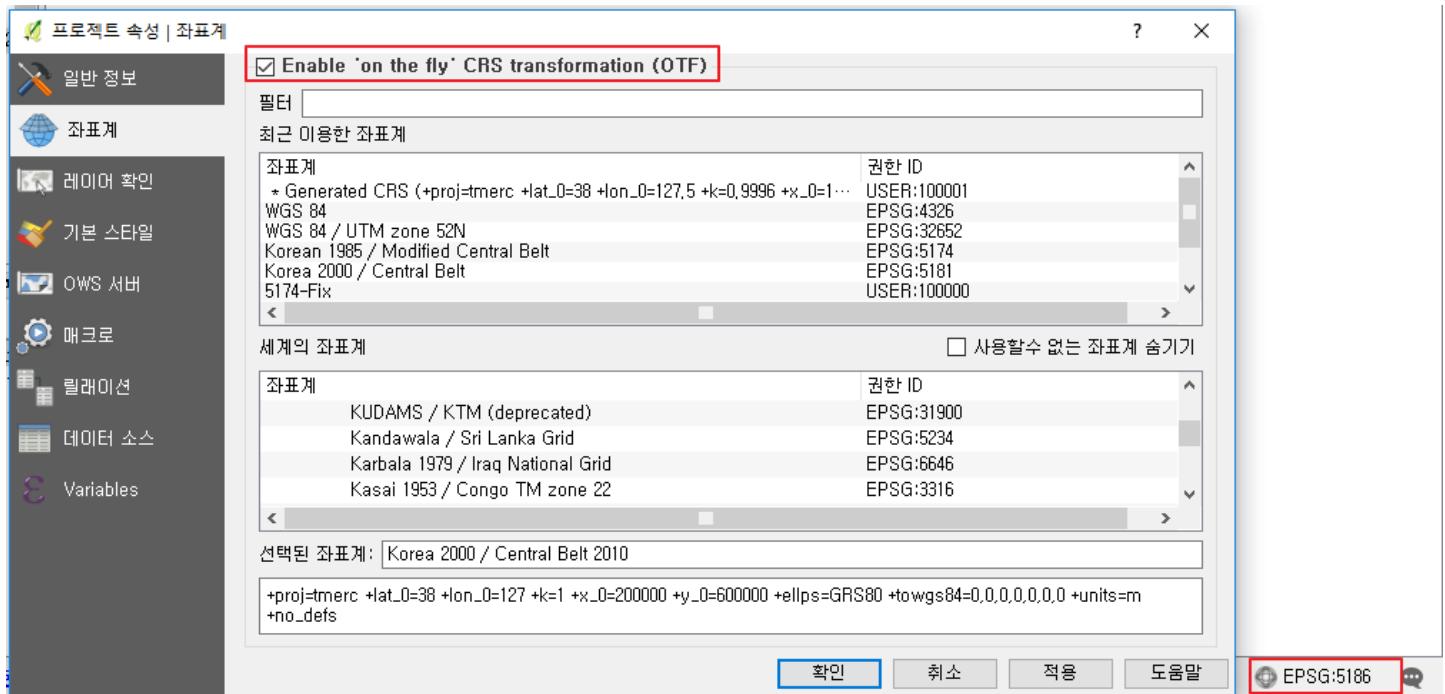


모두 선택하고 확인 하세요.  
또 각 레이어별 좌표계를 물어보는데 아까와 동일하게 EPSG:5186 좌표계를 선택하시면 됩니다.



지적도와 지형도를 동시에 띄워보니 둘 다 세종시의 자료인데 엄청 떨어져 보이네요.  
강의 위치가 맞지를 않네요. 거리를 재어보니 10만 미터 정도 떨어져 있습니다.

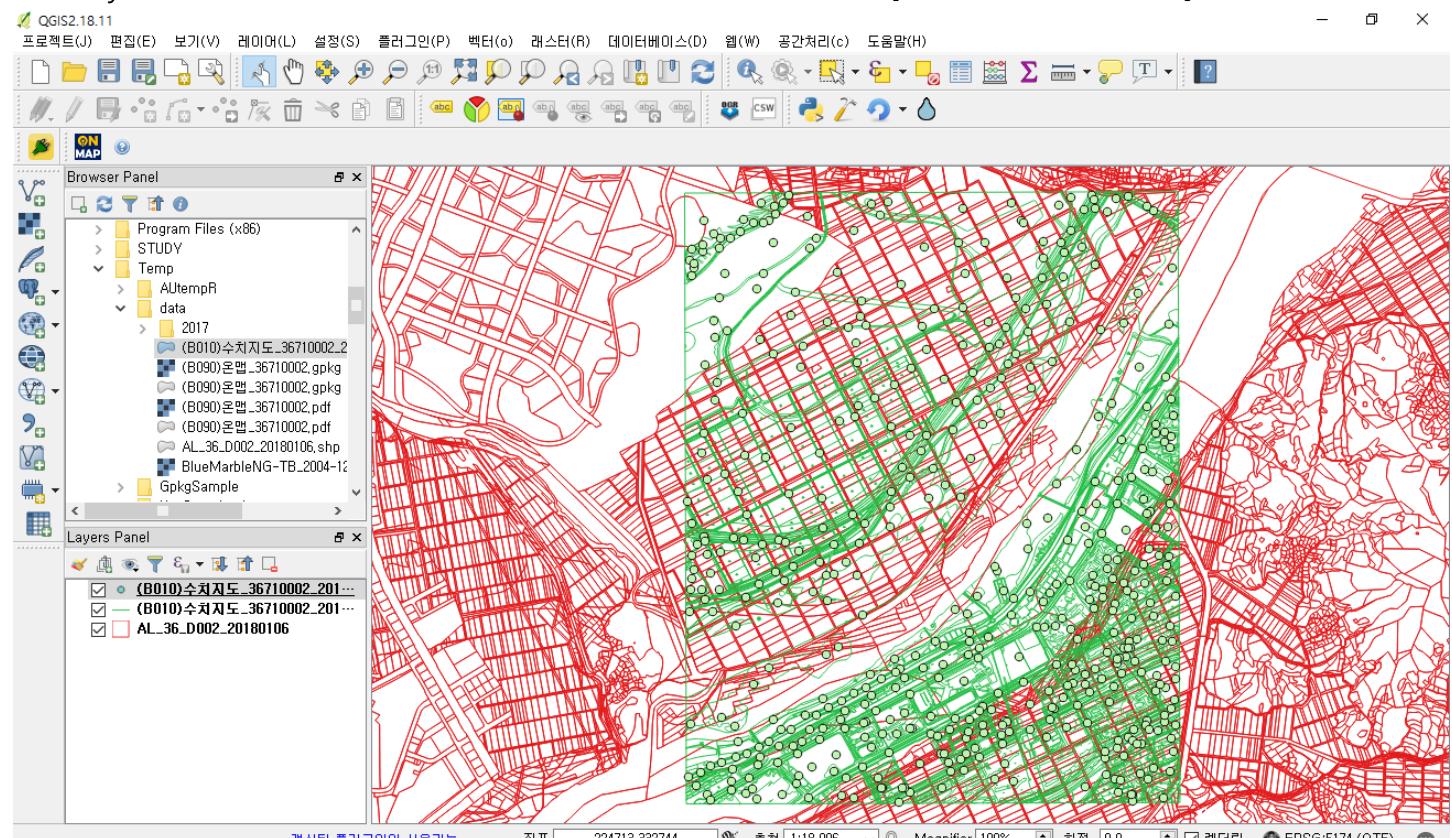
이렇게 보면 보통 Y축에 50만을 더하는 좌표계와 60만을 더하는 좌표계 사이의 차이입니다.  
지금 실습에서는 각 자료의 좌표계는 정확히 지정하였기에, 좌표계를 잘못 지정한 것이 아니고 실시간 좌표계  
변환이 활성화되지 않아서 이렇게 보이는 것입니다.



QGIS 좌하단에 있는 좌표계 표시 부분에서 EPSG:5186 부분을 선택하면, 프로젝트 속성의 좌표계 설정을 한번에 띄울 수 있습니다.

여기서 Enable 'on the fly' CRS transformation (OTF)를 체크하여 활성화하고 [확인]을 누릅니다.

이제 Layers Panel에서 수치지도 레이어를 오른쪽 마우스로 클릭해 [레이어 영역으로 확대] 버튼을 누릅니다.

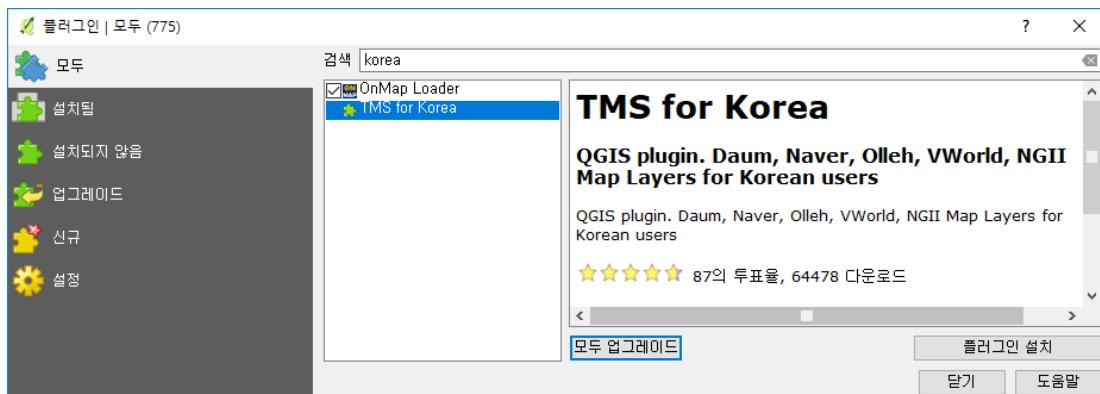


이제 실시간 좌표계 변환(OTP)가 활성화되어 서로 다른 좌표계를 가진 2가지 자료의 위치가 얼추 비슷하게 맞았는데... 아직도 강 부분을 보면 수백미터 차이가 남을 알 수 있습니다. 뭐가 잘못된 것일까요?

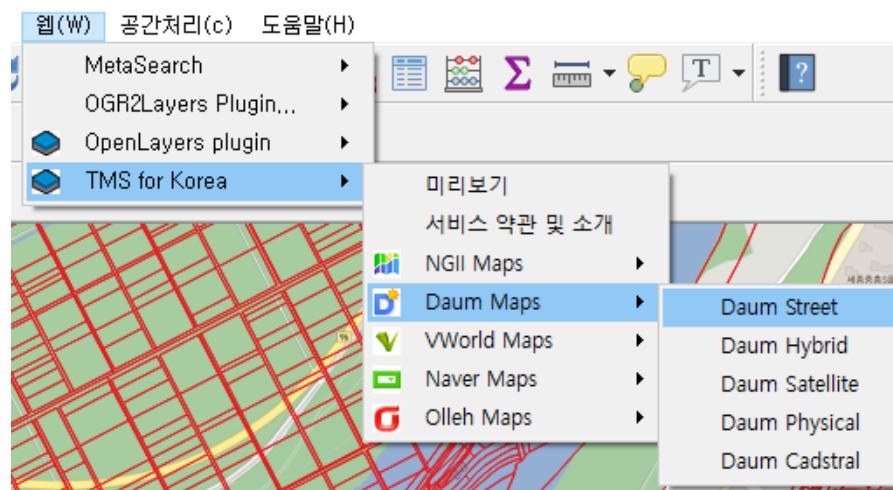
두 레이어 중 어느 것이 위치가 맞고 틀린 것일까요? 혹은 둘 다 틀린 것일까요?  
이를 확인해 보기 위해 인터넷 지도를 바탕에 깔아 보겠습니다.

QGIS 메뉴에서 [플러그인 / 플러그인 관리 및 설치...] 메뉴를 선택합니다.

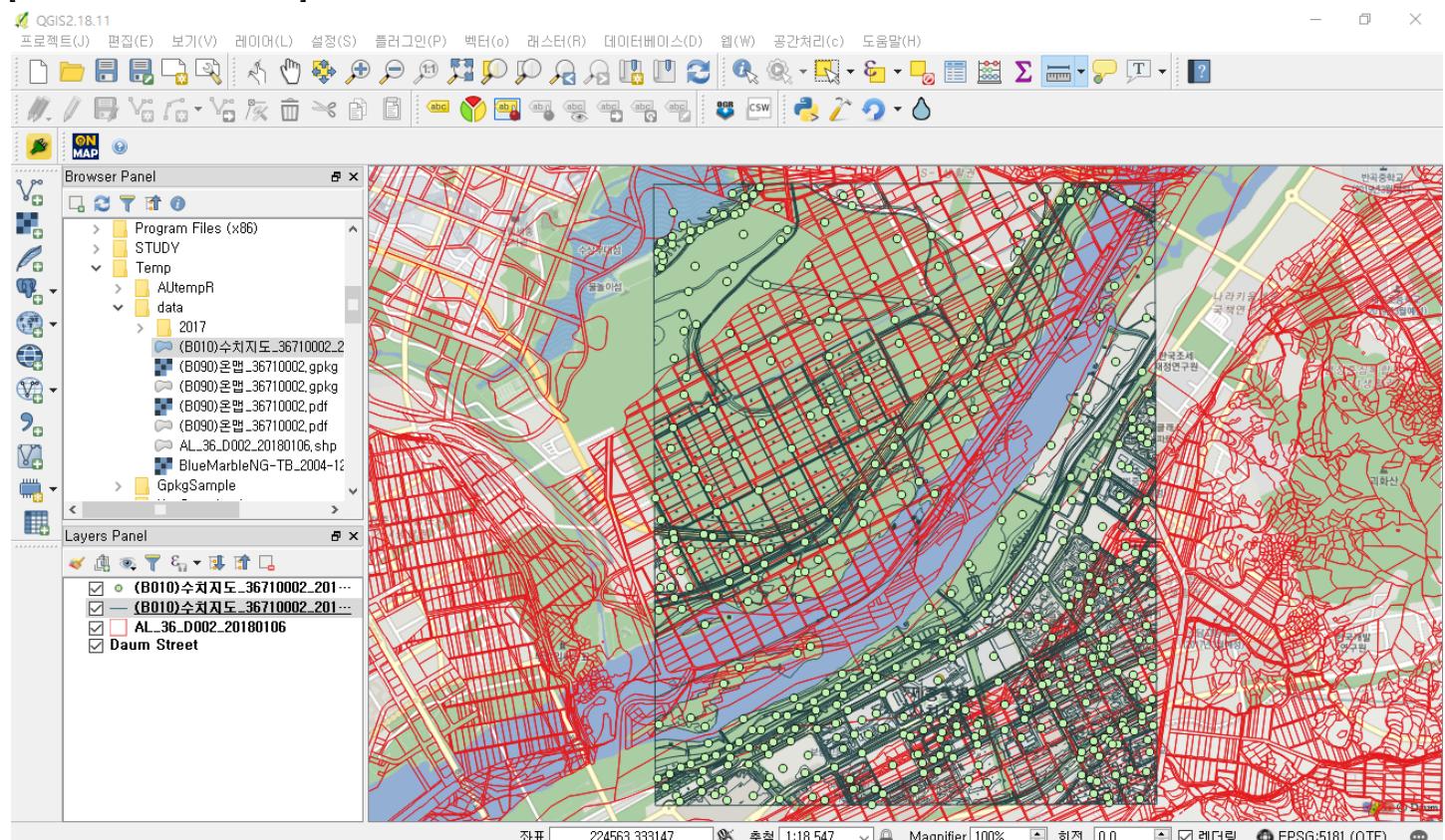
검색에 korea를 입력하면 한국 사용자의 필수 플러그인인 TMS for Korea 플러그인이 보입니다.  
선택하고 [플러그인 설치] 해 주십시오.



설치가 끝나면 플러그인 관리자 창을 닫고, QGIS의 [웹 / TMS for Korea / Daum Maps / Daum Street] 메뉴를 선택합니다.



그리고 Layers Panel에서 Daum Street 레이어를 가장 아래에 가게 끌어다 놓고, 수치지도 레이어를 선택하여 [레이어 영역으로 확대] 하시면 아래 그림처럼 인터넷 지도와 자료들을 중첩해 볼 수 있습니다.



보아 하니... 지형도가 맞고 지적도의 위치가 이상해 보입니다.

이제 좌표계 문제의 끝판왕인 지적도 위치 맞추기를 해 보겠습니다.

이렇게 수백미터 정도 위치가 틀리는 문제는 국내 자료에서 많이 생깁니다. 원인은 2000년대 이전에 많이 사용하던 Bessel 타원체를 이용한 좌표계의 정보가 잘못된 것 때문입니다.

우리가 받은 지적도를 관리하는 KLIS 시스템이 EPSG:5174라는 좌표계를 사용중인데, EPSG:5174 좌표계가 Bessel 타원체를 사용하고 있어 문제가 됩니다.

올바른 좌표계 정보를 찾기위해 다음 웹 페이지로 가 보겠습니다.

<http://osgeo.kr/17>

웹페이지에서 5174를 검색해 보면 올바른 좌표계 정보를 찾을 수 있습니다.

## [오래된 지리원 표준]

2002년 이전에 지리원의 지형도와 KLIS 등 국가 시스템에서 사용되었던 좌표계입니다.

\*보정된 서부원점(Bessel) - KLIS에서 서부지역에 사용중

EPSG:5173

```
+proj=tmerc +lat_0=38 +lon_0=125.0028902777778 +k=1 +x_0=200000 +y_0=500000 +ellps=bessel +units=m  
+no_defs +towgs84=-115.80,474.99,674.11,1.16,-2.31,-1.63,6.43
```

\*보정된 중부원점(Bessel): KLIS에서 중부지역에 사용중

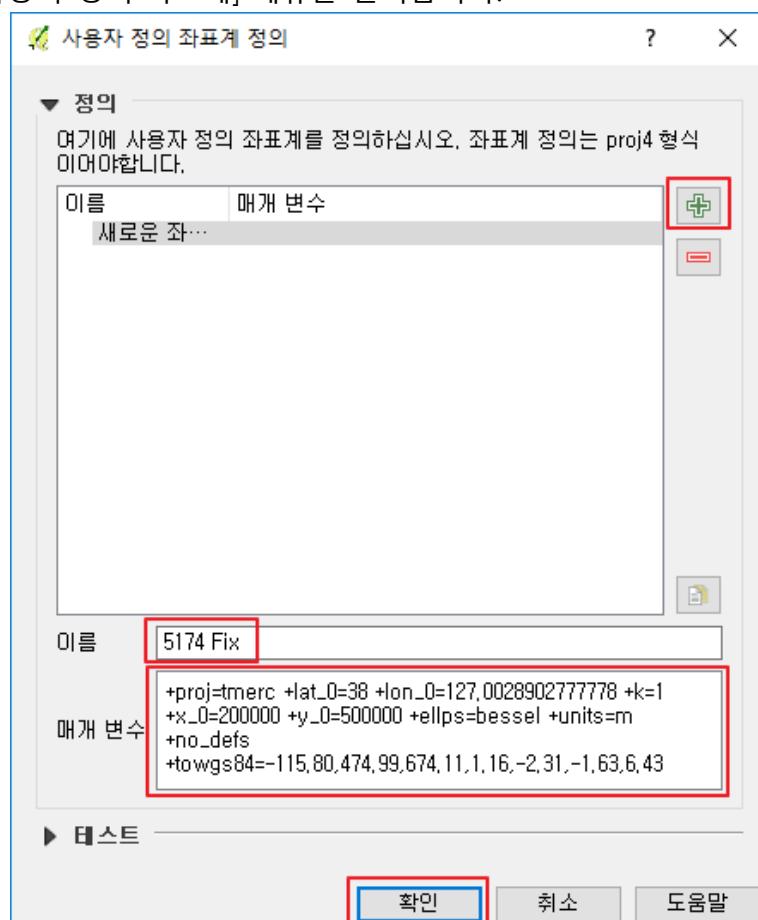
EPSG:5174

```
+proj=tmerc +lat_0=38 +lon_0=127.002  
+no_defs +towgs84=-115.80,474.99,674.11,1.16,-2.31,-1.63,6.43
```

여기서 EPSG:5174 아래의 +proj 부분부터 아래 줄의 -1.63,6.43 부분까지를 클립보드에 복사합니다.  
중요한 부분이 +towgs84 인자인데, 이 부분이 기본 값에는 누락되어 문제가 되고 있습니다.

```
+proj=tmerc +lat_0=38 +lon_0=127.0028902777778 +k=1 +x_0=200000 +y_0=500000 +ellps=bessel +units=m +no_defs  
+towgs84=-115.80,474.99,674.11,1.16,-2.31,-1.63,6.43
```

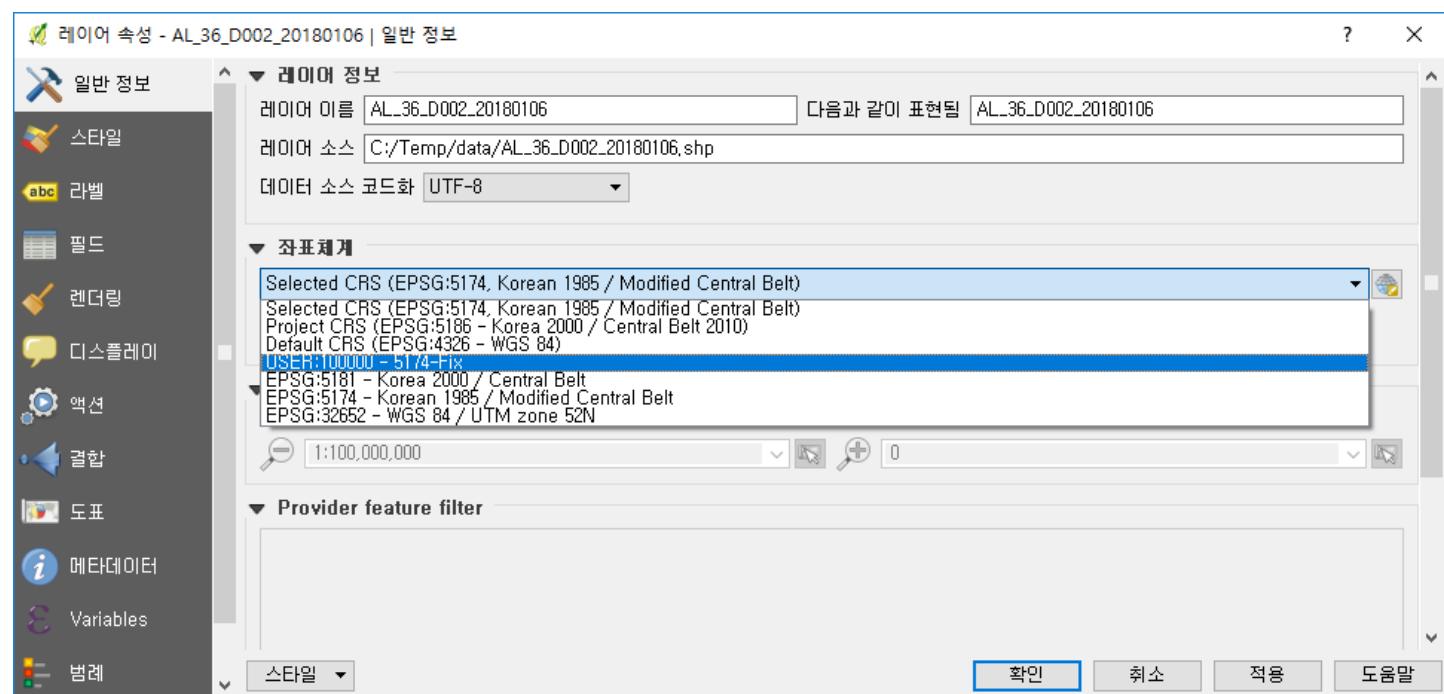
QGIS 메뉴에서 [설정 / 사용자 정의 좌표계] 메뉴를 선택합니다.



[+] 버튼을 눌러 새 좌표계를 만들 수 있도록 하고, 이름에 '5174 Fix' 입력 후, 매개 변수에 복사해 둔 값을 붙여넣어 줍니다. 이 때 출처정보가 복사된 것을 지워주셔야 합니다.

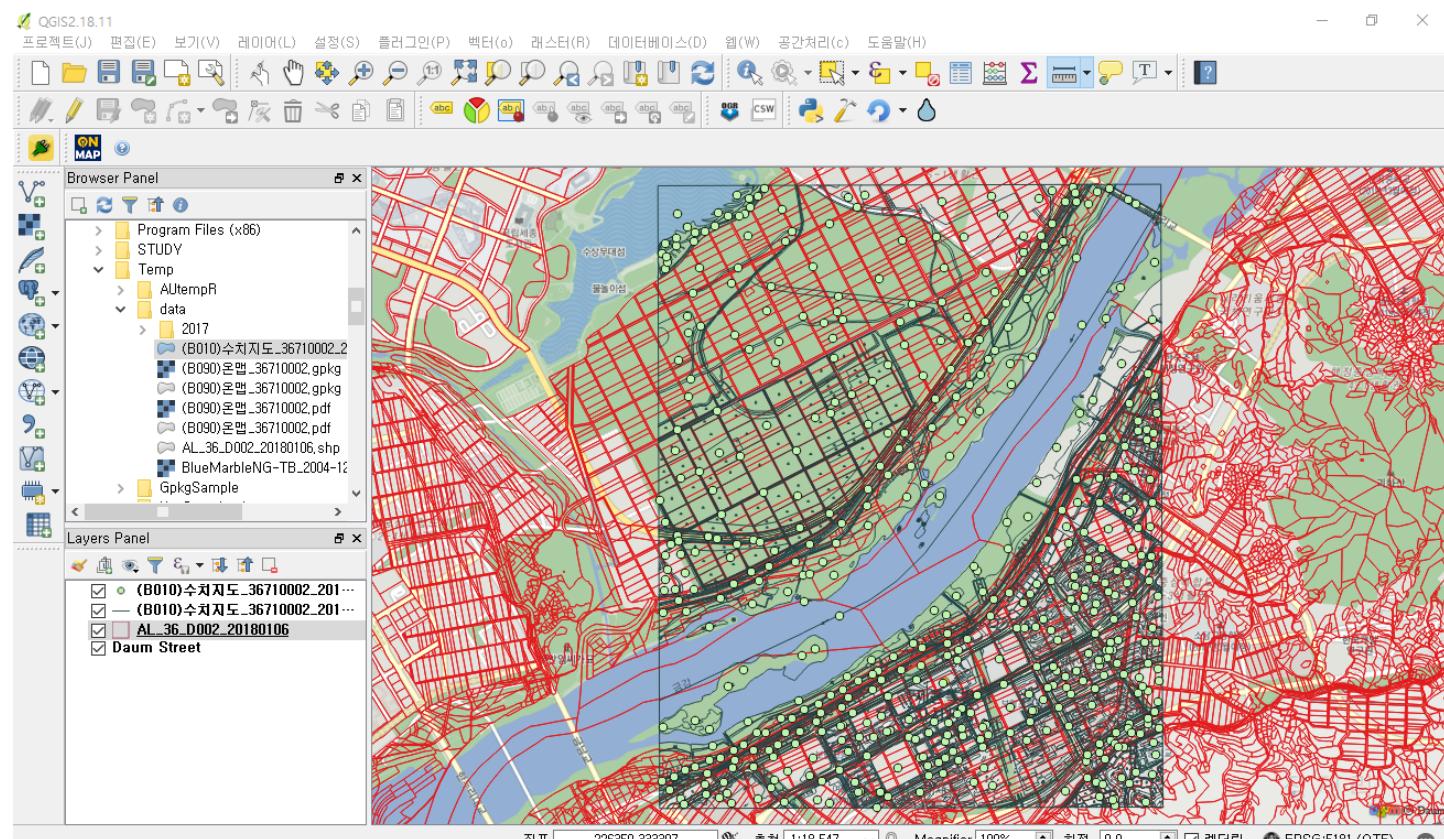
이제 지적도에 올바른 좌표계 정보를 지정해 주겠습니다.

Layers Panel에서 지적도 레이어를 선택하고, [속성] 메뉴를 선택해 레이어 속성 창을 띄웁니다.



일반정보 탭을 선택하고 좌표체계에서 방금 만든 5174 Fix 좌표계를 찾아 선택 후 [확인] 을 눌러 적용합니다.

이제 잘 맞았네요.



좌표계와 좌표계를 변환하는 라이브러리인 proj4에 대해 좀 더 자세히 알고 싶으신 분은 다음 링크의 자료를 참고하세요.

<https://www.slideshare.net/jangbi882/proj4-32605736>

## 3x07 온맵과 GeoPackage를 통해 보는 발전방향

국토지리정보원에서 받은 PDF 포맷의 지도인 온맵은 지형도에 정사영상까지 들어있는 매우 유용한 지도입니다. 하지만, 위치정보를 담은 GeoPDF가 아닌 일반 출력용 PDF이기에 다른 공간자료와 중첩해 사용하기 힘듭니다.

개발자 강의이니 개발을 통해 이런 문제도 극복한 사례를 잠시 살펴보도록 하겠습니다.

다음 링크로 가 보겠습니다.

<https://gaia3d.github.io/OnMapLoader/>

온맵을 읽어 공간데이터로 변환시켜주는 플러그인인 OnMapLoader 입니다.

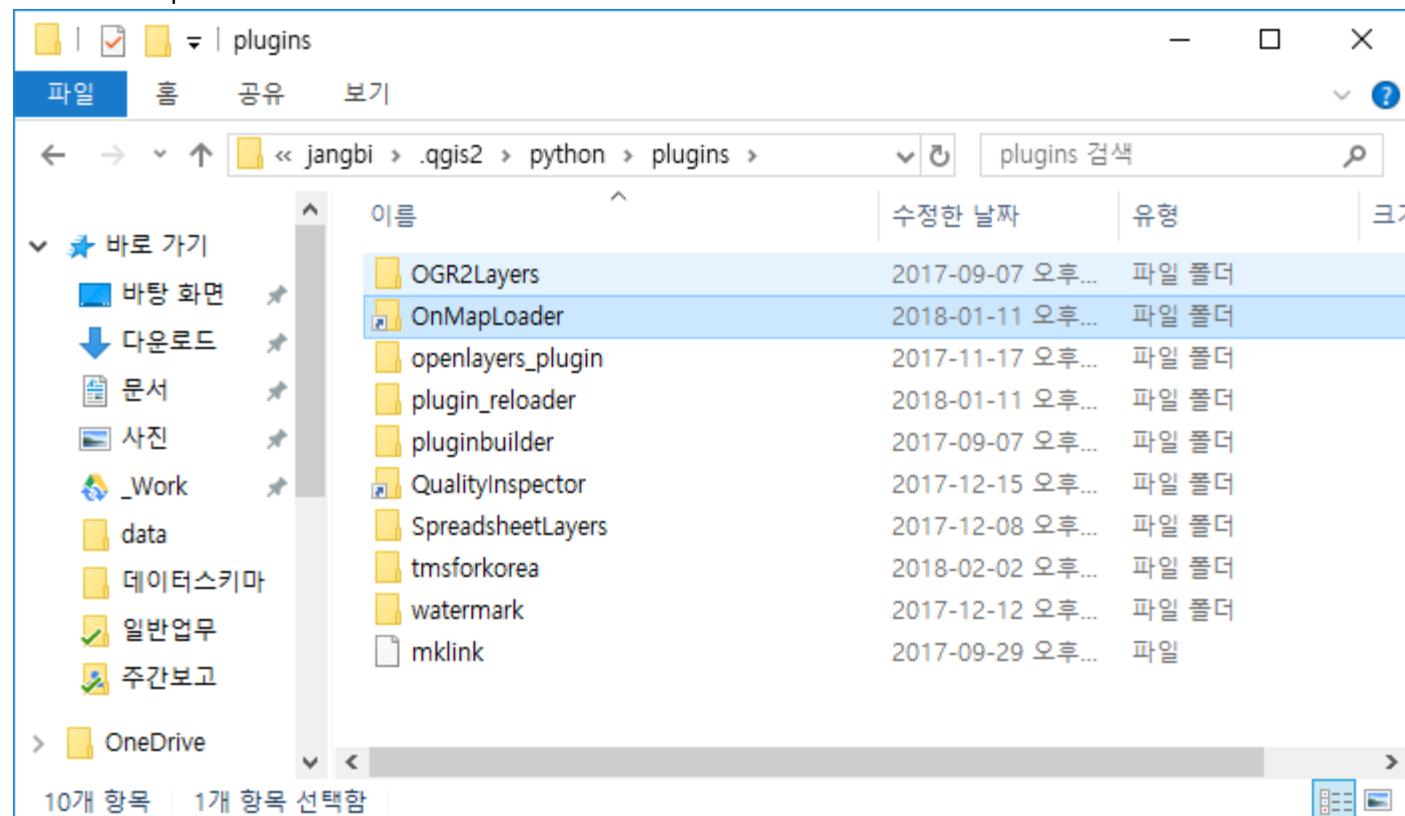
공식 플러그인이 아니라 QGIS 플러그인 관리자로 바로 설치는 못하지만, 우리는 사용할 수 있습니다.  
개발자니까요!

설치 안내에 나온 내용대로 해 봅시다. 우선 다음에서 설치를 위한 파일을 받습니다.

[https://github.com/Gaia3D/OnMapLoader/raw/master/release/OnMapLoader\\_1.2.zip](https://github.com/Gaia3D/OnMapLoader/raw/master/release/OnMapLoader_1.2.zip)

이를 사용자 폴더의 .qgis2/python/plugins에 압축을 풀어줍니다.

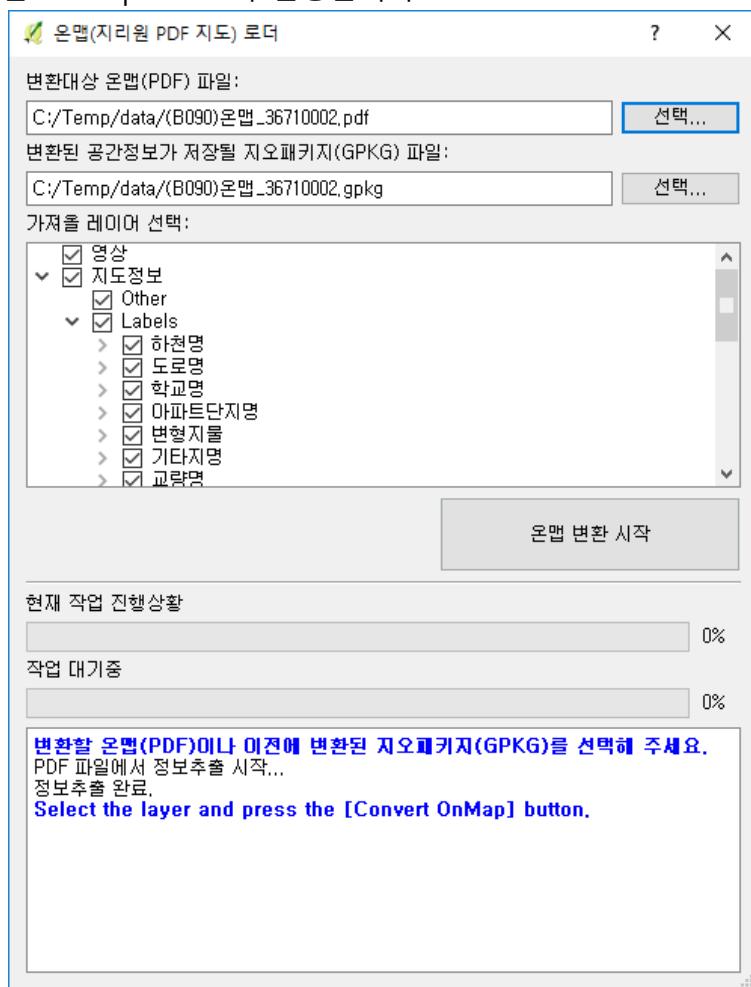
이 때 OnMapLoader 폴더가 2중으로 생기지 않도록 주의해야 합니다.



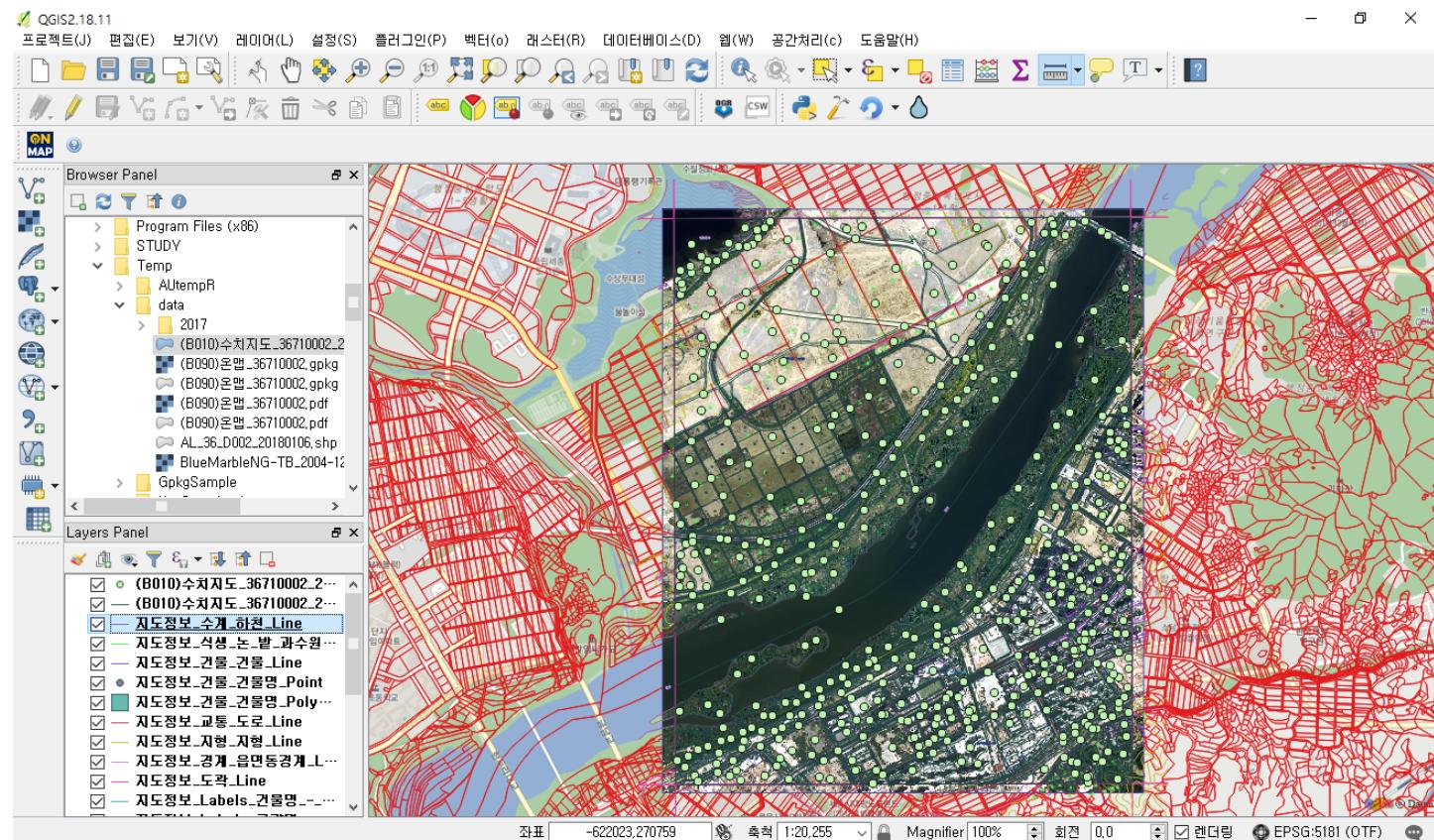
QGIS를 재실행 하면 OnMap 아이콘이 보입니다.



이 버튼을 누르면 온맵 PDF를 읽어 공간정보로 만들고 이를 GeoPackage라는 포맷으로 저장해 나중에 더 빨리 사용할 수 있게 해주는 OnMapLoader가 실행됩니다.

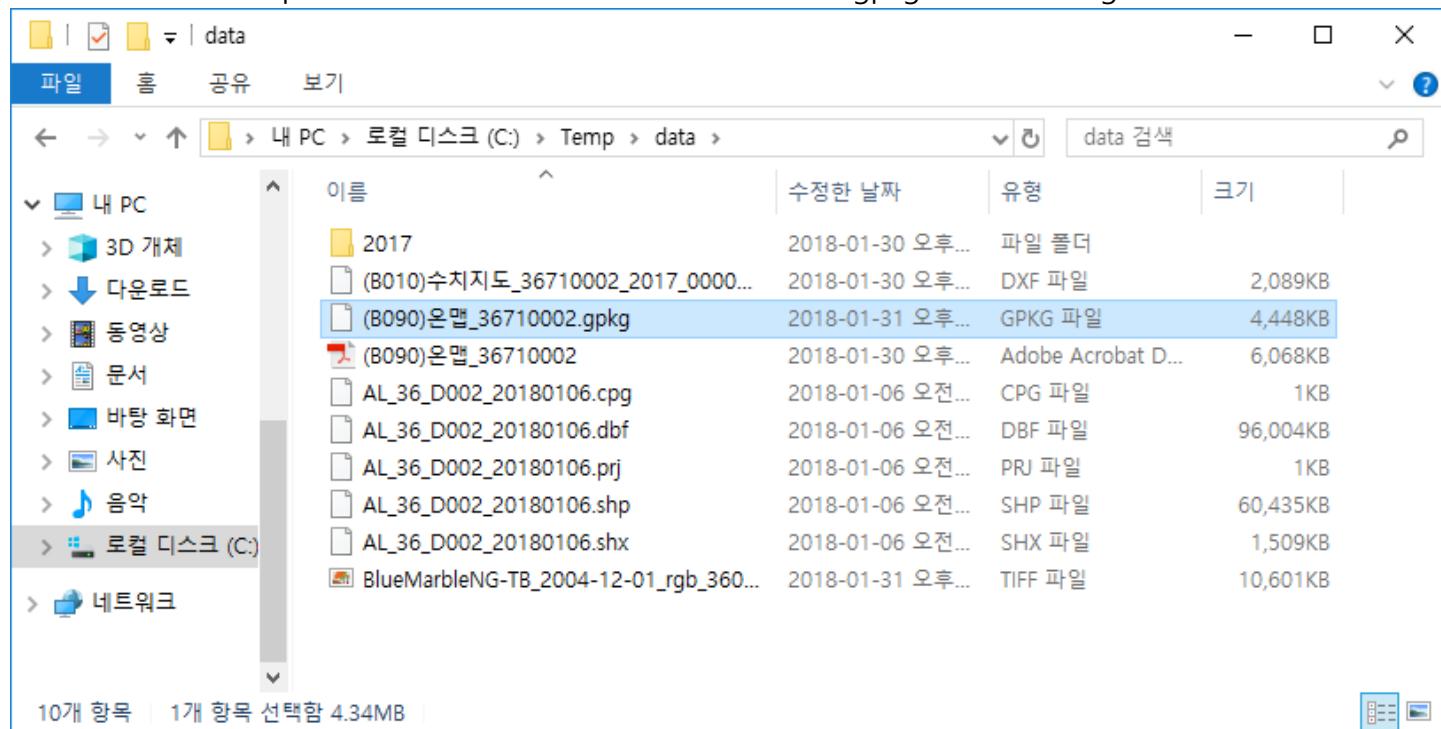


변환에 시간이 꽤 걸립니다. 다 변환이 되면 온맵에 있던 지형도와 정사영상이 QGIS에서 재위치에 올라오는 것을 확인할 수 있습니다.



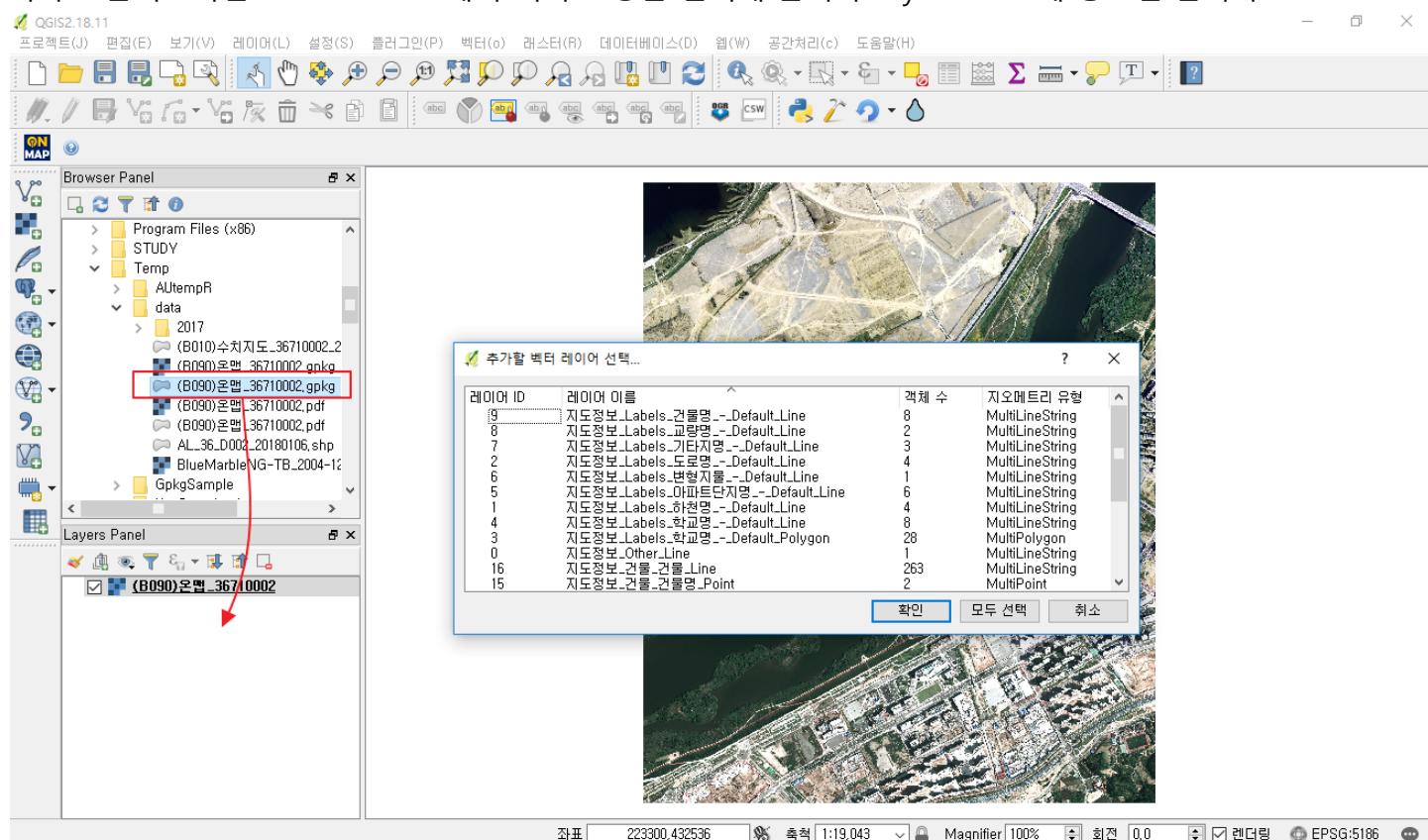
사실 이 OnMapLoader를 소개해 드린 것은 GeoPackage라는 포맷을 홍보하기 위한 것입니다. 다시 아이콘을 눌러 OnMapLoader를 실행해 보면 이미 만들어진 지오패키지 파일을 재활용할지 물어보고, 여기에서 [예]를 선택하면 변환과정 없이 저장된 파일을 읽어옵니다.

제 폴더에 보니 온맵 pdf 파일과 확장자만 다른 파일이 있네요 \*.gpkg 가 GeoPackage의 확장자입니다.



약 4.5 메가 정도의 크기이고, 여러 레이어와 정사영상까지 들어있는데 단 한개의 파일이네요!  
ESRI Shape 처럼 한 레이어도 여러 파일이 아니고요!

이 GeoPackage는 QGIS에서 별도 플러그인 없이도 잘 열립니다.  
탐색기에서 (B090)온맵\_36710002.gpkg 파일을 끌어다가 QGIS에 놓아 보세요.  
먼저 정사영상만 딸랑 들어오긴 하지만, 좌표계도 물어보지 않고 위치도 정확히 맞네요.  
벡터로 불러오려면 Browser Panel에서 벡터 모양을 선택해 끌어다 Layers Panel에 놓으면 됩니다.



한 GeoPackage에 여러 레이어가 있기에 어떤 레이어를 불러올지 물어보네요.  
모두 선택하고 [확인]하면 역시 좌표계 물어보지 않고, 빠른 속도로 올바른 위치에 들어옵니다.

GeoPackage는 현재 공간정보 전문가들이 강력히 추천하는 국제표준 공간정보 저장 포맷입니다. 흔히 사용하는 ESRI Shape을 멸종시키고 GeoPackage를 사용자는 운동까지 세계 각 나라에서 벌어지고 있고, 본 강의의 이 챕터도 ESRI Shape을 멸종 운동의 일부입니다.

GeoPackage에 대해 좀 더 자세히 알아보겠습니다.

GeoPackage는 OGC에서 정한 공간데이터 저장, 교환, 서비스를 위한 국제표준이며 ArcGIS 10.3, QGIS 2.10.1, GeoServer 2.5, GDAL 1.11, GeoTools 11 이상의 버전에서 사용할 수 있습니다.  
관련 상세 정보는 다음 링크에서 찾으실 수 있습니다.

<http://www.geopackage.org/>

특징만 정리해 보겠습니다.

- ESRI Shape 처럼 여러 파일이 아닌 한개의 파일이면 된다.
- 한 파일에 여러 레이어를 담을 수 있다.
- 벡터 뿐 아니라 래스터 데이터도 담을 수 있다.
- 좌표계 정보 등의 메타데이터도 함께 담겨 혼란이 없다.
- 한 레이어에  $2^{64}$ (약 1.8e+19) 개의 행을 담을 수 있다.
- 물리적으로 허용하면 파일크기 제한이 없다.
- 속성 컬럼의 이름 길이에 제한이 없다.
- 공간데이터가 없는 속성 테이블도 같이 담을 수 있다.
- 내부적으로 인덱스를 지원해 빠른 속도의 조회가 가능하다.
- 단지 교환용 포맷 뿐 아니라 직접 서비스에 사용해도 높은 성능을 낼 수 있다.

이렇게 좋은 포맷 안 쓸 이유 없겠지요?

아직 잘 알려지지 않은 듯보糟 포맷이라 불안해 하신다면, 그렇지 않다 자신있게 말씀드릴 수 있습니다.

사실 GeoPackage는 새로운 포맷이 아니라 스마트폰에서도 흔히 쓰는 SQLite라는 파일기반 DB에 공간정보를 담는 규약만 정의한 것입니다.

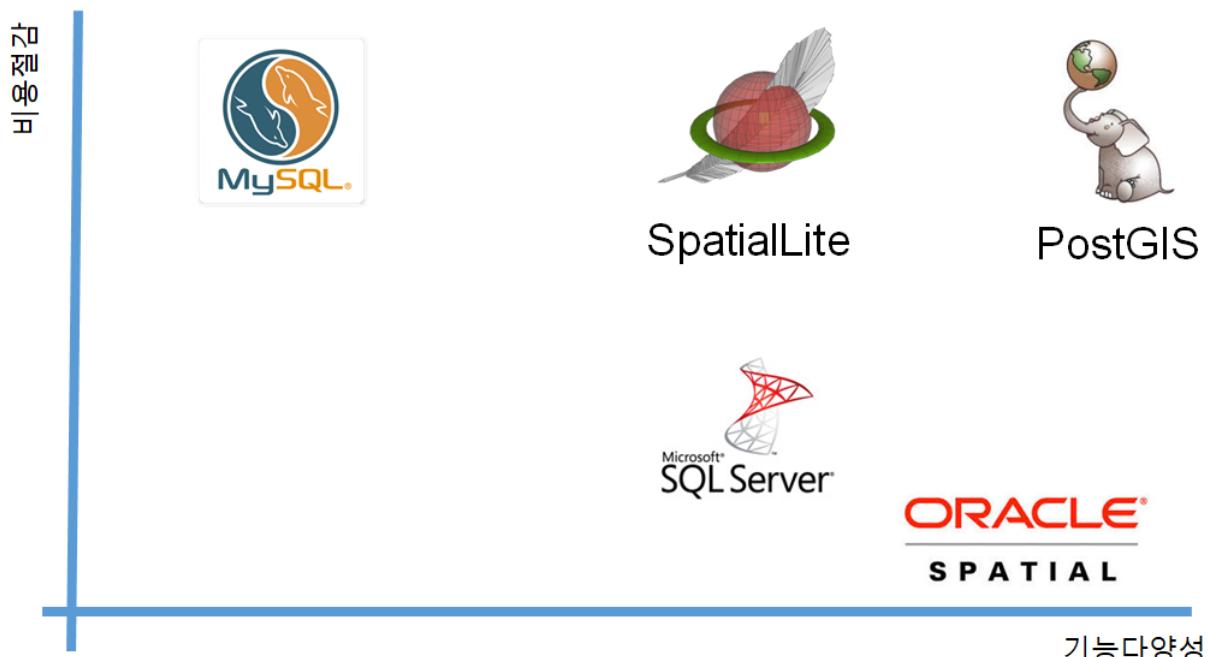
때문에 Java, C++, Python 등 대부분의 주요 언어에서 기본적으로 다룰 수 있는 오래 검증되고 흔히 쓰이고 있는 기술기반을 가지고 있습니다.

## 4x01 PostgreSQL이 공간정보를 다룰 수 있게 준비하기

이제 공간정보를 DBMS에 넣어 효과적으로 관리하는 방법을 배워보도록 하겠습니다.

샘플데이터 정도는 ESRI Shape 파일로도 잘 서비스할 수 있지만, 상업적인 규모의 서비스를 하기 위해서는 공간정보를 지원하는 DBMS를 이용하는 것이 좋습니다.

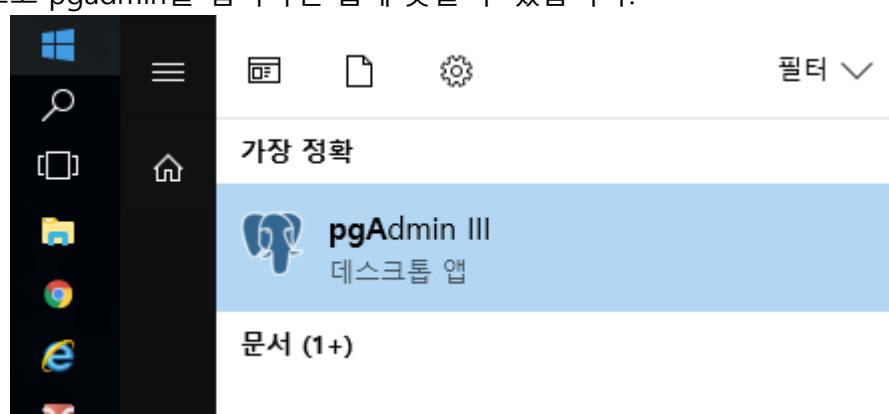
공간 DBMS로 오늘 배우는 것은 PostGIS 이지만, 이 외에도 Oracle Spatial, MS SQL Server, MySQL, SpatialLite 등 여러가지 주요 DBMS가 공간정보를 다룰 수 있습니다.



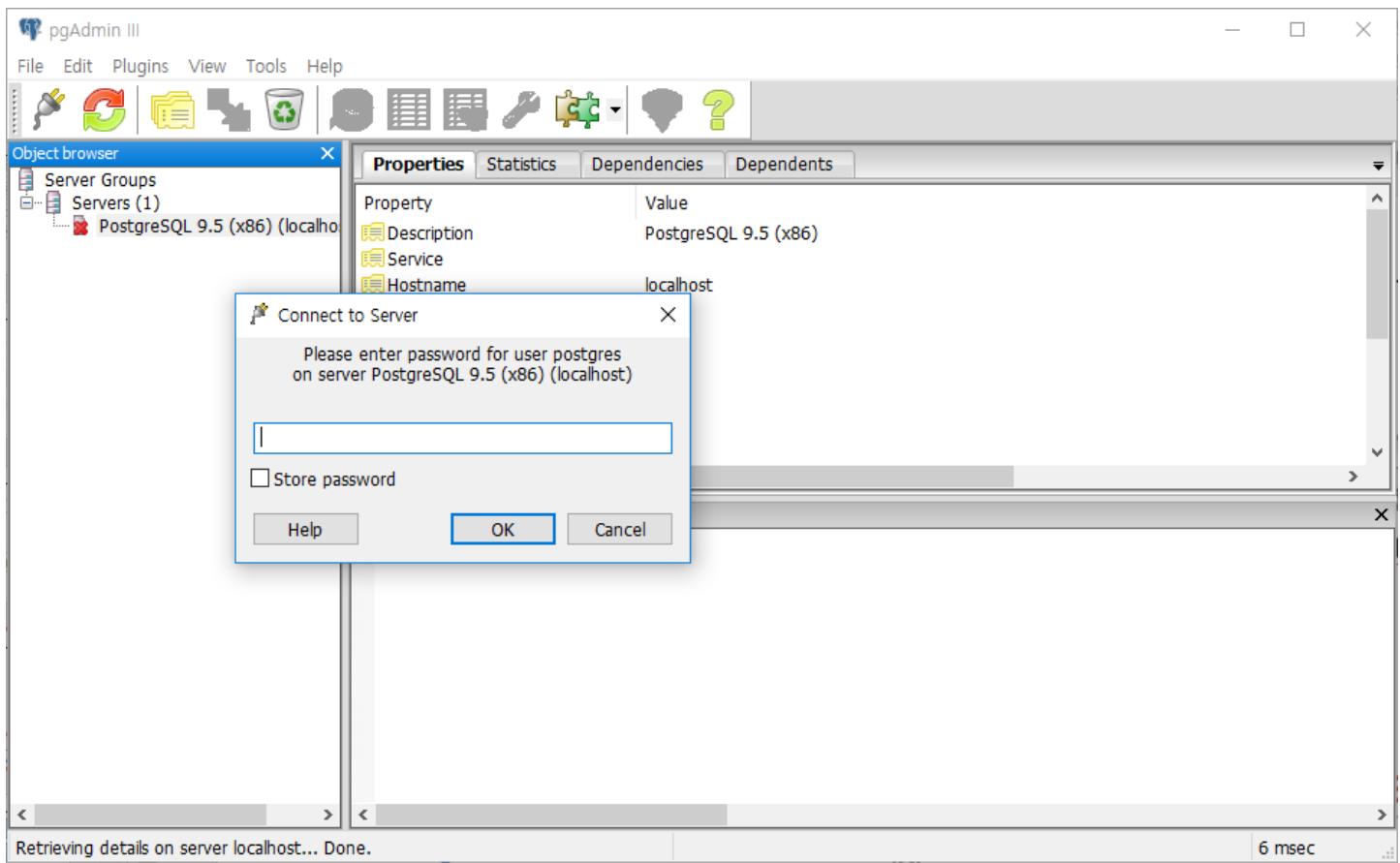
출처: [https://www.slideshare.net/gis\\_todd/postgis-and-spatial-sql](https://www.slideshare.net/gis_todd/postgis-and-spatial-sql)

먼저 PostGIS의 기반 DBMS인 PostgreSQL에 확장 기능인 PostGIS가 적용되도록 하겠습니다.

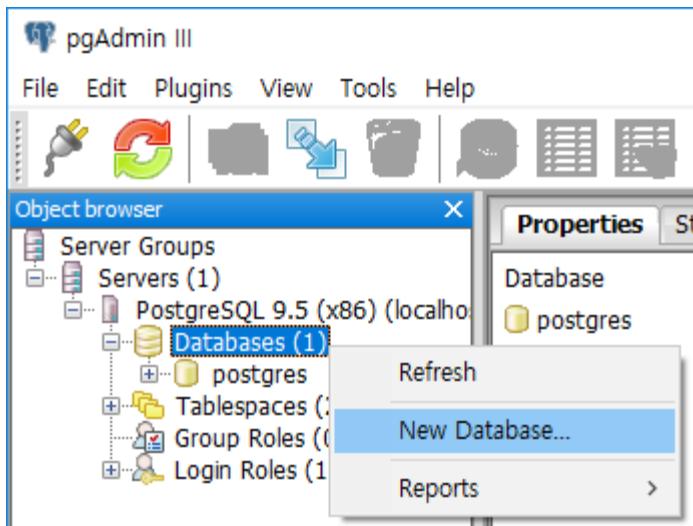
PostgreSQL을 관리하는 편리한 도구인 pgAdmin 3를 사용하겠습니다.  
[윈도우] 버튼을 누르고 pgadmin을 입력하면 쉽게 찾을 수 있습니다.



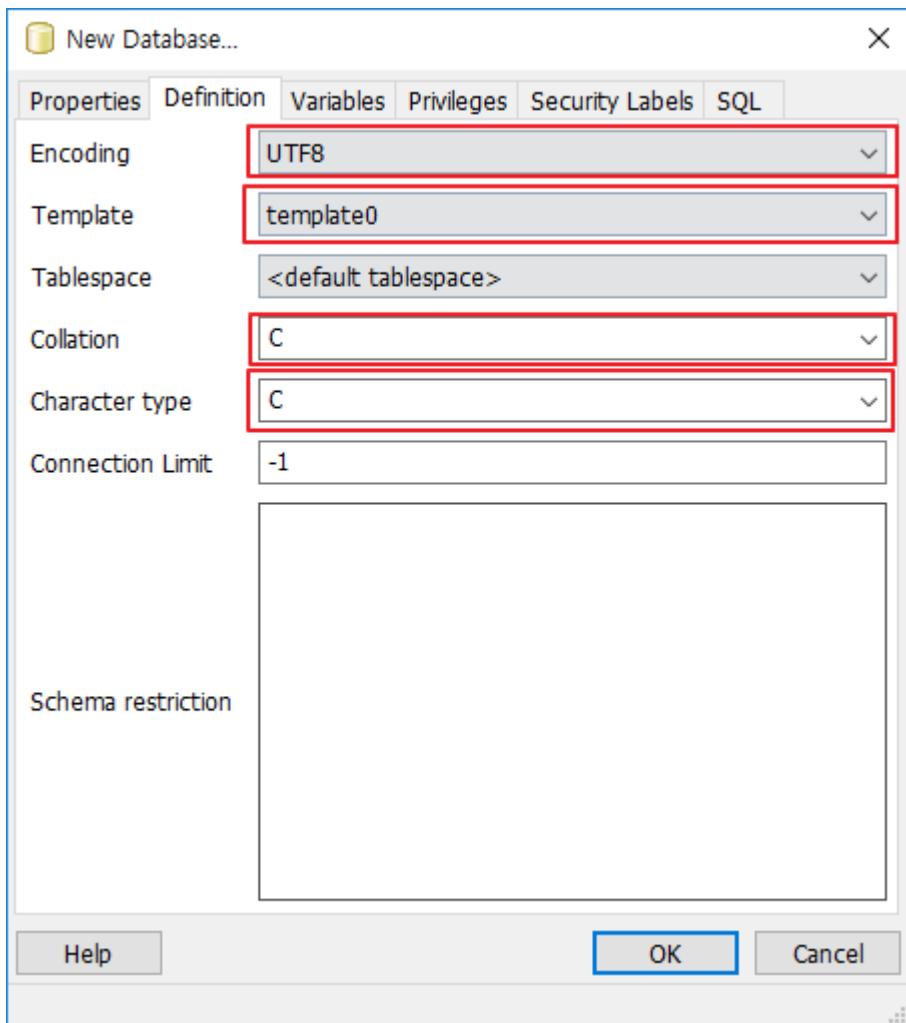
pgAdmin III 버튼을 눌러 시작하십시오.



Server Groups에 있는 PostgreSQL 9.5 (localhost)를 더블클릭하고 postgres 유저의 암호(postgres)를 입력하세요.



접속이 된 후, Databases 항목을 오른쪽 클릭하고 [New Database...] 메뉴를 선택합니다.



새로 만들 Database의 이름을 osgeo라 하겠습니다.

Properties 탭의 Name 항목에 osgeo를 입력해 주세요.

그리고 Definition 탭으로 가셔서 Encoding: UTF-8, Template: template0, Collation: C, Character type: C 를 선택하시고 [OK]를 눌러 주세요.

만약 Collation, Character type에 C가 아닌 다른 값을 선택하면 한글의 정렬과 검색에서 문제가 생길 수도 있으니 주의하세요.

이렇게 만들어진 Database가 바로 공간자료를 다룰 수 있는 것은 아니고 한 단계를 더 거쳐야 합니다.

Databases 아래의 방금 만든 osgeo를 선택하고, [SQL] 버튼을 눌러 Query 창을 엽니다.

```
Query - osgeo on postgres@localhost:5432 *
File Edit Query Favourites Macros View Help
SQL Editor Graphical Query Builder
Previous queries
create extension postgis;
```

SQL Editor에 `create extension postgis;` 라 입력하고 Run 버튼을 누르거나 [F5] 평선키를 눌러 실행합니다. 이제 이 osgeo Database는 공간정보를 다룰 수 있게 되었습니다.

다사 pgAdmin의 메인 창으로 가서 Databases를 선택후 [F5]를 눌러 상태를 갱신해 보면 공간정보를 다룰 수 있는 상태가 되었음을 확인할 수 있습니다.

Property	Value
Name	osgeo
OID	18187
Owner	postgres
ACL	
Tablespace	pg_default
Default tablespace	pg_default
Encoding	UTF8
Collation	C
Character type	C

SQL pane

```
-- Database: osgeo

-- DROP DATABASE osgeo;

CREATE DATABASE osgeo
  WITH OWNER = postgres
    ENCODING = 'UTF8'
      TABLESPACE = pg_default
        LC_COLLATE = 'C'
```

Extension에 postgis가 들어가 있고, Schemas / public / Functions에 평선품수가 1000개가 넘게 되고,, Tables에 spatial\_ref\_sys라는 테이블이 생겨 있음을 보면, 아~ 공간정보를 담을 준비가 되었구나 하시면 됩니다.

spatial\_ref\_sys 테이블은 좌표계 정보를 담고 있는 매우 중요한 테이블입니다.  
내용을 잠깐 살펴보겠습니다.

spatial\_ref\_sys 테이블을 선택 후 툴바에서 표 모양 아이콘을 누르면 내용을 볼 수 있습니다.

Edit Data - PostgreSQL 9.5 (x86) (localhost:5432) - osgeo - public.spatial\_ref\_sys

File Edit View Tools Help

No limit

	srid [PK] integer	auth_name character varying(256)	auth_srid integer	srttext character varying(2048)	proj4text character varying(2048)
1	2000	EPSG	2000	PROJCS["Anguilla 1957 / British West Indies Grid",GEOGCS["Anguilla 1957",DATUM["Anguilla_1957+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.9995000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-62],PARAMETER["scale_factor",0.9995],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.9995000000000000
2	2001	EPSG	2001	PROJCS["Antigua 1943 / British West Indies Grid",GEOGCS["Antigua 1943",DATUM["Antigua_1943+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.9995000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-62],PARAMETER["scale_factor",0.9995],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.9995000000000000
3	2002	EPSG	2002	PROJCS["Dominica 1945 / British West Indies Grid",GEOGCS["Dominica 1945",DATUM["Dominica_1945+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.9995000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-62],PARAMETER["scale_factor",0.9995],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.9995000000000000
4	2003	EPSG	2003	PROJCS["Grenada 1953 / British West Indies Grid",GEOGCS["Grenada 1953",DATUM["Grenada_1953+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.9995000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-62],PARAMETER["scale_factor",0.9995],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.9995000000000000
5	2004	EPSG	2004	PROJCS["Montserrat 1958 / British West Indies Grid",GEOGCS["Montserrat 1958",DATUM["Montserrat_1958+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.9995000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-62],PARAMETER["scale_factor",0.9995],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.9995000000000000
6	2005	EPSG	2005	PROJCS["St. Kitts 1955 / British West Indies Grid",GEOGCS["St. Kitts 1955",DATUM["St_Kitts_1955+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.9995000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-62],PARAMETER["scale_factor",0.9995],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.9995000000000000
7	2006	EPSG	2006	PROJCS["St. Lucia 1955 / British West Indies Grid",GEOGCS["St. Lucia 1955",DATUM["St_Lucia_1955+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.9995000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-62],PARAMETER["scale_factor",0.9995],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.9995000000000000
8	2007	EPSG	2007	PROJCS["St. Vincent 1945 / British West Indies Grid",GEOGCS["St. Vincent 1945",DATUM["St_Vincent_1945+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.9995000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-62],PARAMETER["scale_factor",0.9995],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.9995000000000000
9	2008	EPSG	2008	PROJCS["NAD27(CGQ77) / SCoPQ zone 2 (deprecated)",GEOGCS["NAD27(CGQ77)",DATUM["North_America_1983+proj=tmerc +lat_0=0 +lon_0=-55.5 +k=0.9999 +x_0.0000000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-55.5],PARAMETER["scale_factor",0.9999],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-55.5 +k=0.9999 +x_0.0000000000000000
10	2009	EPSG	2009	PROJCS["NAD27(CGQ77) / SCoPQ zone 3",GEOGCS["NAD27(CGQ77)",DATUM["North_American_Datum_1983+proj=tmerc +lat_0=0 +lon_0=-58.5 +k=0.9999 +x_0.0000000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-58.5],PARAMETER["scale_factor",0.9999],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-58.5 +k=0.9999 +x_0.0000000000000000
11	2010	EPSG	2010	PROJCS["NAD27(CGQ77) / SCoPQ zone 4",GEOGCS["NAD27(CGQ77)",DATUM["North_American_Datum_1983+proj=tmerc +lat_0=0 +lon_0=-61.5 +k=0.9999 +x_0.0000000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-61.5],PARAMETER["scale_factor",0.9999],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-61.5 +k=0.9999 +x_0.0000000000000000
12	2011	EPSG	2011	PROJCS["NAD27(CGQ77) / SCoPQ zone 5",GEOGCS["NAD27(CGQ77)",DATUM["North_American_Datum_1983+proj=tmerc +lat_0=0 +lon_0=-64.5 +k=0.9999 +x_0.0000000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-64.5],PARAMETER["scale_factor",0.9999],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-64.5 +k=0.9999 +x_0.0000000000000000
13	2012	EPSG	2012	PROJCS["NAD27(CGQ77) / SCoPQ zone 6",GEOGCS["NAD27(CGQ77)",DATUM["North_American_Datum_1983+proj=tmerc +lat_0=0 +lon_0=-67.5 +k=0.9999 +x_0.0000000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-67.5],PARAMETER["scale_factor",0.9999],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-67.5 +k=0.9999 +x_0.0000000000000000
14	2013	EPSG	2013	PROJCS["NAD27(CGQ77) / SCoPQ zone 7",GEOGCS["NAD27(CGQ77)",DATUM["North_American_Datum_1983+proj=tmerc +lat_0=0 +lon_0=-70.5 +k=0.9999 +x_0.0000000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-70.5],PARAMETER["scale_factor",0.9999],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-70.5 +k=0.9999 +x_0.0000000000000000
15	2014	EPSG	2014	PROJCS["NAD27(CGQ77) / SCoPQ zone 8",GEOGCS["NAD27(CGQ77)",DATUM["North_American_Datum_1983+proj=tmerc +lat_0=0 +lon_0=-73.5 +k=0.9999 +x_0.0000000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-73.5],PARAMETER["scale_factor",0.9999],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-73.5 +k=0.9999 +x_0.0000000000000000
16	2015	EPSG	2015	PROJCS["NAD27(CGQ77) / SCoPQ zone 9",GEOGCS["NAD27(CGQ77)",DATUM["North_American_Datum_1983+proj=tmerc +lat_0=0 +lon_0=-76.5 +k=0.9999 +x_0.0000000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-76.5],PARAMETER["scale_factor",0.9999],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-76.5 +k=0.9999 +x_0.0000000000000000
17	2016	EPSG	2016	PROJCS["NAD27(CGQ77) / SCoPQ zone 10",GEOGCS["NAD27(CGQ77)",DATUM["North_American_Datum_1983+proj=tmerc +lat_0=0 +lon_0=-79.5 +k=0.9999 +x_0.0000000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-79.5],PARAMETER["scale_factor",0.9999],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-79.5 +k=0.9999 +x_0.0000000000000000
18	2017	EPSG	2017	PROJCS["NAD27(76) / MTM zone 8",GEOGCS["NAD27(76)",DATUM["North_American_Datum_1927_1976+proj=tmerc +lat_0=0 +lon_0=-73.5 +k=0.9999 +x_0.0000000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-73.5],PARAMETER["scale_factor",0.9999],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-73.5 +k=0.9999 +x_0.0000000000000000
19	2018	EPSG	2018	PROJCS["NAD27(76) / MTM zone 9",GEOGCS["NAD27(76)",DATUM["North_American_Datum_1927_1976+proj=tmerc +lat_0=0 +lon_0=-76.5 +k=0.9999 +x_0.0000000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-76.5],PARAMETER["scale_factor",0.9999],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-76.5 +k=0.9999 +x_0.0000000000000000
20	2019	EPSG	2019	PROJCS["NAD27(76) / MTM zone 10",GEOGCS["NAD27(76)",DATUM["North_American_Datum_1927_1976+proj=tmerc +lat_0=0 +lon_0=-79.5 +k=0.9999 +x_0.0000000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-79.5],PARAMETER["scale_factor",0.9999],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-79.5 +k=0.9999 +x_0.0000000000000000
21	2020	EPSG	2020	PROJCS["NAD27(76) / MTM zone 11",GEOGCS["NAD27(76)",DATUM["North_American_Datum_1927_1976+proj=tmerc +lat_0=0 +lon_0=-82.5 +k=0.9999 +x_0.0000000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-82.5],PARAMETER["scale_factor",0.9999],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-82.5 +k=0.9999 +x_0.0000000000000000
22	2021	EPSG	2021	PROJCS["NAD27(76) / MTM zone 12",GEOGCS["NAD27(76)",DATUM["North_American_Datum_1927_1976+proj=tmerc +lat_0=0 +lon_0=-81 +k=0.9999 +x_0.0000000000000000"],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["degree",1, "angle"]],PROJECTION["Transverse_Mercator"],PARAMETER["central_meridian",-81],PARAMETER["scale_factor",0.9999],PARAMETER["false_easting",0],PARAMETER["false_northing",0]]	+proj=tmerc +lat_0=0 +lon_0=-81 +k=0.9999 +x_0.0000000000000000

Scratch pad

srid 컬럼에 숫자들이 보이고, auth\_name에 EPSG라는 글자들이 보입니다. proj4text라는 컬럼에도 어디서 본듯한 내용들이 들어있네요.

앞 시간에 배운 좌표계 정보가 들어 있다는 것을 느끼실 수 있지요?

QGIS에서 문제가 되었던 EPSG:5174 좌표계 정보를 조회해 보겠습니다.

Query 창에 SQL을 입력해 조회할 수 있습니다.

다음 SQL을 입력하고 실행해 봅시다.

```
select proj4text from spatial_ref_sys where srid = 5174;
```

Query - osgeo on postgres@localhost:5432 \*

File Edit Query Favourites Macros View Help

SQL Editor Graphical Query Builder

Previous queries

```
select proj4text
from spatial_ref_sys
where srid = 5174;
```

Output pane

Data Output Explain Messages History

proj4text character varying(2048)
+proj=tmerc +lat_0=38 +lon_0=127.0028902777778 +k=1 +x_0=200000 +y_0=500000 +ellps=bessel +units=m +no_defs

OK. DOS Ln 1, Col 17, Ch 17 1 row. 18 msec

결과를 보면 여기서도 +towgs84 인자가 누락되어 있음을 확인할 수 있습니다.

이를 바로잡기 위한 SQL이 미리 만들어져 있어 어렵지 않게 바로잡을 수 있습니다.  
다음 링크로 가 보십시오.

<http://osgeo.kr/205>

이 페이지에서 postgis\_korea\_epsg\_towgs84.sql을 다운로드 받습니다.

Query 창에서 열기 버튼을 눌러 이 SQL 파일을 열고 [F5]눌러 실행하면 됩니다.

```
설명 : PostGIS Korean 좌표계 등록 스크립트 예 2 - TOWGS84 파라미터 적용
proj   : +towgs84=-115.80,474.99,674.11,1.16,-2.31,-1.63,6.43
WKT    : TOWGS84[-115.80,474.99,674.11,1.16,-2.31,-1.63,6.43]

2096=PROJCS["Korean 1985 / East Belt", GEOGCS["Korean 1985", DATUM["Korean 1985", SPHEROID["Bessel 1841", 6378137, 299004.9, AUTHORITY["EPSG", "7015"]], PRIMEM["Greenwich", 0, AUTHORITY["EPSG", "8901"]], AUTHORITY["EPSG", "4293"]], PROJECTION["Transverse_Mercator"], PARAMETER["latitude_of_origin", 37.5, AUTHORITY["EPSG", "8901"]], PARAMETER["central_meridian", 135, AUTHORITY["EPSG", "8901"]], PARAMETER["scale_factor", 0.9999, AUTHORITY["EPSG", "8901"]], PARAMETER["false_easting", 500000, AUTHORITY["EPSG", "8901"]], PARAMETER["false_northing", 0, AUTHORITY["EPSG", "8901"]], AUTHORITY["EPSG", "2096"]]
2097=PROJCS["Korean 1985 / Central Belt", GEOGCS["Korean 1985", DATUM["Korean 1985", SPHEROID["Bessel 1841", 6378137, 299004.9, AUTHORITY["EPSG", "7015"]], PRIMEM["Greenwich", 0, AUTHORITY["EPSG", "8901"]], AUTHORITY["EPSG", "4293"]], PROJECTION["Transverse_Mercator"], PARAMETER["latitude_of_origin", 37.5, AUTHORITY["EPSG", "8901"]], PARAMETER["central_meridian", 127, AUTHORITY["EPSG", "8901"]], PARAMETER["scale_factor", 0.9999, AUTHORITY["EPSG", "8901"]], PARAMETER["false_easting", 500000, AUTHORITY["EPSG", "8901"]], PARAMETER["false_northing", 0, AUTHORITY["EPSG", "8901"]], AUTHORITY["EPSG", "2097"]]
2098=PROJCS["Korean 1985 / West Belt", GEOGCS["Korean 1985", DATUM["Korean 1985", SPHEROID["Bessel 1841", 6378137, 299004.9, AUTHORITY["EPSG", "7015"]], PRIMEM["Greenwich", 0, AUTHORITY["EPSG", "8901"]], AUTHORITY["EPSG", "4293"]], PROJECTION["Transverse_Mercator"], PARAMETER["latitude_of_origin", 37.5, AUTHORITY["EPSG", "8901"]], PARAMETER["central_meridian", 119, AUTHORITY["EPSG", "8901"]], PARAMETER["scale_factor", 0.9999, AUTHORITY["EPSG", "8901"]], PARAMETER["false_easting", 500000, AUTHORITY["EPSG", "8901"]], PARAMETER["false_northing", 0, AUTHORITY["EPSG", "8901"]], AUTHORITY["EPSG", "2098"]]
```

srid	auth_name	auth_srid	srtext	proj4text
1	2096	EPSG	2096 PROJCS["Korean 1985 / East Belt", GEOGCS["Korean 1985", DATUM["Korean 1985", SPHEROID["Bessel 1841", 6378137, 299004.9, AUTHORITY["EPSG", "7015"]], PRIMEM["Greenwich", 0, AUTHORITY["EPSG", "8901"]], AUTHORITY["EPSG", "4293"]], PROJECTION["Transverse_Mercator"], PARAMETER["latitude_of_origin", 37.5, AUTHORITY["EPSG", "8901"]], PARAMETER["central_meridian", 135, AUTHORITY["EPSG", "8901"]], PARAMETER["scale_factor", 0.9999, AUTHORITY["EPSG", "8901"]], PARAMETER["false_easting", 500000, AUTHORITY["EPSG", "8901"]], PARAMETER["false_northing", 0, AUTHORITY["EPSG", "8901"]], AUTHORITY["EPSG", "2096"]]	+proj=tmerc +lat 0+38 +lon 0+12!
2	2097	EPSG	2097 PROJCS["Korean 1985 / Central Belt", GEOGCS["Korean 1985", DATUM["Korean 1985", SPHEROID["Bessel 1841", 6378137, 299004.9, AUTHORITY["EPSG", "7015"]], PRIMEM["Greenwich", 0, AUTHORITY["EPSG", "8901"]], AUTHORITY["EPSG", "4293"]], PROJECTION["Transverse_Mercator"], PARAMETER["latitude_of_origin", 37.5, AUTHORITY["EPSG", "8901"]], PARAMETER["central_meridian", 127, AUTHORITY["EPSG", "8901"]], PARAMETER["scale_factor", 0.9999, AUTHORITY["EPSG", "8901"]], PARAMETER["false_easting", 500000, AUTHORITY["EPSG", "8901"]], PARAMETER["false_northing", 0, AUTHORITY["EPSG", "8901"]], AUTHORITY["EPSG", "2097"]]	+proj=tmerc +lat 0+38 +lon 0+12!
3	2098	EPSG	2098 PROJCS["Korean 1985 / West Belt", GEOGCS["Korean 1985", DATUM["Korean 1985", SPHEROID["Bessel 1841", 6378137, 299004.9, AUTHORITY["EPSG", "7015"]], PRIMEM["Greenwich", 0, AUTHORITY["EPSG", "8901"]], AUTHORITY["EPSG", "4293"]], PROJECTION["Transverse_Mercator"], PARAMETER["latitude_of_origin", 37.5, AUTHORITY["EPSG", "8901"]], PARAMETER["central_meridian", 119, AUTHORITY["EPSG", "8901"]], PARAMETER["scale_factor", 0.9999, AUTHORITY["EPSG", "8901"]], PARAMETER["false_easting", 500000, AUTHORITY["EPSG", "8901"]], PARAMETER["false_northing", 0, AUTHORITY["EPSG", "8901"]], AUTHORITY["EPSG", "2098"]]	+proj=tmerc +lat 0+38 +lon 0+12!

이제 PostGIS에서는 한국 좌표계들이 정상적으로 좌표계 변환이 됩니다.

이렇게 PostGIS를 활성화시키고 한국 좌표계를 정정하는 작업은 새로운 Database를 만들때만 하면 됩니다.

## 4x02 PostGIS에 공간정보 올리기

준비가 끝났으니 이제 공간정보를 올려보겠습니다.

먼저 실습에 사용할 파일을 받겠습니다.

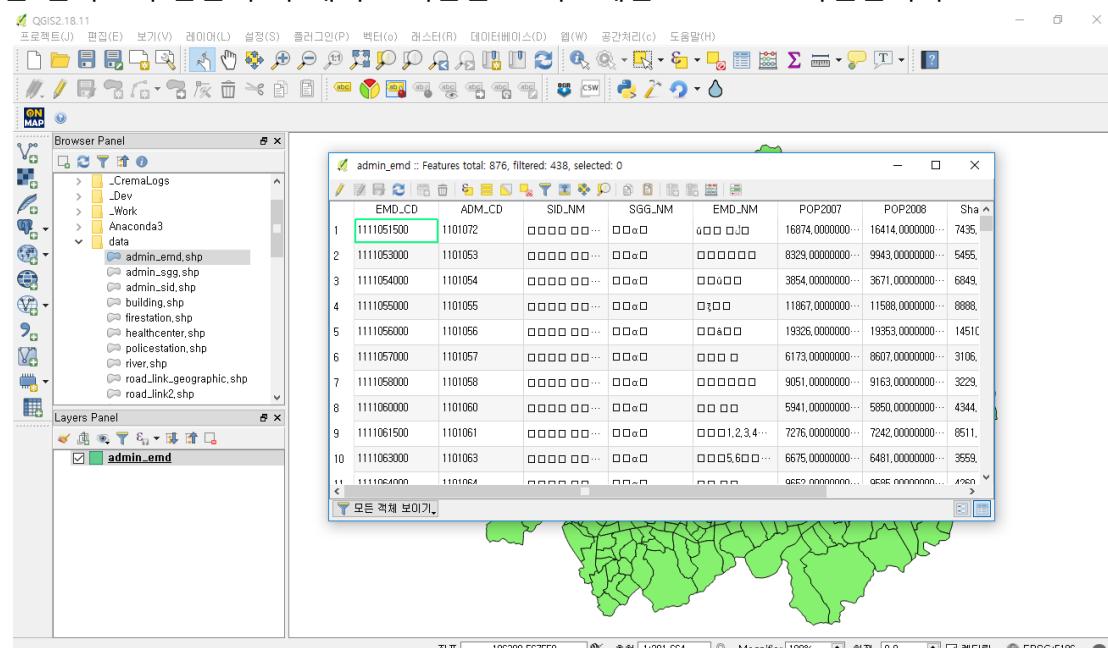
다음 링크의 자료를 받아 주세요.

[https://github.com/Gaia3D/workshop/raw/master/20171208\\_%EC%84%9C%EC%9A%B8%EC%97%B0\\_%EA%B3%B5%EA%B0%84SQL/data.zip](https://github.com/Gaia3D/workshop/raw/master/20171208_%EC%84%9C%EC%9A%B8%EC%97%B0_%EA%B3%B5%EA%B0%84SQL/data.zip)

다운로드 받은 자료를 C:\data 폴더에 풀어주세요.

먼저 한글 코드페이지를 확인해 보기 위해 QGIS에서 자료를 열어보겠습니다.

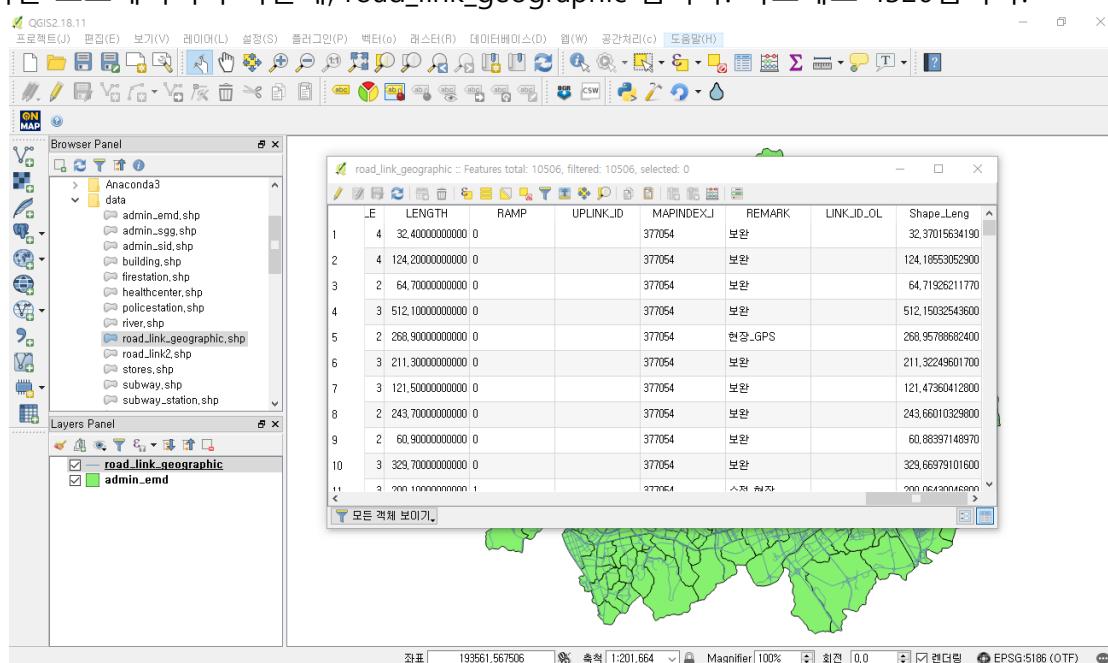
admin\_emd를 열어보니 한글이 다 깨져 보이는군요. 좌표계는 5186으로 확인됩니다.



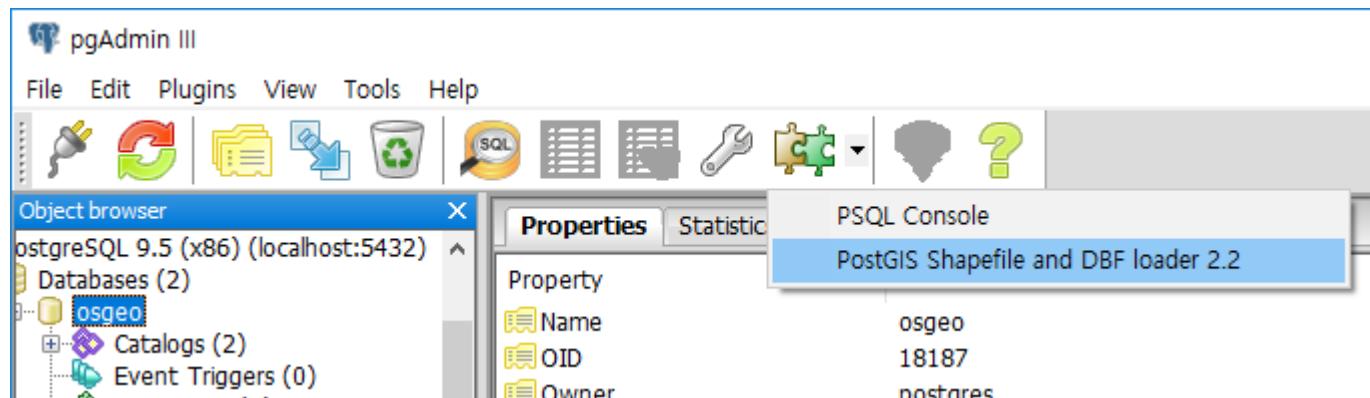
데이터 소스 코드화 값을 'cp949'로 바꾸니 정상적으로 보입니다.

이 데이터들은 대부분 윈도우에서 사용하는 cp949 코드페이지를 사용하고 있습니다.

단 한 레이어만 코드페이지가 다른데, road\_link\_geographic 입니다. 좌표계도 4326입니다.

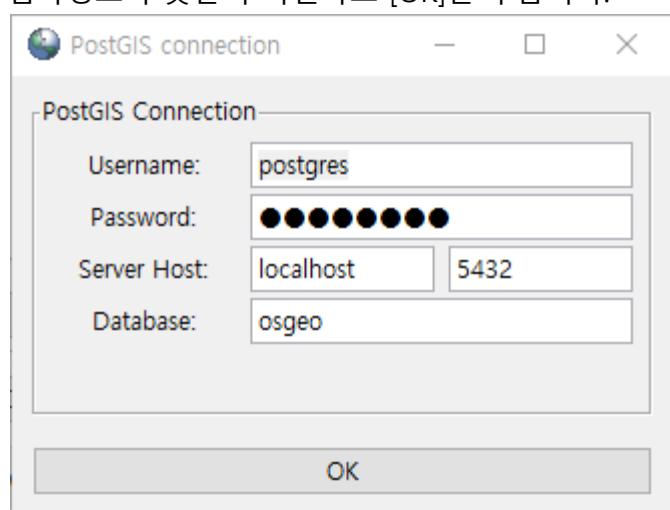


공간정보를 PostGIS에 올릴 수 있는 도구는 여러가지가 있습니다.  
가장 쉽고 강력한 Shapefile and DBF loader를 사용해 보겠습니다.



pgAdmin에서 툴바의 PostGIS Shapefile and DBF loader 2.2 를 선택하시면 실행하실 수 있습니다.

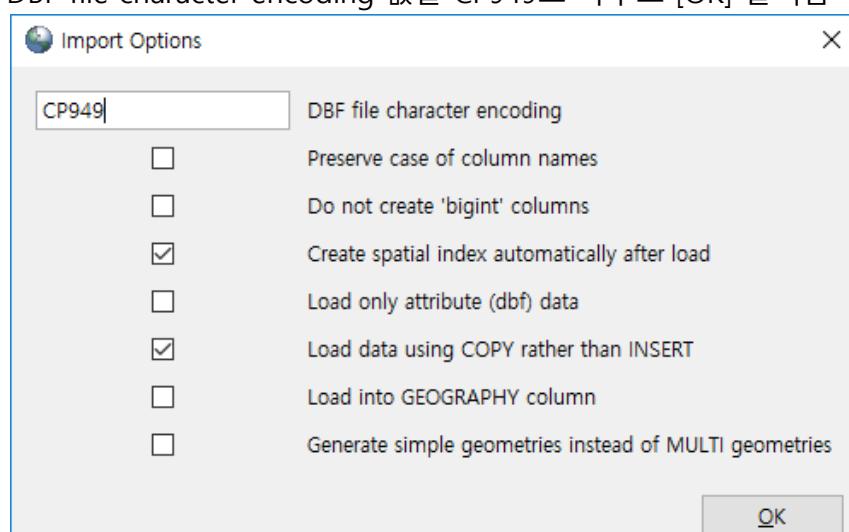
먼저 DB 접속이 잘 되는지 확인하기 위해 [View connection details...] 버튼을 누릅니다.  
접속정보가 맞는지 확인하고 [OK]를 누릅니다.



import 탭이 선택되어 있는 상태에서 [Add file] 버튼을 눌러줍니다.  
C:\data 폴더에 있는 Shape 들을 다 선택하고 [Open]을 누릅니다.

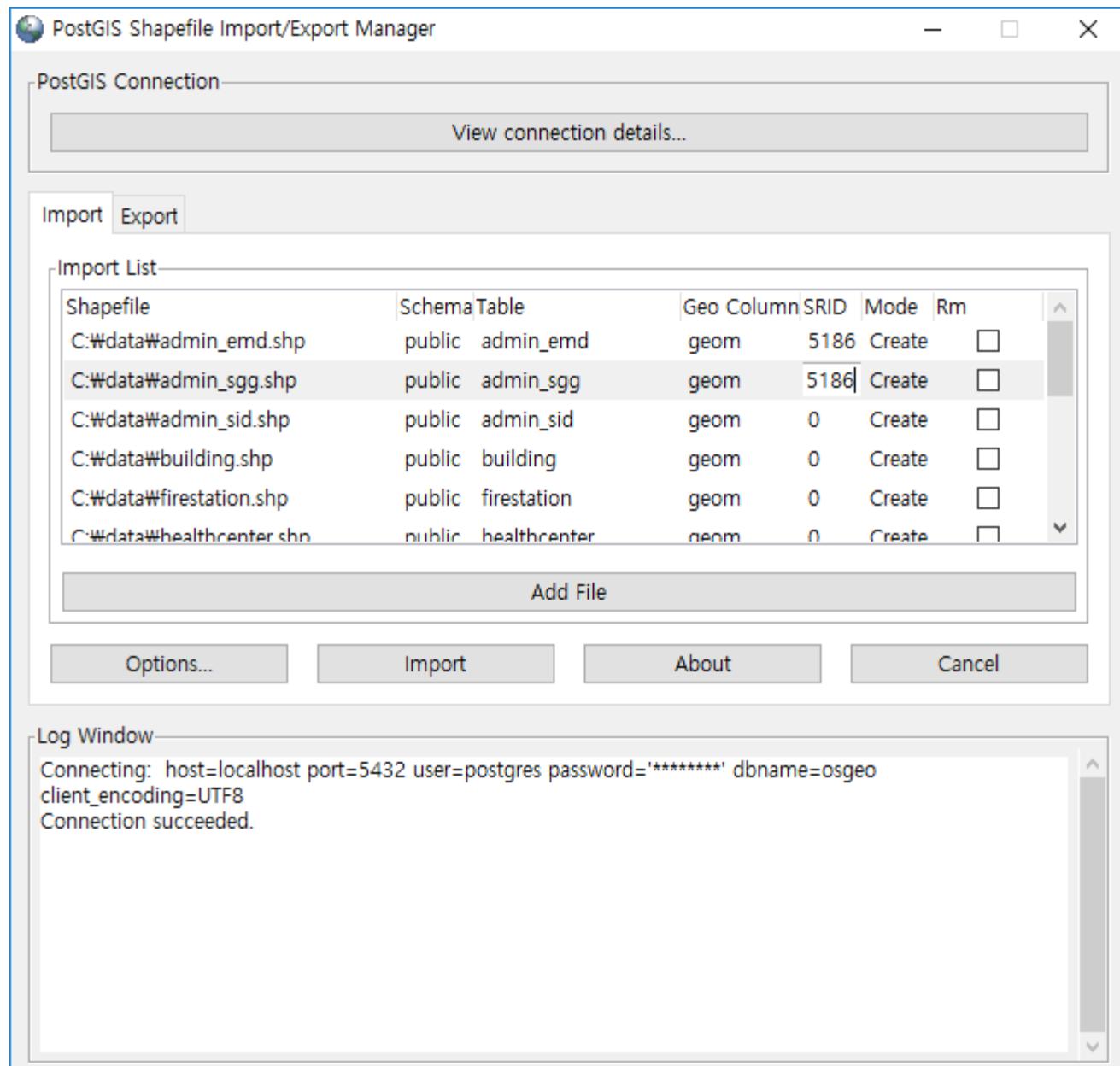
한국 자료를 다루기 위해서는 이제부터가 중요합니다.  
한글이 들어있는 자료를 오류없이 읽기 위해서는 코드페이지를 자료에 맞게 지정해 주어야만 합니다. 만일 잘못 지정하면 DB에 올리는 중에 오류가 발생하게 됩니다.

[Options...] 버튼을 눌러 상세한 정보를 지정하면 코드페이지를 지정할 수 있습니다.  
DBF file character encoding 값을 CP949로 바꾸고 [OK] 놀려줍니다.



이제 모든 자료의 SRID를 모두 5186으로 바꿔줍니다. SRID가 좌표계의 ID를 이야기합니다. 때로는 CRS라고 불리기도 합니다.

조금 귀찮기는 하지만, 하나씩 바꿔줘야 합니다. 하지만, 이렇게 각 레이어별로 좌표계를 지정할 수 있으니 여러가지 좌표계를 가진 자료들도 한꺼번에 PostGIS에 올릴 수 있다는 장점도 있습니다.



총 14개의 Shape 파일에 대해 좌표계를 다 바꿔주었습니다.

다 설정이 끝났으면 [Import] 버튼을 눌러 DB로 import를 시작합니다.  
하나씩 파일을 읽어 DB에 넣고, 공간인덱스까지 알아서 잘 만들어줍니다.

다시 pgAdmin으로 가서 Object Browser의 public Schema 아래의 Tables를 보니 14개의 레이어가 잘 만들어져 있습니다. 혹시 안보이시는 분은 Object Browser에서 [F5]를 눌러 리프래시 해 주세요.

그런데, 뭔가 좀 찜찜하네요. 원래 spatial\_ref\_sys 테이블이 있었고, 여기에 14개 레이어를 올렸으니 15개가 되어야 하는데...

올라가지 못하고 미아가 된 레이어가 있을 듯 합니다.

Log Window의 로그를 자세히 보면 road\_link\_geographic 레이어의 로그에서 오류를 찾을 수 있습니다.  
아까 QGIS에서 이 자료만 UTF-8 코드페이지를 가지고 있음을 확인했었는데 Import 시에 주의를 덜 했네요.

Log Window

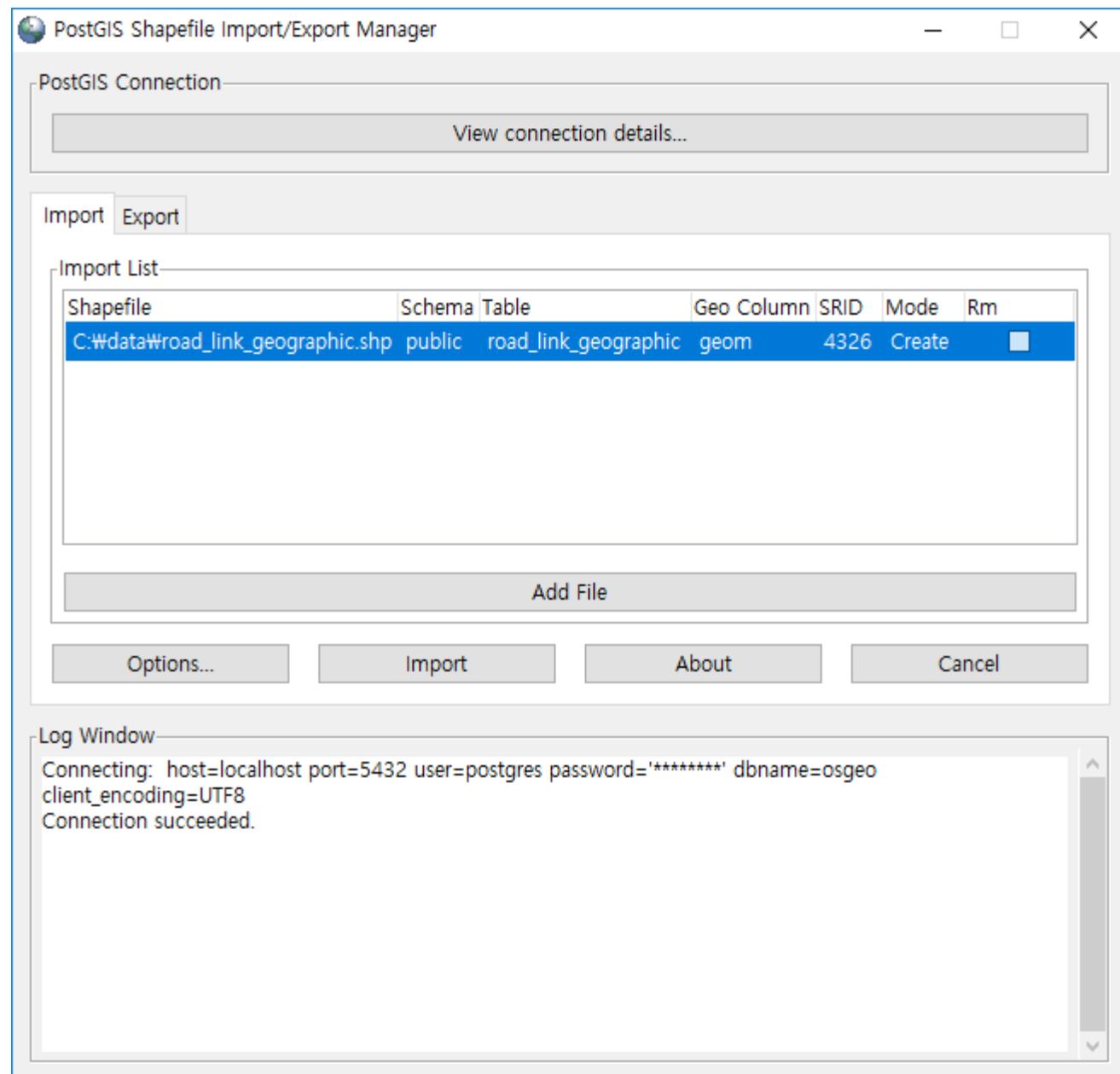
```
=====
Importing with configuration: road_link_geographic, public, geom, C:\data\road_link_geographic.shp,
mode=c, dump=1, simple=0, geography=0, index=1, shape=1, srid=5186
Shapefile type: Arc
PostGIS type: MULTILINESTRING[2]
Unable to convert data value to UTF-8 (iconv reports "Illegal byte sequence"). Current encoding is "CP949".
Try "LATIN1" (Western European), or one of the values described at http://www.postgresql.org/docs/current/
static/multibyte.html.

Shapefile import failed.
```

Unable to convert data to UTF-8~ 이런 류의 오류가 나오면 보통 코드페이지가 잘못된 것입니다. 단지 한글로 된 자료에서만 나오는 것은 아니고, 일본이나 중국, 러시아, 독일 등의 비 영어권 자료에서 흔히 발생하는 문제입니다.

이런 경우 다시 오류가 난 레이어만 옮겨주면 됩니다. 선택된 파일의 리스트를 바꾸기 위해서는 Shapefile and DBF loader를 닫았다가 다시 실행해 줘야 합니다. 좋은 프로그램인데 이 부분은 좀 아쉽네요.

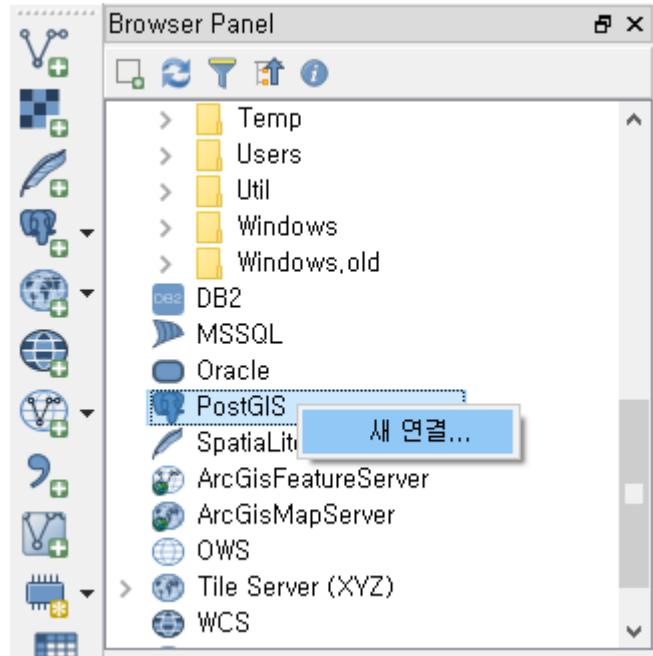
다시 실행하시고, road\_link\_geographic 레이어만 선택 후 좌표계만 바꿔주고 [Import] 하시면 됩니다. 그리고 보니 이 자료의 좌표계는 WGS84 경위도인 EPSG:4326인데 아까 5186으로 설정하고 옮겨주었네요.



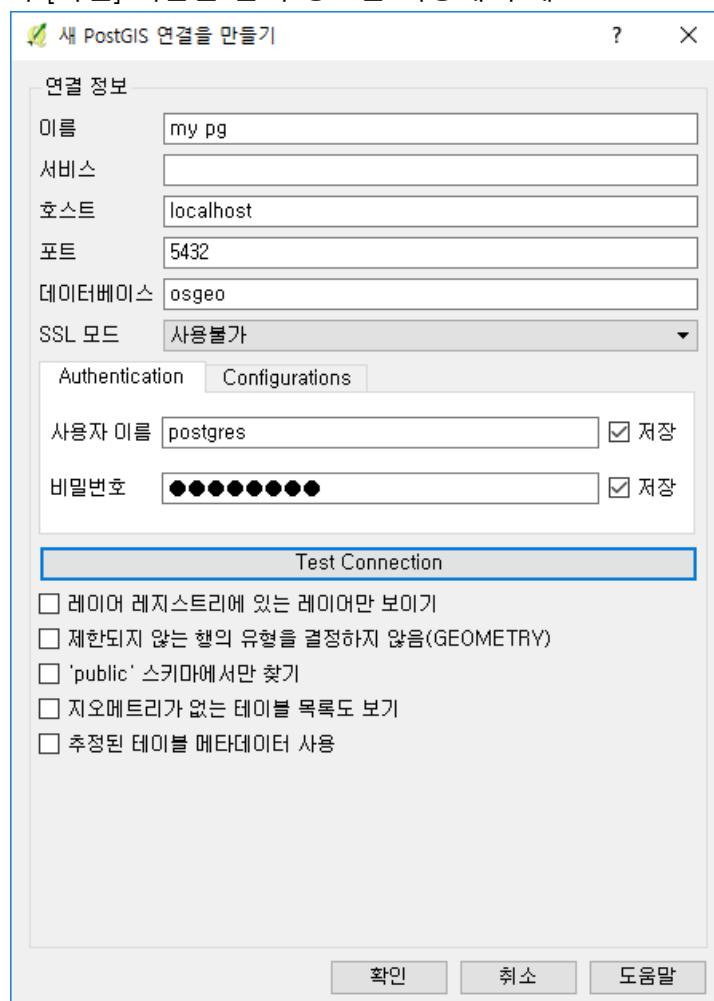
코드페이지는 기본 값이 UTF-8이기에 구지 바꾸지 않아도 됩니다.

pgAdmin에서 다시 레이어를 확인해 보니 잘 들어와 있습니다.  
QGIS에서도 확인해 보겠습니다.

QGIS의 Browser Panel의 아래 부분을 보면 PostGIS라는 코끼리 모양 아이콘의 항목이 있습니다.  
오른쪽 마우스로 클릭해 [새 연결] 메뉴를 누릅니다.

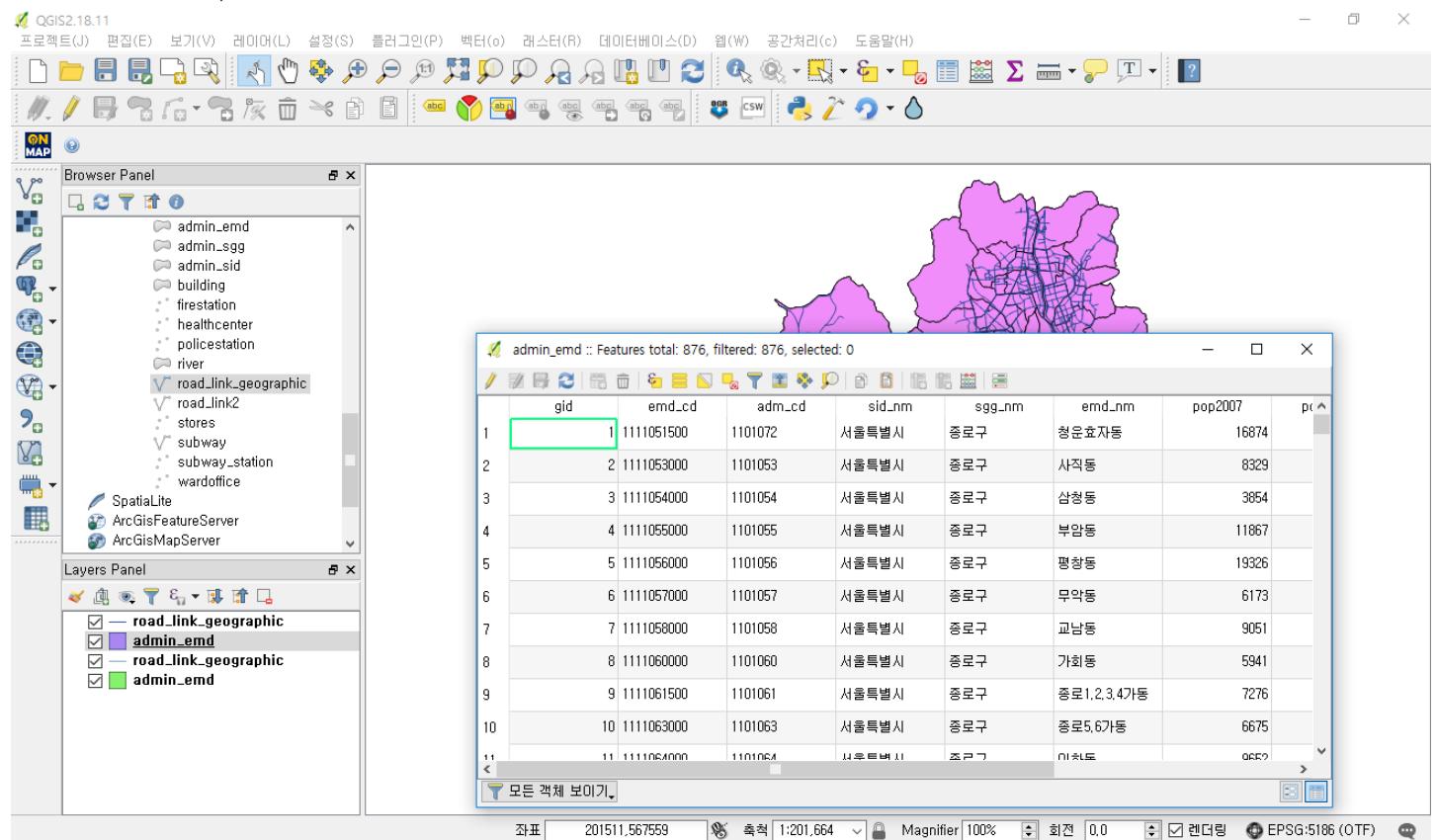


이제까지 공부하셨으니 PostGIS 연결을 위한 정보를 스스로 잘 넣어주실 수 있겠지요?  
다 입력하시고 [Test Connection] 버튼을 눌러 OK 메시지를 받으시면 성공입니다.  
꼭 [확인] 버튼을 눌러 정보를 저장해 주세요.



이제 QGIS의 Browser Panel에서 PostGIS의 my pg 연결을 펼쳐보면 public 스키마 아래에 레이어들이 보이네요.

확인을 위해 admin\_emd와 road\_link\_geographic 레이어를 더블클릭해 불러와서 Shape 파일과 동일한 위치에 보이는지, 한글이 잘 나오는지 확인해 주세요.



지금까지 가장 편리한 툴인 Shapefile and DBF loader로 공간정보를 올려보셨는데, 경험하셨다시피 완전 자동으로 모든 것이 처리되는 것은 아니고 생각보다 실수하는 경우도 많습니다.

PostGIS에 공간정보를 올리는 툴은 이 밖에도 shp2pgsql 명령이나 ogr2ogr 명령 등 커맨드 라인 명령들도 많이 사용됩니다.

사실 Shapefile and DBF loader 툴은 shp2pgsql 명령에 UI를 씌워놓은 툴입니다.

실무환경에서는 shp2pgsql 명령을 이용해 여러 파일을 올릴 수 있는 스크립트를 \*.bat나 \*.sh 파일로 만들어 사용하는 것이 더 일반적입니다.

## 4x03 공간 SQL 맛보기

DB에 공간데이터를 올렸으니 공간 SQL이 무엇인지 배워보겠습니다.

이전시간에 실습해본 공간데이터를 저장할 수 있는 환경을 만들고, 공간자료를 DB에 올리는 작업이 실은 내부적으로는 공간 SQL을 통해 이루어 졌었습니다.

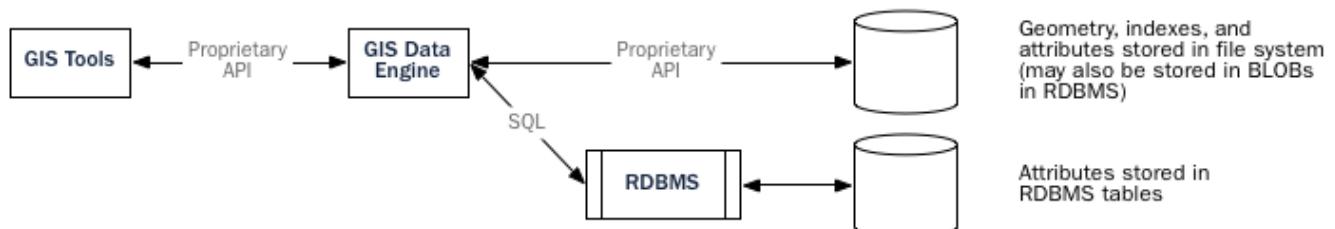
공간 SQL로 자료를 올렸으니, 불러올 수도 있겠지요? 불러 올 때 공간정보를 조건으로 줄 수도 있습니다. 또 QGIS 엔진들이 제공하는 공간자료 분석기능도 상당수 공간 SQL로 제공되고 있어 복잡한 분석도 가능합니다.

## Evolution of GIS Architectures

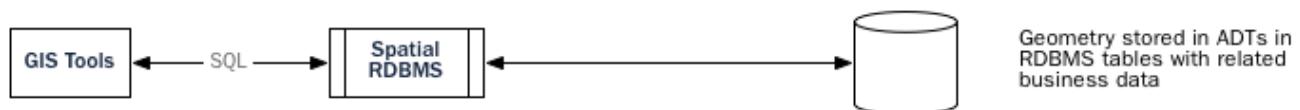
### First-Generation GIS:



### Second-Generation GIS:



### Third-Generation GIS:



출처: <http://workshops.boundlessgeo.com/postgis-intro/introduction.html>

미국의 유명한 오픈소스 GIS 기업인 Boundless 사의 자료에 따르면, GIS 엔진이 파일을 직접 읽는 형태를 1세대, 용량 등의 한계를 극복하기 위해 여기에 RDBMS가 붙어 속성자료를 처리하는 형태가 2세대, 공간 RDBMS를 이용하는 것이 3세대로 설명되고 있습니다.

간단한 작업을 예로 실제 공간 SQL이 어떻게 동작하고 어떤 장점이 있는지 확인해 보겠습니다. 이전시간에 PostGIS에 올려둔 자료를 가지고, 서울시내 8차선 이상 도로에서 500미터 이내 거리의 지하철역을 찾는 SQL을 여러가지 형태로 만들어 보겠습니다.

전통적인 GIS 툴에서는 아래 그림과 같은 5단계로 이 작업을 수행합니다.

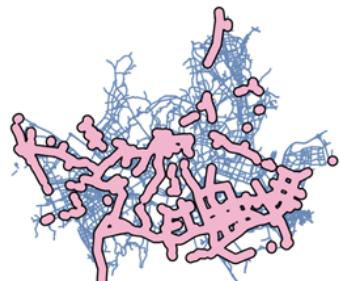
1) 도로 레이어 불러서



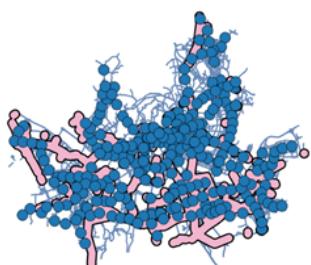
2) 8차선 이상 필터링



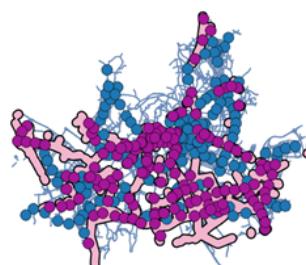
3) 선택도로 500미터 버퍼링



4) 지하철역 불러서



5) 버퍼도로 내의 지하철역 필터링



이 과정을 5가지 방식으로 공간 Python과 SQL로 만들어 보았습니다.

- 1세대 방식 : 파일기반으로 GIS 툴을 이용해 분석
- 2세대 방식 : DB에서 데이터 불러와 GIS 툴을 이용해 분석
- 3세대 방식 : 기존 분석과정 그대로 SQL로 분석
- 3세대 방식 개선 : DB에서 효율적으로 동작 가능한 SQL로 분석
- 3세대 방식 더 개선 : 더 효율적인 함수로 변경

## ● 1세대 방식 : 파일기반으로 GIS 툴을 이용해 분석

QGIS에서 이 과정을 수행할 수 있는 스크립트를 만들면 다음과 같습니다.

```
#-*- coding: utf-8 -*-
import timeit
from qgis.analysis import QgsGeometryAnalyzer

# 캔버스 초기화
QgsMapLayerRegistry.instance().removeAllMapLayers()

start = timeit.default_timer()
pre = start

# 도로 레이어 불러
roadLayer = iface.addVectorLayer("/Data/road_link2.shp", "road", "ogr")

crr = timeit.default_timer()
print(u"도로 읽기 : {}ms".format(int((crr - pre)*1000)))
pre = crr

# 8차선 이상 선택하여
expr = QgsExpression("WLANESW>=8")
it = roadLayer.getFeatures( QgsFeatureRequest( expr ) )
```

```

ids = [i.id() for i in it]
roadLayer.setSelectedFeatures( ids )

crr = timeit.default_timer()
print(u"8차선 이상 필터링 : {}ms".format(int((crr - pre)*1000)))
pre = crr

# 500 미터 버퍼 생성 후
QgsGeometryAnalyzer().buffer(roadLayer, "/Data/buffer500.shp", 500, True, True, -1)
crr = timeit.default_timer()
print(u"500 미터 버퍼 생성: {}ms".format(int((crr - pre)*1000)))
pre = crr

bufferLayer = iface.addVectorLayer("/Data/buffer500.shp", "buffer500", "ogr")

feats = [ feat for feat in bufferLayer.getFeatures() ]
geom_buffer = feats[0].geometry()

crr = timeit.default_timer()
print(u"버퍼 파일 읽기 : {}ms".format(int((crr - pre)*1000)))
pre = crr

# 지하철역 불러
stationLayer = iface.addVectorLayer("/Data/subway_station.shp", "station", "ogr")

crr = timeit.default_timer()
print(u"지하철역 읽기 : {}ms".format(int((crr - pre)*1000)))
pre = crr

# 버퍼 안에 들어가는 것 선택 후
selectedStation = [feature.id() for feature in stationLayer.getFeatures()
    if feature.geometry().within(geom_buffer) ]

crr = timeit.default_timer()
print(u"버퍼 안의 지하철역 필터링 : {}ms".format(int((crr - pre)*1000)))
pre = crr

# 결과 파일로 저장
stationLayer.setSelectedFeatures( selectedStation )
QgsVectorFileWriter.writeAsVectorFormat( stationLayer, "/Data/Result.shp", "cp949",
    stationLayer.crs(), "ESRI Shapefile", 1)
iface.addVectorLayer("/Data/Result.shp", "result", "ogr")

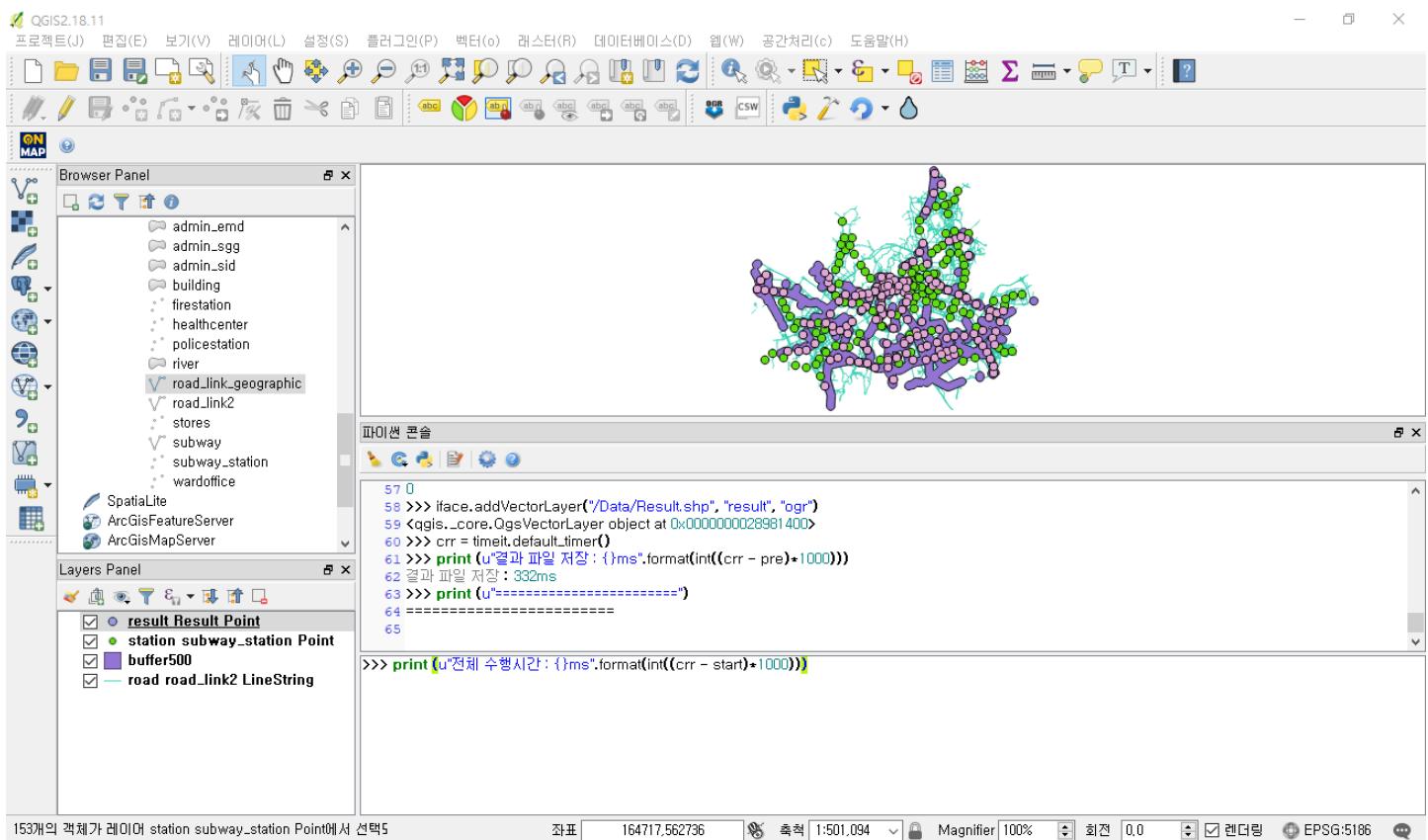
crr = timeit.default_timer()
print(u"결과 파일 저장 : {}ms".format(int((crr - pre)*1000)))

print(u"====")
print(u"전체 수행시간 : {}ms".format(int((crr - start)*1000)))

```

약 70줄의 소스인데 시간측정 부분을 빼면 40줄 정도 되겠네요.  
 QGIS에서 한번 직접 실행해 보지요.

QGIS 메뉴에서 [플러그인 - 파이썬 콘솔]을 선택하면 하단에 파이썬 콘솔이 열립니다.  
 여기에 위의 소스를 붙여 넣어 실행하시면 작업이 자동으로 진행되고 결과가 나옵니다.  
 스크립트를 파일로 저장해 두었다가 불러 쓸 수도 있습니다.



## ● 2세대 방식 : DB에서 데이터 불러와 GIS 툴을 이용해 분석

1세대 방식을 데이터 소스만 바꿔 DB에서 실행하도록 한 것입니다.  
DB 접속정보를 설정하고 이를 이용해 자료를 불러오는 방식에 집중해 보시면 좋습니다.

이 샘플데이터는 너무 크기가 작아 2세대 방식의 장점을 살릴 수는 없습니다.

```

#-*- coding: utf-8 -*-
import timeit
from qgis.analysis import QgsGeometryAnalyzer

# 캔버스 초기화
QgsMapLayerRegistry.instance().removeAllMapLayers()

# DB 접속정보
uri = QgsDataSourceURI()
uri.setConnection("localhost", "5432", "osgeo", "postgres", "postgres")

start = timeit.default_timer()
pre = start

# 도로 레이어 불러
uri.setDataSource("public", "road_link2", "geom")
roadLayer = iface.addVectorLayer(uri.uri(False), "road", "postgres")

crr = timeit.default_timer()
print(u"도로 읽기 : {}ms".format(int((crr - pre)*1000)))
pre = crr

# 8차선 이상 선택하여
expr = QgsExpression( "W" "LANESW" >= 8" )

```

```

it = roadLayer.getFeatures( QgsFeatureRequest( expr ) )
ids = [i.id() for i in it]
roadLayer.setSelectedFeatures( ids )

crr = timeit.default_timer()
print (u"8차선 이상 필터링 : {}ms".format(int((crr - pre)*1000)))
pre = crr

# 500 미터 버퍼 생성 후
QgsGeometryAnalyzer().buffer(roadLayer, "/Data/buffer500.shp", 500, True, True, -1)
crr = timeit.default_timer()
print (u"500 미터 버퍼 생성: {}ms".format(int((crr - pre)*1000)))
pre = crr

bufferLayer = iface.addVectorLayer("/Data/buffer500.shp", "buffer500", "ogr")

feats = [ feat for feat in bufferLayer.getFeatures() ]
geom_buffer = feats[0].geometry()

crr = timeit.default_timer()
print (u"버퍼 파일 읽기 : {}ms".format(int((crr - pre)*1000)))
pre = crr

# 지하철역 불러
uri.setDataSource("public", "subway_station", "geom")
stationLayer = iface.addVectorLayer(uri.uri(False), "road", "postgres")

crr = timeit.default_timer()
print (u"지하철역 읽기 : {}ms".format(int((crr - pre)*1000)))
pre = crr

# 버퍼 안에 들어가는 것 선택 후
selectedStation = [feature.id() for feature in stationLayer.getFeatures()
    if feature.geometry().within(geom_buffer) ]

crr = timeit.default_timer()
print (u"버퍼 안의 지하철역 필터링 : {}ms".format(int((crr - pre)*1000)))
pre = crr

# 결과 파일로 저장
stationLayer.setSelectedFeatures( selectedStation )
QgsVectorFileWriter.writeAsVectorFormat( stationLayer, "/Data/Result.shp", "cp949",
    stationLayer.crs(), "ESRI Shapefile", 1)
iface.addVectorLayer("/Data/Result.shp", "result", "ogr")

crr = timeit.default_timer()
print (u"결과 파일 저장 : {}ms".format(int((crr - pre)*1000)))

print (u"=====")
print (u"전체 수행시간 : {}ms".format(int((crr - start)*1000)))

```

## ● 3세대 방식 : 기존 분석과정 그대로 SQL로 분석

이제 앞에서 해본 전통적인 GIS 작업을 그대로 SQL로 바꾼 것입니다.  
얼마나 짧아졌는지에 집중해 봐 주세요.

```

select st.*
from subway_station as st,
(
  select st_buffer(st_union(geom), 500) as geom
  from road_link2 where lanes >=8
) as buf
where st_within(st.geom, buf.geom)

```

도로중심선 레이어에서 8차선 이상만 필터링 해, st\_buffer 함수로 버퍼링하고, 그 결과를 지하철역 레이어와 st\_within 함수를 조건으로 JOIN 하는 것으로 끝입니다.

### ● 3세대 방식 개선 : DB에서 효율적으로 동작 가능한 SQL로 분석

앞의 공간 SQL을 좀더 RDBMS의 장점을 살리도록 바꿀 수도 있습니다.

```

select * from subway_station
where id in
(
  select distinct st.id
  from subway_station as st, road_link2 as road
  where road.lanes >= 8
    and ST_Distance(st.geom, road.geom) <= 500
)

```

핵심은 시간이 많이 소요되는 버퍼링 과정 대신 st\_distance 함수를 써서 거리 계산으로 바꿔버렸다는 것입니다. 이렇게 DBMS에서는 공간연산 보다는 숫자계산이 월씬 빠릅니다.

주의할 것은 이렇게 JOIN을 이용하는 경우 동일한 데이터가 여러번 나오는 경우가 있습니다. 이를 피하기 위해 위 소스의 앞 2줄처럼 필요한 공간객체의 id 값만을 서브쿼리 내에서 조회한 것을 다시 원하는 레이어의 where 절 조건으로 주는 것이 좋습니다.

### ● 3세대 방식 더 개선 : 더 효율적인 함수로 변경

조금 더 고민하면 이를 더 빨리 할 수도 있습니다.

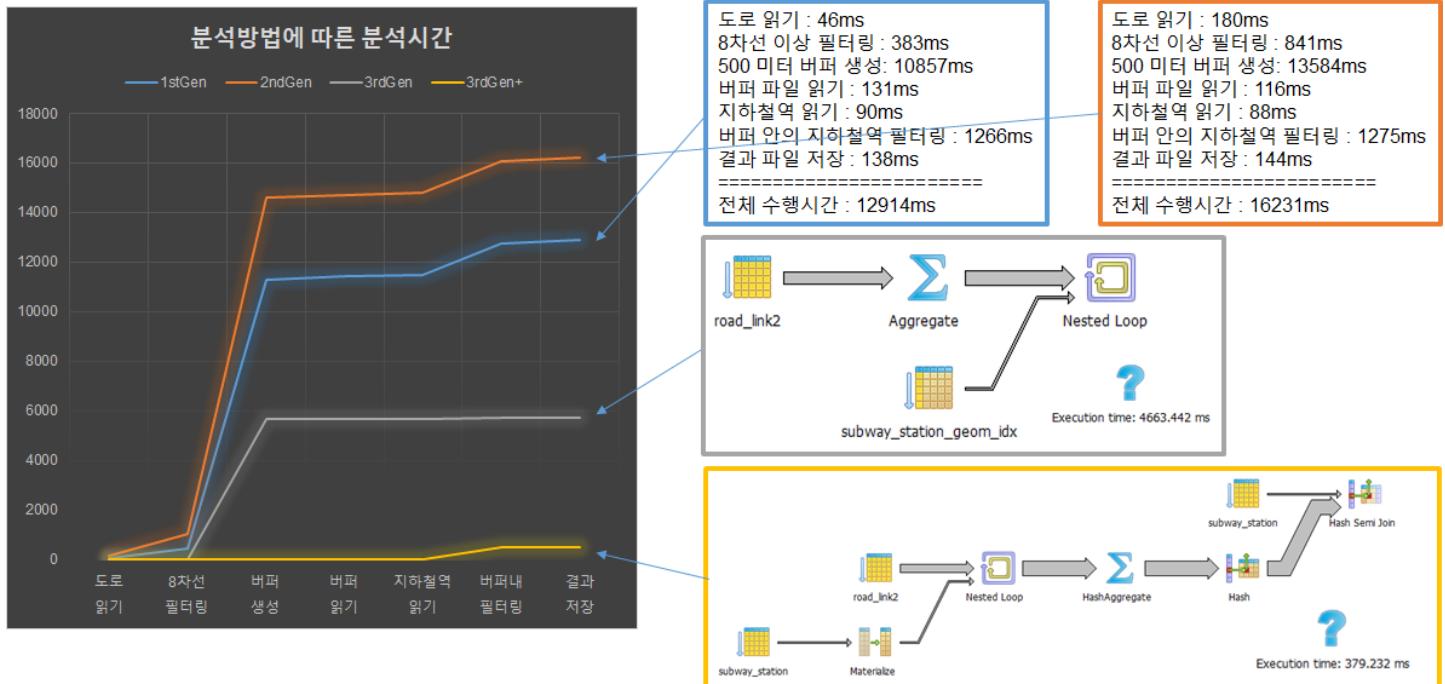
```

select * from subway_station
where id in
(
  select distinct st.id
  from subway_station as st, road_link2 as road
  where road.lanes >= 8
    and ST_DWithin(st.geom, road.geom, 500)
)

```

비슷해 보이지만 st\_distance 대신 st\_dwithin을 사용해 거리기준 필터링을 했다는 점에서 차이가 있습니다. st\_distance를 사용한 방식에서는 모든 지하철역과 모든 도로간 거리를 다 계산해 500미터 이내만 필터링 했습니다.

하지만, st\_dwithin을 사용한 방식은 내부적으로 먼저 최소영역사각형(MBR, Minimum Bounded Rectangle)으로 판단해 500미터 이내에 들어올 가능성이 없는 것은 아예 거리계산도 안하고 걸러내고 가는성이 있는 것들만 거리계산을 한다는 차이가 있습니다.



앞에서 살펴본 방식들의 성능을 비교해 본 그림입니다.

노란색의 4번째 방식이 어마어마하게 빠르지요? 그림에는 없는 5번째 방식은 4번째 방식보다 10배 정도 빠릅니다.

## 4x04 공간 SQL 실습

먼저 간단한 일반 SQL 부터 시작해 보겠습니다.  
pgAdmin으로 가서 새 Query 창을 띄우고 실습하시면 됩니다.

```
SELECT *
FROM stores;
```

매장들이 들어있는 store 테이블의 내용을 모두 보는 쿼리입니다.  
공간데이터 컬럼이 어찌 보이는지는 한번 더 확인해 보세요.  
혹시 빈칸처럼 보여도 실제로는 비어있지는 않습니다.

```
SELECT *
FROM stores
WHERE brand='이마트';
```

매장 중 이마트만 필터링해 보는 쿼리입니다.

```
SELECT COUNT(nam)
FROM stores
WHERE addr like '%영등포구%';
```

영등포구의 매장만도 간단히 필터링 할 수 있습니다.

```
SELECT brand, COUNT(nam) as "점포수"
FROM stores
GROUP BY brand;
```

각 브랜드별 점포수를 조회하기 위해 집계 함수인 COUNT를 사용했습니다.  
집계함수를 사용할 때는 보통 GROUP BY 문이 필요합니다.

```
SELECT brand, AVG(char_length(nam)), STDDEV(char_length(nam))
FROM stores GROUP BY brand;
```

평균과 분산을 구하는 정도는 DB에게는 일도 아니지요.

```
SELECT ST_AsText(geom)
FROM firestation;
```

드디어 공간 SQL입니다. 지오메트리를 텍스트로 만들어 사람이 알아볼 수 있는 형태로 조회했네요.  
ST\_로 시작하는 함수가 공간 SQL을 위한 함수입니다.

```
SELECT link_id, ST_Length (geom)  
FROM road_link_geographic;
```

길이를 구하는 것 쯤이야 쉽지요.

```
SELECT link_id, ST_Length(ST_Transform(geom,5179))  
FROM road_link_geographic;
```

하지만, 경위도 등 미터단위가 아닌 좌표계에서 거리나 면적을 계산하면 영뚱한 값이 나옵니다.  
이럴 때는 미터단위를 사용하는 좌표계로 변환하여 계산하면 됩니다.

```
SELECT ufid, ST_Area(ST_Transform(geom,5179))  
FROM building  
LIMIT 100;
```

면적도 어려울 것이 없지요.

마지막 줄의 LIMIT 100은 결과중 100개만 보여주는 것인데, 자료량이 많을 때 테스트시 특히 유용합니다.  
웹에서의 페이지 등에 조회기법을 위해서도 사용합니다.

```
SELECT ufid, ST_Area(geom)  
FROM building  
LIMIT 100;
```

하지만, 앞에서의 SQL에는 불필요한 좌표계 변환이 들어있습니다.

원 자료의 5186 좌표계도 미터단위 TM 직각좌표계고, 변환한 5179 좌표계도 미터단위 TM 직각좌표계여서  
둘 좌표계가 타원체가 다르기는 하지만 면적 계산시에는 전혀 좌표계 변환할 필요가 없습니다.

```
ALTER TABLE stroes  
ADD Column buffer geometry(Polygon,4326);
```

기존 테이블에 공간데이터를 저장할 수 있는 컬럼을 추가하는 것도 쉽습니다.  
이제 stroes 테이블은 한 행당 2가지 공간자료가 들어갈 수 있게 되었습니다.

```
UPDATE stores  
SET buffer=ST_Transform  
(ST_Buffer(ST_Transform(geom,5186), 30), 4326);
```

이렇게 하면 방금 만든 buffer라는 컬럼에 실제로 30미터 버퍼를 만든 것을 경위도로 변환해 저장하게 됩니다.  
만약 경위도로 들어오는 사람이나 자동차의 실시간 위치를 파악해 점포주변 30미터에 들어오면 쿠폰을  
발송하는 등의 시스템을 만들려면 이렇게 자료를 다음어 두면 좋겠지요.

```
SELECT ST_AsGeoJSON(ST_Transform(geom, 4326))  
FROM stores;  
SELECT ST_AsGML(geom) FROM stores;
```

인터넷 상에서 자료를 교환할 때 보통 JSON이나 XML을 사용합니다.  
공간정보를 위한 JSON이 GeoJSON이라는 표준이고, XML의 경우는 GML이란 표준입니다.

주의할 것은 위의 ST\_AsGeoJSON, ST\_AsGML 함수는 도형부분만을 반환하고 속성은 넣어주지 않는다는 것입니다. 실제로 교환시에는 아래 예처럼 속성까지 포함된 완전한 형태로 전달하도록 코딩해 주어야 합니다.

```
{  
  "type": "Feature",  
  "geometry": {  
    "type": "Point",  
    "coordinates": [127.09, 37.59]  
  },  
  "properties": {  
    "name": "Dinagat Islands"  
  }  
}
```

```
<Feature>  
  <name>Dinagat Islands</name>  
  <position>  
    <gml:Point srsName="EPSG:5186">  
      <gml:coordinates>  
        207965.03669, 555284.45453  
      </gml:coordinates>  
    </gml:Point>  
  </position>  
</Feature>
```

```
SELECT ST_NPoints(geom) AS num_point  
FROM road_link2  
ORDER BY num_point DESC  
LIMIT 100;
```

도형이 몇 개의 점으로 이루어졌는지 등 도형에 관한 상세한 정보도 조회할 수 있습니다.

```
SELECT shop.nam AS "매장명", metro.nam AS "인근역"  
FROM stores AS shop, subway_station AS metro  
WHERE ST_Intersects(ST_Buffer(shop.geom, 500), metro.geom);
```

앞에서도 해 봤듯이 공간연산을 조건절에 추가해 공간을 기준으로 JOIN 할 수도 있습니다.

```
SELECT shop.nam AS "매장명", metro.nam AS "인근역"  
FROM stores AS shop, subway_station AS metro  
WHERE shop.addr LIKE '%영등포%'  
AND ST_Intersects(ST_Buffer(shop.geom, 500), metro.geom);
```

공간연산 뿐 아니라 일반 속성에 대한 연산까지 추가해 더욱 효과적인 자료조회가 가능합니다.

본 강의에서 배우지 않는 강력한 공간 SQL 들이 많이 있습니다.  
다음 링크의 PostGIS Reference에서 공간정보 함수들의 정보를 얻을 수 있습니다.  
<http://postgis.net/docs/manual-2.4/reference.html>

다음 OSGeo 한국어지부 문서저장소에서 찾아보시면 한글로 된 자료도 있습니다.  
<http://tinyurl.com/osgeo-kr-docs>

## 4x05 공간자료를 효과적으로 다루는 커맨드라인 명령어

공간자료 자체를 다루는 강의의 마지막 부분으로 커맨드라인 명령어(CLI, Command Line Interface)를 이용해 공간자료를 다루는 방법을 배워보겠습니다.

벡터 데이터를 다룰 때는 ogr 명령들이 주로 사용됩니다.

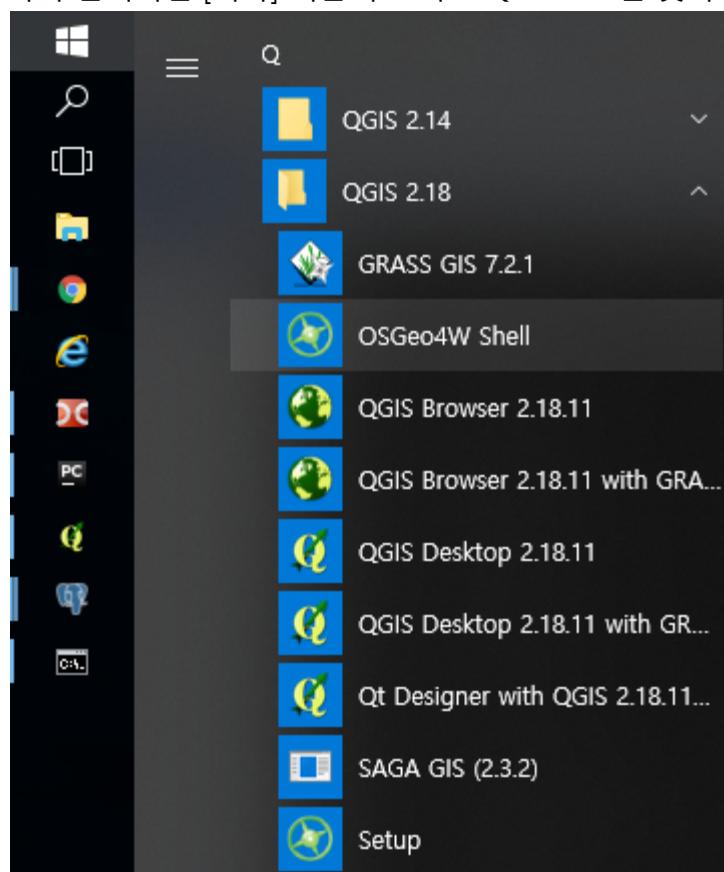
[http://www.gdal.org/ogr\\_utilities.html](http://www.gdal.org/ogr_utilities.html)

이 링크에 가면 여러가지 명령이 있는데 본 강의에서는 ogr2ogr만을 배웁니다.

윈도우 환경에서 공간정보 관련 명령어들을 쓰시려면 QGIS와 함께 설치된 OSGeo4W Shell을 이용하시는 것이 편합니다.

화면이나 키보드의 원도우 버튼을 누르고 'osgeo4w'를 검색하시면 쉽게 찾으실 수 있을 것입니다.

혹시 안되시면 [시작] 버튼 누르시고 QGIS 2.18을 찾아 그 안에 보시면 OSGeo4W Shell이 있습니다.



아이콘을 눌러 시작하면 검은 도스창이 나옵니다.

여기에 ogr2ogr 명령을 입력해서 옵션 설명이 보이면 정상 동작하는 것입니다.

```

OSGeo4W Shell
C:\#>ogr2ogr
Usage: ogr2ogr [--help-general] [-skipfailures] [-append] [-update]
                [-select field_list] [-where restricted_where[@filename]}
                [-progress] [-sql <sql_statement>[@filename]] [-dialect dialect]
                [-preserve_fid] [-fid FID] [-limit nb_features]
                [-spat xmin ymin xmax ymax] [-spat_srs srs_def] [-geomfield field]
                [-a_srs srs_def] [-t_srs srs_def] [-s_srs srs_def]
                [-f format_name] [-overwrite] [[-dsco NAME=VALUE] ...]
                dst_datasource_name src_datasource_name
                [-lco NAME=VALUE] [-nln name]
                [-nlt type{PROMOTE_TO_MULTI|CONVERT_TO_LINEAR|CONVERT_TO_CURVE}]
                [-dim XY|XYZ|XYM|XYZM|layer_dim] [layer [layer ...]]

Advanced options :
    [-gt n] [-ds_transaction]
    [[-oo NAME=VALUE] ...] [[-doo NAME=VALUE] ...]
    [-clipsrc [xmin ymin xmax ymax]|WKTDatasource|spat_extent]
    [-clipsrcsql sql_statement] [-clipsrclayer layer]
    [-clipsrcwhere expression]
    [-clipdst [xmin ymin xmax ymax]|WKTDatasource]
    [-clipdstsql sql_statement] [-clipdstlayer layer]
    [-clipdstwhere expression]
    [-wrapdateeline] [-dateelineoffset val]
    [[-simplify tolerance] | [-segmentize max_dist]]
    [-addfields] [-unsetFid]
    [-relaxedFieldNameMatch] [-forceNullable] [-unsetDefault]
    [-fieldTypeToString All|{type1[,type2]*}] [-unsetFieldWidth]
    [-mapFieldType srctype|All=dstype[,srctype2=dstype2]*]
    [-fieldmap identity | index1[,index2]*]
    [-splitlistfields] [-maxsubfields val]
    [-explodecollections] [-zffield field_name]
    [-gcp pixel_line easting northing [elevation]]* [-order n | -tps]
    [-nomd] [-mo "META-TAG=VALUE"]* [-noNativeData]

Note: ogr2ogr --long-usage for full help.

FAILURE: no target datasource provided
C:\#>

```

ogr2ogr로 가장 많이 하는 작업은 공간자료를 다룰 때 가장 힘든 작업인 벡터자료의 좌표계를 바꿔주는 작업입니다.

실습 자료가 들어 있는 C:\data 폴더로 이동해서 firestation 레이어를 EPSG:5174 좌표계로 변환해 보겠습니다.

```

cd C:\data

ogr2ogr -s_srs EPSG:5186 -t_srs "+proj=tmerc +lat_0=38 +lon_0=127.0028902777778
+k=1 +x_0=200000 +y_0=500000 +ellps=bessel +units=m +no_defs
+towgs84=-115.80,474.99,674.11,1.16,-2.31,-1.63,6.43" -f "ESRI Shapefile"
--config SHAPE_ENCODING "CP949" firestation_5174.shp firestation.shp

```

ogr2ogr의 인자를 하나씩 살펴보겠습니다.

-s\_srs 인자는 원본(source) 자료의 좌표계를 의미합니다. EPSG:5186 좌표계로 지정했네요.

-t\_srs 인자는 대상(target) 자료의 좌표계를 의미합니다. EPSG:5174 좌표계로 바꾼다고 했는데, 뭔가 많은 내용이 들어있네요. 이것은 ogr2ogr도 Bessel 타원체의 변환정보가 잘못되어 있기 때문입니다. 이런 경우에는 간단히 적어줄 수 없고, proj4용 좌표계 인자들을 모두 써줘야 합니다.

-f 인자는 대상자료를 어떤 포맷(format)으로 만들지를 의미합니다.

--config SHAPE\_ENCODING 인자는 ESRI Shape을 다룰 때만 들어가는 추가인자로 한글 등의 코드페이지를 의미합니다.

그리고 주의할 것이 마지막 부분의 인자 순서가 다른 명령과 달리 조금 특이합니다.

대상파일 명(firestation\_5174.shp)이 먼저 나오고 원본파일 명(firestation.shp)이 뒤에 나옵니다.

스크립트의 인자가 많아 생각보다 복잡했지요?

이런 좌표계 변환 작업은 QGIS에서도 쉽게 할 수 있긴 합니다. 심지어 UI가 있어 더 편합니다.

하지만, 좌표계 변환해야 하는 파일이 100개라면? 혹은 1000개라면?

많은 수의 파일을 작업해야 할 때, 혹은 아주 상세한 설정이 필요할 때 커맨드라인 명령어를 사용하면 대단히 효과적이고, 스크립트로 저장해서 실행하면 어떤 작업을 했는지 나중에도 명확히 관리할 수 있는 장점이 있습니다.

이번에는 공간자료의 파일 포맷 변환에 ogr2ogr을 사용해 보겠습니다.

```
ogr2ogr -f "GPKG" output.gpkg PG:"host=localhost dbname=osgeo user=postgres password=postgres schemas=public tables=admin_emd,admin_sgg,admin_sid,building,firestation,healthcenter,policestation,river,road_link2,road_link_geographic,stores,subway,subway_station,wardoffice"
```

-f 인자를 "GPKG"로 해서 만들어지는 파일 포맷을 GeoPackage로 지정했네요.

다음 링크에 가시면 어떤 포맷을 어떤 예약어로 사용할 수 있는지 확인하실 수 있습니다.

[http://www.gdal.org/ogr\\_formats.html](http://www.gdal.org/ogr_formats.html)

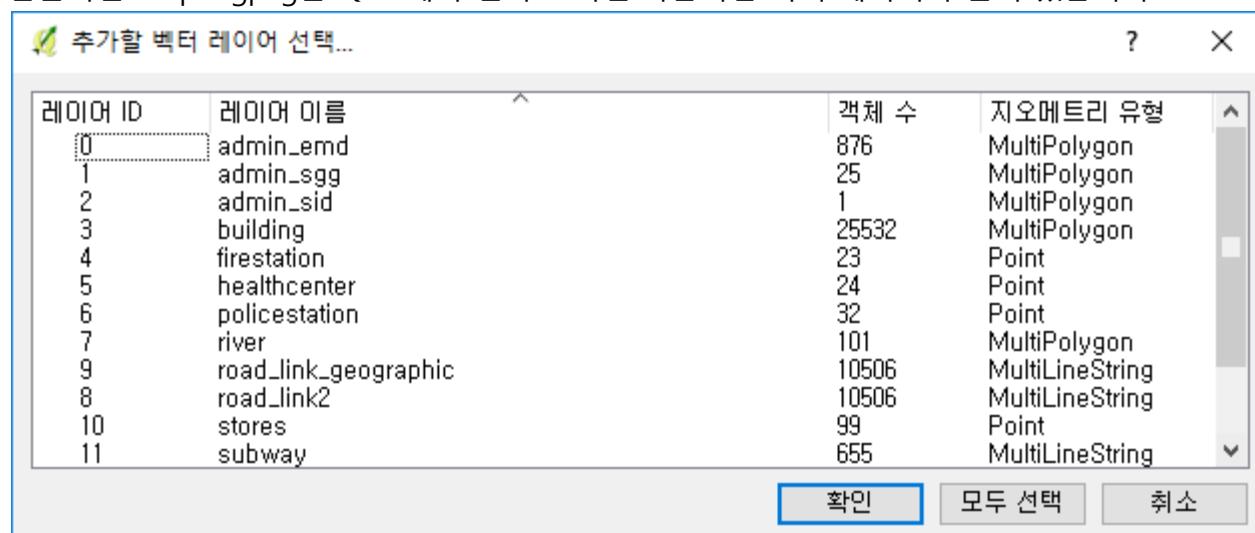
그리고 좌표계 등 다른 인자 없이 바로 대상파일과 소스 파일이 나오네요.

대상 파일은 output.gpkg 파일이군요. 이건 쉽습니다.

원본 파일은 파일이 아니네요! PostGIS 접속정보를 써 놨습니다.

특히 tables 항목에 여러 레이어를 지정해서 한꺼번에 받고 있네요.

만들어진 output.gpkg를 QGIS에서 열어 보시면 다음처럼 여러 레이어가 들어 있습니다.



이렇게 ogr2ogr을 다양한 공간정보간의 포맷변환에 사용하실 수 있습니다. 심지어 Oracle Spatial, ArcSDE 등의 독점 소프트웨어 DBMS의 자료도 잘 다룹니다.

심지어 DBMS가 아닌 파일에도 SQL로 원하는 자료만 뽑아 낼 수 있습니다.

```
ogr2ogr -sql "select * from road_link2 where lanes >= 8" --config SHAPE_ENCODING "CP949" lane8.shp road_link2.shp
```

-sql 옵션으로 SQL을 실행하고 있네요. 이 SQL은 PostGIS의 SQL과는 약간 다르고 공간 SQL은 안됩니다.

[http://www.gdal.org/ogr\\_sql.html](http://www.gdal.org/ogr_sql.html)

--config SHAPE\_ENCODING 옵션은 CP949 인코딩의 한글이 들어있는 Shape 파일을 다룰 때는 꼭 필요합니다. 없으면 오류가 납니다.

대상파일과 원본파일은 형식을 지정하지 않았는데도 문제 없네요.

파일의 확장자나 파일 내용을 보고 OGR이 알아서 판단합니다. 하지만, 알아서 판단하는 것이 틀리는 경우가 있기에 가능하다면 명확히 지정해 주는 것이 좋습니다.

경고들이 주욱 나오기는 하는데요, 이는 자리수의 문제 때문이고 오류는 아닙니다.

이제 래스터 자료를 다뤄 보겠습니다.

래스터 자료용 명령어는 GDAL이 담당하고 있습니다.

[http://www.gdal.org/gdal\\_utilities.html](http://www.gdal.org/gdal_utilities.html)

GDAL 명령어는 목적에 따라 여러가지를 사용합니다.

먼저 래스터 데이터의 정보를 조회해 보겠습니다.

```
gdalinfo BlueMarbleNG-TB_2004-12-01_rgb_3600x1800.TIFF
```

여러가지 정보를 확인할 수 있는데, 영상의 크기가 3600, 1800로 나옵니다.

Coordinate System 부분에서 좌표계도 알 수 있습니다. EPSG:4326이니 경위도군요.

Pixel Size = (0.1000000000000000,-0.1000000000000000)를 보니 한 픽셀이 0.1 단위고 경위도니 0.1도 해상도 자료입니다. Y 방향에는 보통 마이너스 기호가 붙어 있습니다.

공간적 범위는 Corner Coordinates 부분을 보면 됩니다. 전지구 범위의 영상이네요.

이제 포맷 변환을 해보겠습니다.

```
gdal_translate -of JPEG BlueMarbleNG-TB_2004-12-01_rgb_3600x1800.TIFF  
WorldMap.jpg
```

-of 인자가 대상파일의 포맷을 지정하고 있습니다. 사용 가능한 포맷은 다음 링크를 참고하세요.

[http://www.gdal.org/formats\\_list.html](http://www.gdal.org/formats_list.html)

원본과 대상 파일을 지정하는 순서가 ogr2ogr과는 달리 상식적으로 원본, 대상 순임도 주의하세요.

10 메가 바이트 GeoTIFF 파일을 600 킬로 바이트 JPEG 파일로 변환했습니다.

변환된 JPEG 파일과 함께 공간정보를 담고 있는 WorldMap.jpg.aux.xml 파일도 같이 생겨서 QGIS 등의 프로그램에서 여전히 공간정보로 사용할 수 있네요.

래스터 데이터의 좌표계 변환도 많이 하는 작업입니다.

```
gdalwarp -s_srs EPSG:4326 -t_srs EPSG:5179 -of GTiff -r cubic -te 123 32 132 44  
-te_srs EPSG:4326 BlueMarbleNG-TB_2004-12-01_rgb_3600x1800.TIFF Korea_5179.tif
```

-s\_srs 인자는 원본 좌표계이고 경위도좌표계네요.

-t\_srs 인자는 대상 좌표계이고 국가인터넷지도와 네이버지도에서 사용중인 GRS80타원체 UTM-K네요.

-of 인자는 대상파일 포맷임을 다 아시겠지요?

-r 인자는 대상영상의 각 픽셀값을 결정할때 원본에 있는 점들 여려개를 어찌 보간할지인데, 2차방정식을 선택했네요.

-te 인자는 대상파일을 만들 공간영역입니다. minX minY maxX maxY 순서로 적습니다.

-te\_srs 인자는 -te 인자에 쓰인 좌표계가 무엇인지 설정하는 것이네요. 경위도로 했군요.

뒤에는 역시 원본파일 대상파일 순으로 적었습니다.

래스터 데이터를 빨리 보이게 하는 대표적인 방법이 미리보기(Overlay) 영상을 피라미드 처럼 다단계로 만들어 두는 것입니다.

```
gdaladdo -r average BlueMarbleNG-TB_2004-12-01_rgb_3600x1800.TIFF 2 4 8 16 32
```

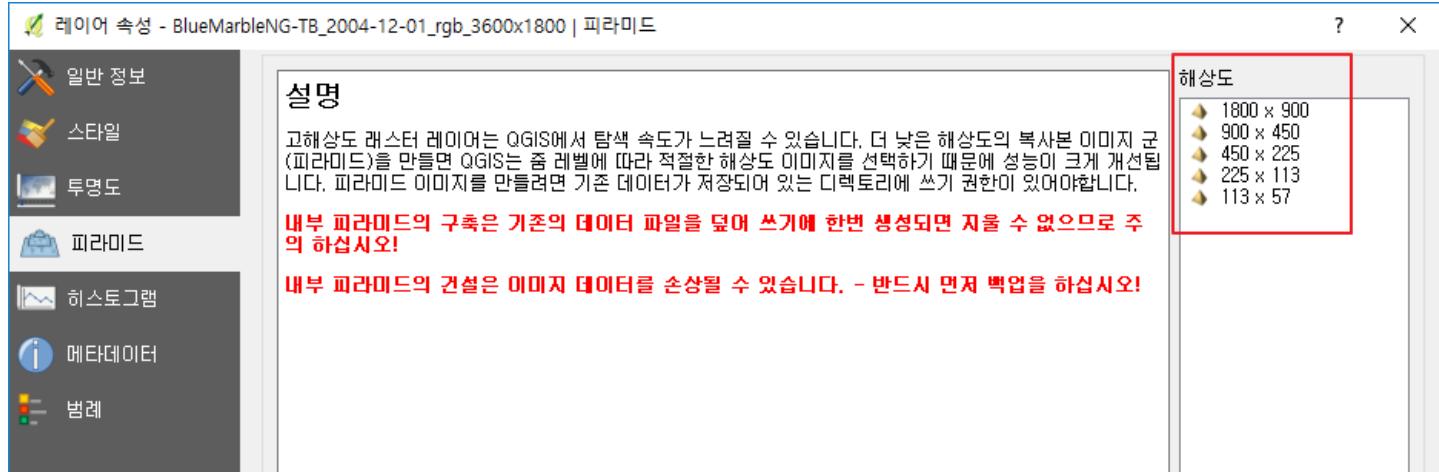
-r 인자로 보간방법을 지정했는데 이번에는 평균법으로 했습니다.

그리고 오버레이를 만들 파일이름이 나오고 몇 개씩의 픽셀을 합쳐 오버레이를 만들지 숫자들이 나오네요.

이렇게 오버레이를 만들어 두면 QGIS나 GeoServer 등에서 큰 영상을 매우 빠른 속도로 볼 수 있습니다.

오버레이가 만들어진 영상은 QGIS의 속성창에서 피라미드 탭의 해상도 부분을 보면 확인할 수 있습니다.

오버레이가 만들어지지 않은 단계가 있는 경우 이 부분이 붉은색 아이콘으로 표시됩니다.



## 5x01 GeoServer에 레이어 등록

이제 인터넷에 공간정보를 서비스하는 GeoServer에 대해 배워보겠습니다.  
우선 무작정 데이터를 등록해 봅시다.

먼저 GeoServer를 실행해 주세요. [시작] 버튼을 누르시고, GeoServer 2.11.4 안의 Start GeoServer를 선택하시면 시작됩니다. 혹시 서비스로 설치하신 분은 자동으로 시작되어 있을 것입니다.

웹브라우저를 띄우시고 다음 주소로 갑니다.

<http://localhost:8080/geoserver>

GeoServer: 시작하기

localhost:8080/geoserver/web/

GeoServer

정보 & 상태

GeoServer 정보

데이터

레이어 미리보기

데모

시작하기

활영합니다

GeoServer의 서비스는 다음의 기관에서 운영합니다: The Ancient Geographers.

GeoServer는 2.11.4 버전에서 운영 중입니다. 더 자세한 정보는 관리자에게 문의 하십시오.

Service Capabilities

- WCS
  - 1.0.0
  - 1.1.0
  - 1.1.1
  - 1.1
  - 2.0.1
- WFS
  - 1.0.0
  - 1.1.0
  - 2.0.0
- WMS
  - 1.1.1
  - 1.3.0
- TMS
  - 1.0.0
- WMS-C
  - 1.1.1
- WMPS
  - 1.0.0

로그인 해 주세요.

설치시 바꾸지 않으셨다면 username: admin, password: geoserver 입니다.

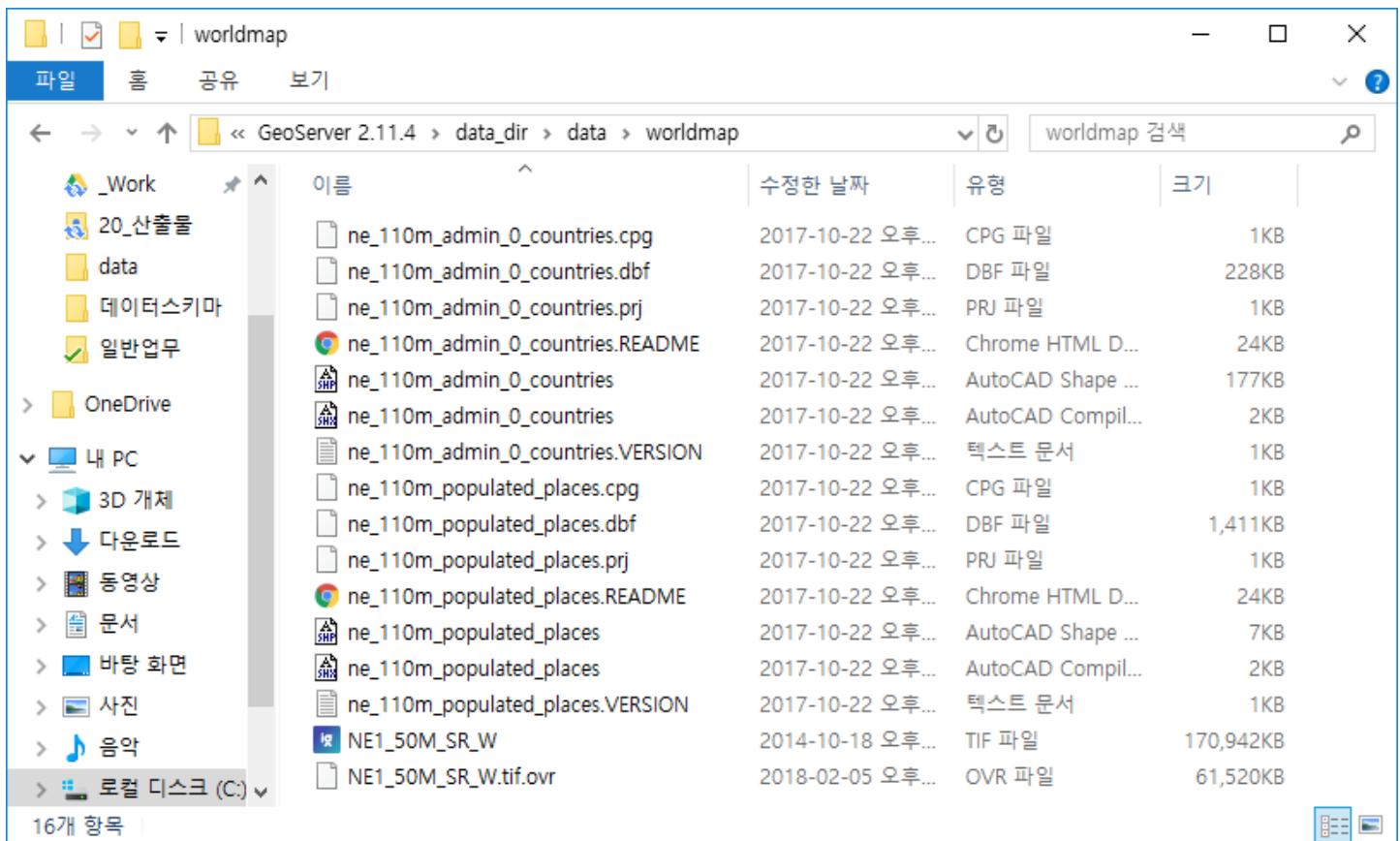
레이어 미리보기 메뉴에 들어가보면 미리 등록된 여러 레이어들을 볼 수 있습니다.

이 레이어들은 사용자의 PC를 벗어나 인터넷 상에 공간정보를 서비스할 준비가 되어 있는 것입니다.

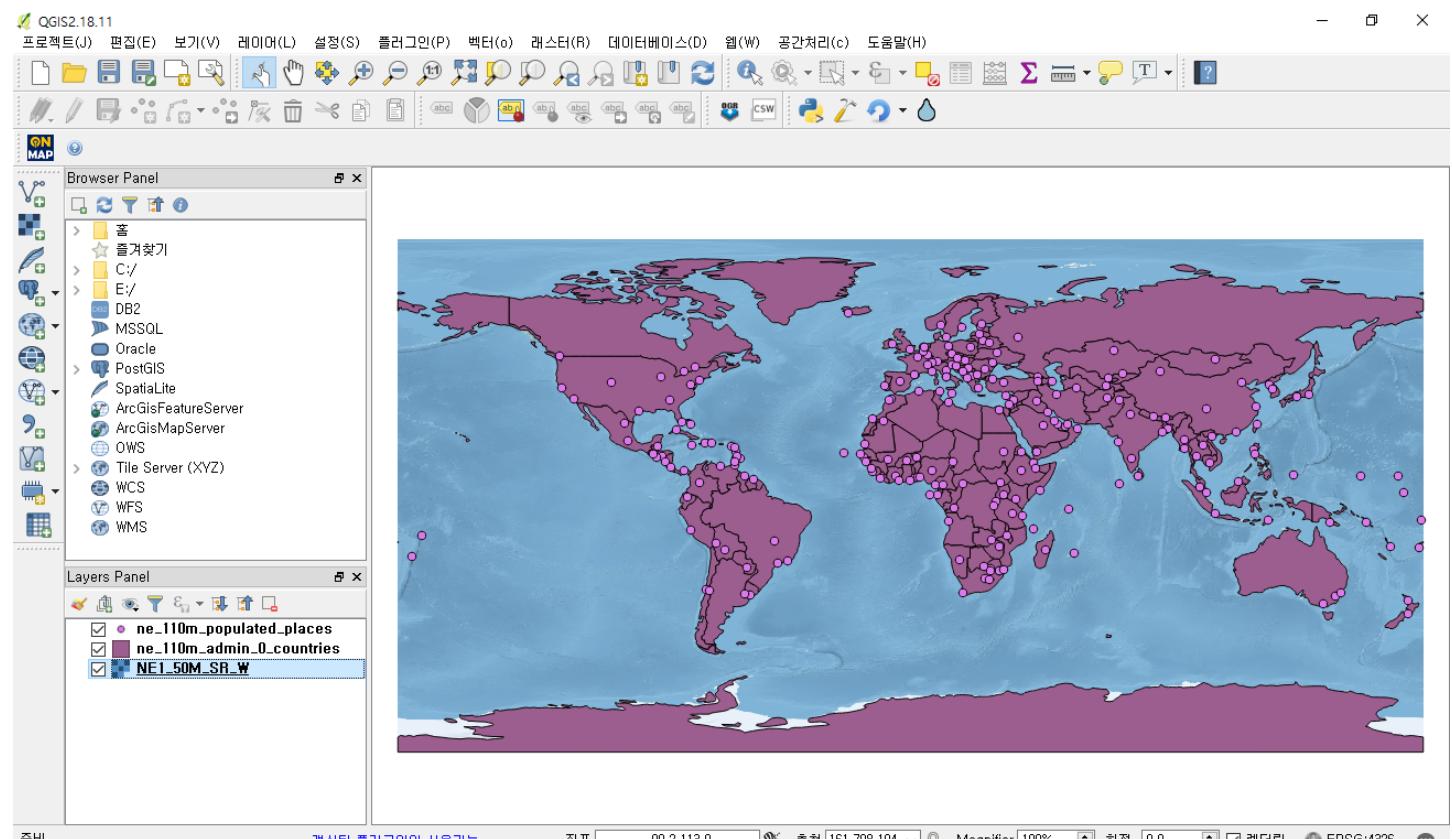
우리도 이렇게 내 컴퓨터에 저장된 공간정보를 인터넷 상에 서비스해 보도록 하겠습니다.

사용할 데이터는 미리 받아둔 Natural Earth Quick Start 세계지도 중 50m\_raster/NE1\_50M\_SR\_W 폴더의 NE1\_50M\_SR\_W.tif, 110m\_cultural 폴더의 ne\_110m\_admin\_0\_countries.shp, ne\_110m\_populated\_places.shp 3개 레이어만 사용하겠습니다.

GeoServer가 깔린 폴더에 있는 data\_dir 폴더 아래의 data 폴더에 worldmap 이란 폴더를 만들어 주세요. 압축된 파일에서 앞에 설명한 3개 레이어를 찾아 이 worldmap 폴더에 복사해 주세요. ESRI Shape 파일의 경우 확장자만 다른 관련 파일들을 모두 복사해야 함을 잊지 마세요.



QGIS에서 이 레이어들을 불러보면 다음과 같이 보입니다.



이제 GeoServer에서 이 레이어들을 등록해 보겠습니다.  
GeoServer에 레이어를 등록하는 과정은 일반적으로 다음과 같은 과정을 거칩니다.

1. 작업공간 만들기
2. 저장소 만들기
3. 레이어 만들기

작업공간은 보통 프로젝트가 새로 시작될 때 한번씩만 만들어 주면 됩니다.

작업공간

GeoServer의 작업공간을 관리합니다

새로운 작업공간 추가하기

선택된 작업공간 제거하기

<< < 1 > >> 결과: 1에서 8(8 항목 중)

Search

작업공간 이름 기본값

- cite ✓
- it.geosolutions
- nurc
- sde
- sf
- tiger
- topp

<< < 1 > >> 결과: 1에서 8(8 항목 중)

웹브라우저에 띄워놓은 GeoServer 관리화면에서 [ 데이터-작업공간 ] 메뉴를 선택합니다.  
[새로운 작업공간 추가하기] 버튼을 누릅니다.

새로운 작업공간

새로운 작업공간을 생성합니다

Name

worldmap

네임스페이스 URI

http://myservice.com

이 작업공간과 연결된 네임스페이스 URI입니다

기본 작업공간으로 설정하기

제출 취소

Name에 worldmap, 네임스페이스 URI에 <http://myservice.com>을 입력하고, 기본 작업공간으로 설정하기에 체크 후 [저장]을 누릅니다.

네임스페이스 URI에 지금은 아무 웹주소나 넣은 것이지만, 원래 공간정보를 서비스 하는 웹사이트 주소를 넣어주어야 합니다. 이 정보는 메타데이터로 사용자에게 전달됩니다.

이제 작업공간을 만들겠습니다.

관리화면에서 [저장소] 메뉴를 선택하고, [새로운 저장소 생성하기] 버튼을 누릅니다.

정보 & 상태

- 서버 상태
- GeoServer 로그
- 연락처 정보
- GeoServer 정보

데이터

- 레이어 미리보기
- 작업공간
- 저장소
- 레이어
- 레이어 그룹
- 스타일

서비스

- WMTS
- WCS
- WFS
- WMS

환경설정

- 전역 환경설정
- 이미지 프로세싱
- 래스터 액세스

## 새로운 데이터 저장소

저장소를 구성할 데이터 원본 유형을 선택하세요

### 벡터 데이터 저장소

- Directory of spatial files (shapefiles) - Takes a directory of shapefiles and exposes it as a data store
- PostGIS - PostGIS Database
- PostGIS (JNDI) - PostGIS Database (JNDI)
- Properties - Allows access to Java Property files containing Feature information
- Shapefile - ESRI(tm) Shapefiles (\*.shp)
- Web Feature Server (NG) - Provides access to the Features published a Web Feature Service, and the ability to perform transactions on the server (when supported / allowed).

### 래스터 데이터 저장소

- ArcGrid - ARC/INFO GRID Coverage Format
- GeoTIFF - Tagged Image File Format with Geographic information
- Gtopo30 - Gtopo30 Coverage Format
- ImageMosaic - Image mosaicking plugin
- WorldImage - A raster file accompanied by a spatial data file

### 기타 데이터 저장소

- WMS - 원격의 Web Map Service를 사용합니다

먼저 Directory of spatial files (shapefiles)를 선택해 ESRI Shape 파일이 있는 폴더를 저장소로 만들어 봅시다.

정보 & 상태

- 서버 상태
- GeoServer 로그
- 연락처 정보
- GeoServer 정보

데이터

- 레이어 미리보기
- 작업공간
- 저장소
- 레이어
- 레이어 그룹
- 스타일

서비스

- WMTS
- WCS
- WFS
- WMS

환경설정

- 전역 환경설정
- 이미지 프로세싱
- 래스터 액세스

타일 캐시

- 타일 레이어
- 캐시 기본 설정
- 그리드셋
- 디스크 할당량
- Blob 저장소

## 새로운 벡터 데이터 저장소 추가

새로운 벡터 데이터 저장소를 추가합니다

### 기본 저장소 정보

작업공간 \*

worldmap

데이터 저장소 이름 \*

worldmap\_shape

설명

활성화

연결 파라미터

Shapefile을 포함한 디렉토리 위치 \*

file:data/worldmap

DBF 문자셋

UTF-8

공간 인덱스가 없거나 유효하지 않은 경우 생성하기

메모리 맵 버퍼 사용(윈도우 사용 불가)

캐시 및 메모리 맵 재사용('메모리 맵 버퍼 사용' 활성화 필요)

저장 취소

데이터 저장소 이름에 worldmap\_shape 입력 후, Shapefile을 포함한 디렉토리 위치의 [탐색] 버튼 눌러 data/worldmap 폴더 선택하고, DBF 문자셋으로 UTF-8을 선택하고 [저장]을 누릅니다.  
만일 cp949로 된 한글이 있는 경우 문자셋에 x-windows-949를 선택하시면 됩니다.

## 새로운 레이어

### 정보 & 상태

-  서버 상태
-  GeoServer 로그
-  연락처 정보
-  GeoServer 정보

### 데이터

-  레이어 미리보기
-  작업공간
-  저장소
-  레이어
-  레이어 그룹
-  스타일

새로운 레이어를 추가합니다.

필드 이름과 유형을 설정하여 직접 새로운 피처 타입을 생성할 수 있습니다. [새로운 피처 타입 생성하기...](#)  
다음은 저장소에 포함된 레이어 목록입니다: 'worldmap\_shape'. 환경을 구성할 레이어를 클릭하세요.

<< < 1 > >> 결과: 1에서 2 ( 2 항목 중)

발행됨	레이어 이름	동작
	ne_110m_admin_0_countries	<a href="#">발행하기</a>
	ne_110m_populated_places	<a href="#">발행하기</a>

<< < 1 > >> 결과: 1에서 2 ( 2 항목 중)

자동으로 새로운 레이어 화면으로 전환되네요.

[발행하기] 링크를 눌러 ne\_110m\_admin\_0\_countries 레이어 부터 만들어 봅시다.

여기서 '발행하기'란 것은 영어로는 publish로 내 컴퓨터 안의 자료를 인터넷으로 내보낸다는 의미를 가지고 있습니다.

[발행하기] 링크를 누르면 옵션이 매우 많은 '새로운 레이어' 화면으로 넘어갑니다.

여기서 꼭 3가지 값을 확인해야 합니다.

### 1) 정의한 좌표체계

- 데이터의 좌표계를 지정하는 부분입니다.
- 자동으로 지정되는 경우도 있지만 안 그런 경우도 많습니다.
- 실제 자료의 좌표계와 일치하게 지정해야 합니다.

### 2) 원본 데이터 최소경계 영역

- 보통 최초에는 비어 있습니다.
- [데이터로부터 계산하기] 링크를 누르면 자동으로 값이 지정됩니다.

### 3) 위/경도 영역

- 정의한 좌표체계의 원본 데이터 최소경계 영역을 경위도로 바꾼 값입니다.
- 최초에는 보통 비어 있습니다.
- [원본 영역으로부터 계산하기] 링크를 누르면 자동으로 값이 지정됩니다.
- 자신이 알고 있는 영역값과 맞는지 꼭 확인해야 합니다.

## 레이어 편집

레이어 데이터 및 발행 정보를 편집합니다

### worldmap:ne\_110m\_admin\_0\_countries

현재 레이어의 리소스 및 발행 정보를 구성합니다

데이터  발행  차원  Tile Caching

#### 레이어 편집

##### 레이어 기본정보

이름

ne\_110m\_admin\_0\_countries

활성화

Capabilities 문서, 레이어 미리보기 등에서 정보 보기

제목

ne\_110m\_admin\_0\_countries

개요

(미리보기)

#### 키워드

현재 키워드 목록

features  
ne\_110m\_admin\_0\_countries

선택항목 제거

새로운 키워드

단어집

키워드 추가

#### 메타데이터 연결

현재까지 연결된 메타데이터가 없습니다

링크 추가 WMS 1.1.1 capabilities에서는 FGDC 및 TC211 메타데이터 연결만 표시됩니다

#### 데이터 링크

현재까지 데이터 링크가 없음

링크 추가

#### 공간 좌표 체계

원본 데이터 좌표체계

EPSG:4326  WGS84...

정의한 좌표체계

EPSG:4326  검색...  EPSG:WGS 84...

좌표체계 처리 방식

정의한 좌표체계 사용

#### 레이어 최소경계 영역

원본 데이터 최소경계 영역

Min X	Min Y	Max X	Max Y
-179.999999999999	-90.000000000000	180.000000000000	83.645130000000

데이터로부터 계산하기

SRS 범위로 계산

위/경도 영역

Min X	Min Y	Max X	Max Y
-179.999999999999	-90.000000000000	180.000000000000	83.645130000000

원본 영역으로부터 계산하기

#### 곡선 지오메트리 제어

선형 지오메트리는 원호를 포함할 수 있습니다

선형화 허용오차 (데이터가 곡선 지오메트리를 포함하는 경우에만 유용)

이 값들을 지정한 다음에는 [발행] 탭을 눌러 캐시와 심볼에 관한 설정을 해야 합니다.

## 레이어 편집

레이어 데이터 및 발행 정보를 편집합니다

### worldmap:ne\_110m\_admin\_0\_countries

현재 레이어의 리소스 및 발행 정보를 구성합니다

데이터      **발행**      차원      Tile Caching

#### HTTP 설정

##### 캐시 설정

응답 캐시 헤더

캐시 시간 (초)

31536000

#### WFS 설정

##### 피처 설정

요청 당 피처 제한

0

소수점 최대 자리수

0

##### NumberMatched 속성 건너뛰기

NumberMatched 속성 계산 건너뛰기

#### WFS Capabilities 생성을 위한 추가 SRS 코드

WFS 확장 SRS 목록 재정의

#### WMS 설정

##### 레이어 설정

조회가능

투명

기본 스타일

line



추가 스타일

사용가능한 스타일 목록

poly\_landmarks

polygon

pophatch

population

rain

raster

restricted

simple\_roads

simple\_streams

tiger\_roads



선택된 스타일

HTTP 설정 부분의 응답 캐시 헤더는 HTTP 규약에 따라 이미지 등 컨텐츠를 웹브라우저가 얼마동안 캐시를하게 할지 설정하는 헤더를 불일지를 결정하는 옵션입니다.

그 아래 있는 값인 캐시시간이 얼마나 캐시가 지속될지 시간을 초단위로 나타낸 것입니다.

이 캐시시간을 31536000로 지정한 것은 60(초) \* 60(분) \* 24(시간) \* 365(일) 한 값으로 1년을 의미합니다.

그리고 보통 이 화면에서 바꿔 주어야 할 것이 WMS 설정 부분의 기본 스타일입니다.

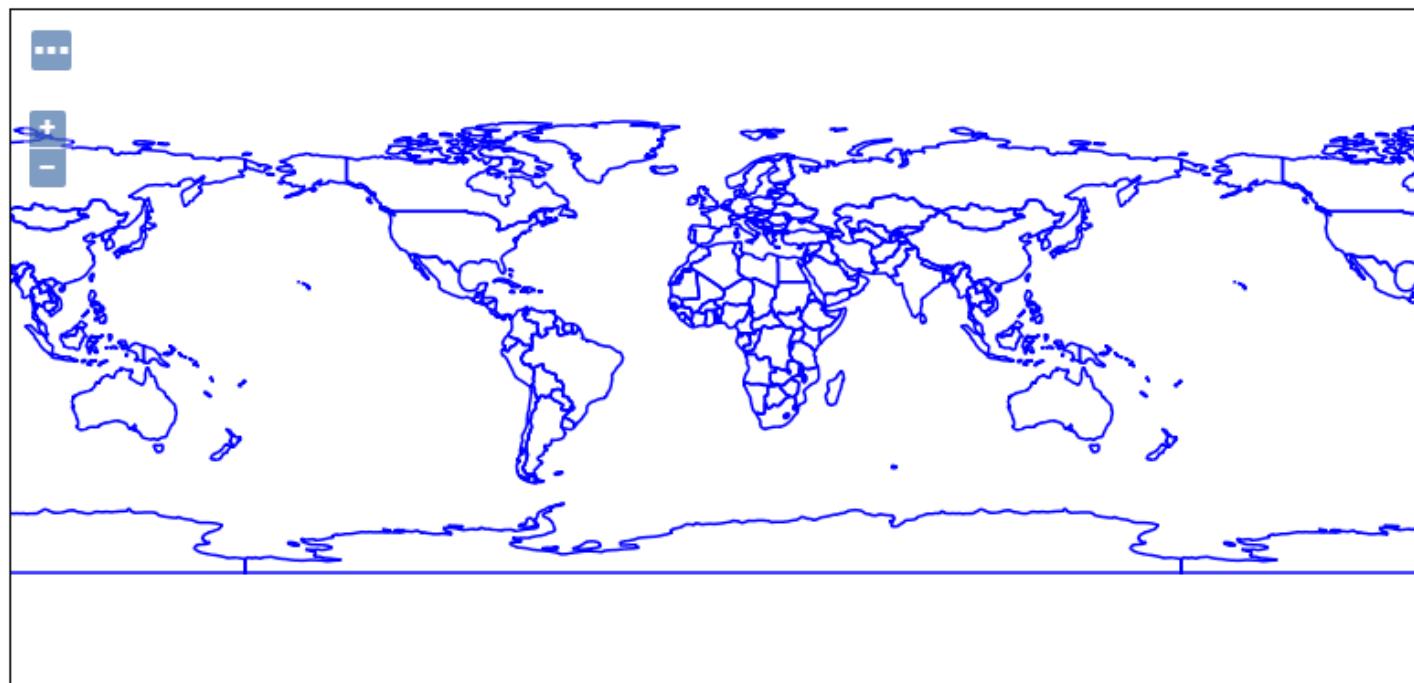
우리는 국가 경계를 선으로 나타내 아래의 영상이 보이게 할 것이니 일단 line을 선택합니다.

여기서 주의 할 것이 아래에 있는 사용 가능한 스타일 목록에서 선택을 바꾸는 것이 아니라 기본 스타일에서 바꿔야 한다는 것입니다.

이제 화면 제일 하단의 [저장]을 눌러 저장합니다.

잘 발행되었는지 [레이어 미리보기] 메뉴로 가서 확인해 봅시다.

ne\_110m\_admin\_0\_countries 레이어를 찾아 [OpenLayers] 링크를 누르면 동적으로 이동/확대 가능한 지도로 표시됩니다.



생각보다 이쁘지 않지만 만들어지긴 했네요.

이제 나머지 레이어도 등록하기 위해서 [레이어] 메뉴를 누르고 [새로운 레이어 추가하기] 버튼을 누릅니다. 저장소로 worldmap:worldmap\_shape 을 선택하고 아직 미발행 상태인 ne\_110m\_populated\_places 레이어의 [발행하기] 링크를 누릅니다.

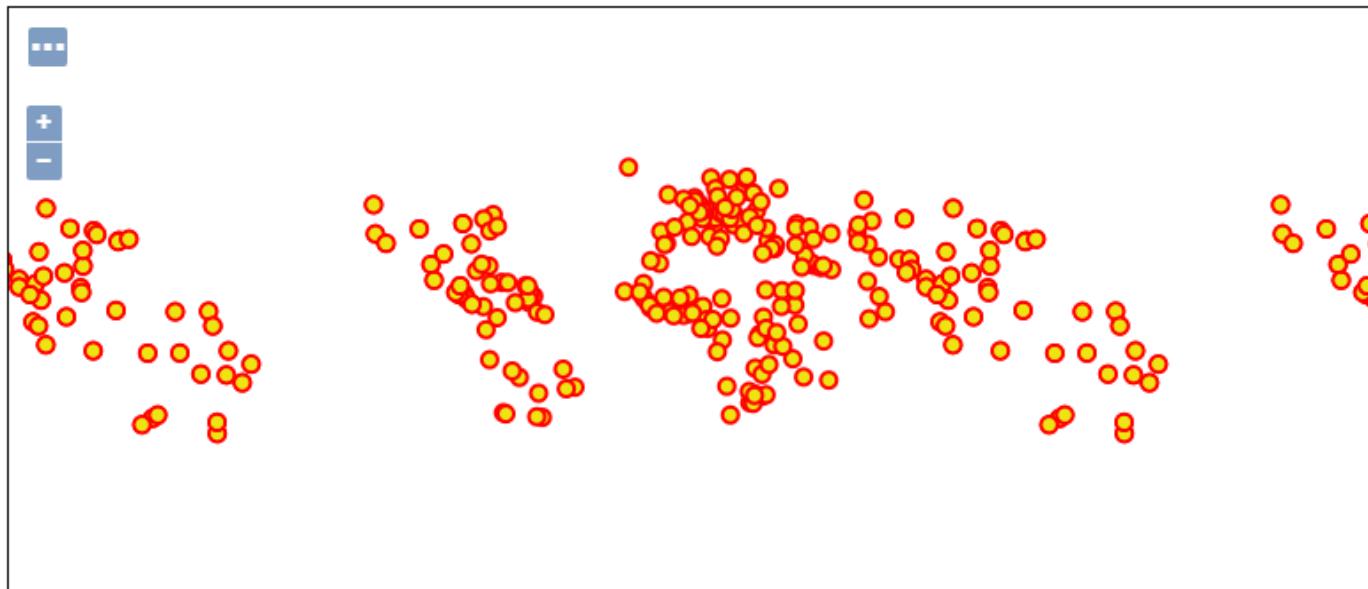
정의한 좌표계 부분의 값을 확인하고, [데이터로부터 계산하기], [원본 영역으로부터 계산하기]를 누릅니다.

[발행] 탭으로 가서 응답 캐시 헤더를 켜고, 캐시 시간으로 31536000을 입력합니다.

기본 스타일은 'poi'를 선택합니다.

[저장]을 눌러 저장합니다.

[레이어 미리보기] 메뉴를 누르고 레이어를 찾아 미리보기를 합니다.



이제 래스터 데이터를 등록해 보겠습니다.

래스터 데이터는 각 파일을 하나씩의 저장소를 만들어 주어야 합니다.  
[저장소] 메뉴를 선택하시고, [새로운 저장소 생성하기] 버튼을 누릅니다.

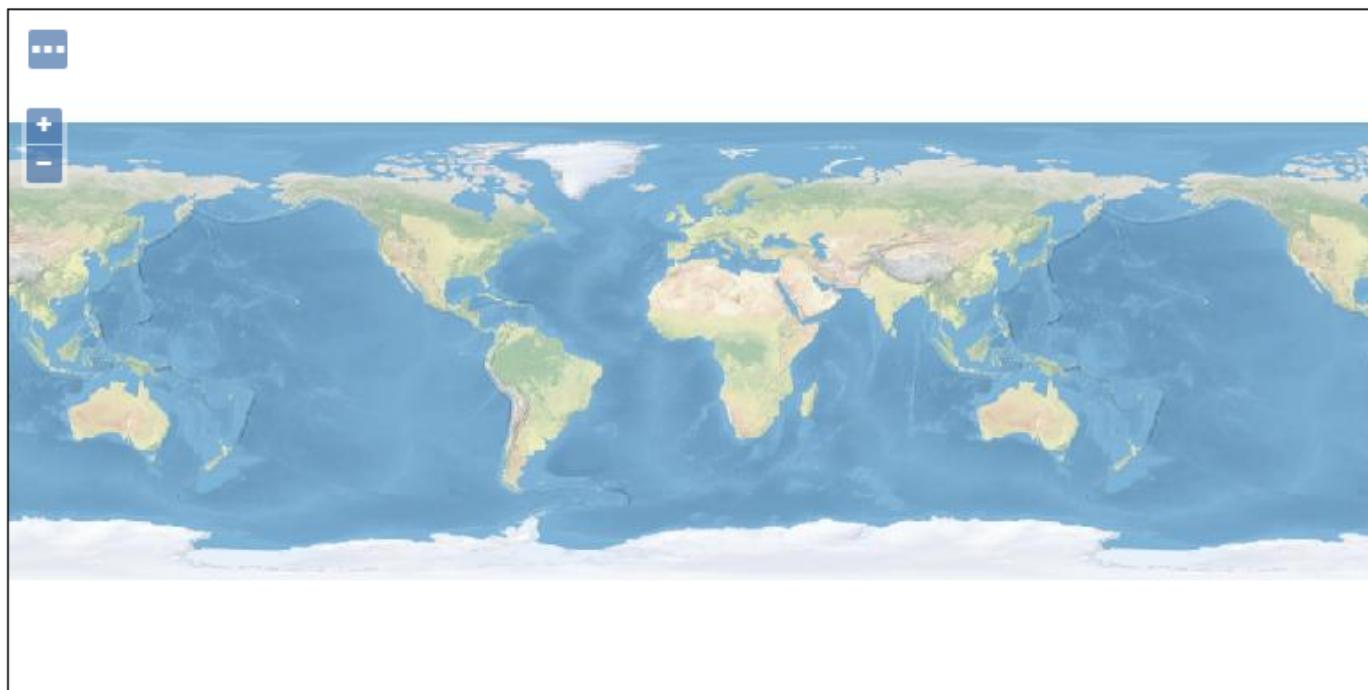
[GeoTIFF] 링크를 누릅니다.

URL 부분에서 [탐색...] 버튼을 누르고, data/worldmap/NE1\_50M\_SR\_W.tif 파일을 선택합니다.  
데이터 저장소 이름에 NE1\_50M\_SR\_W를 입력하고 [저장] 버튼을 눌러 완료합니다.

새로운 레이어 화면에서 [발행하기] 링크를 누릅니다.  
정의한 좌표체계, 원본 데이터 최소경계 영역, 위/경도 영역을 확인합니다.

[발행] 탭을 누르고, 응답 캐시 헤더에 체크하고 캐시 시간에 31536000 입력하고 [저장]을 누릅니다.

[레이어 미리보기]에서 레이어를 찾아 미리보기 합니다.



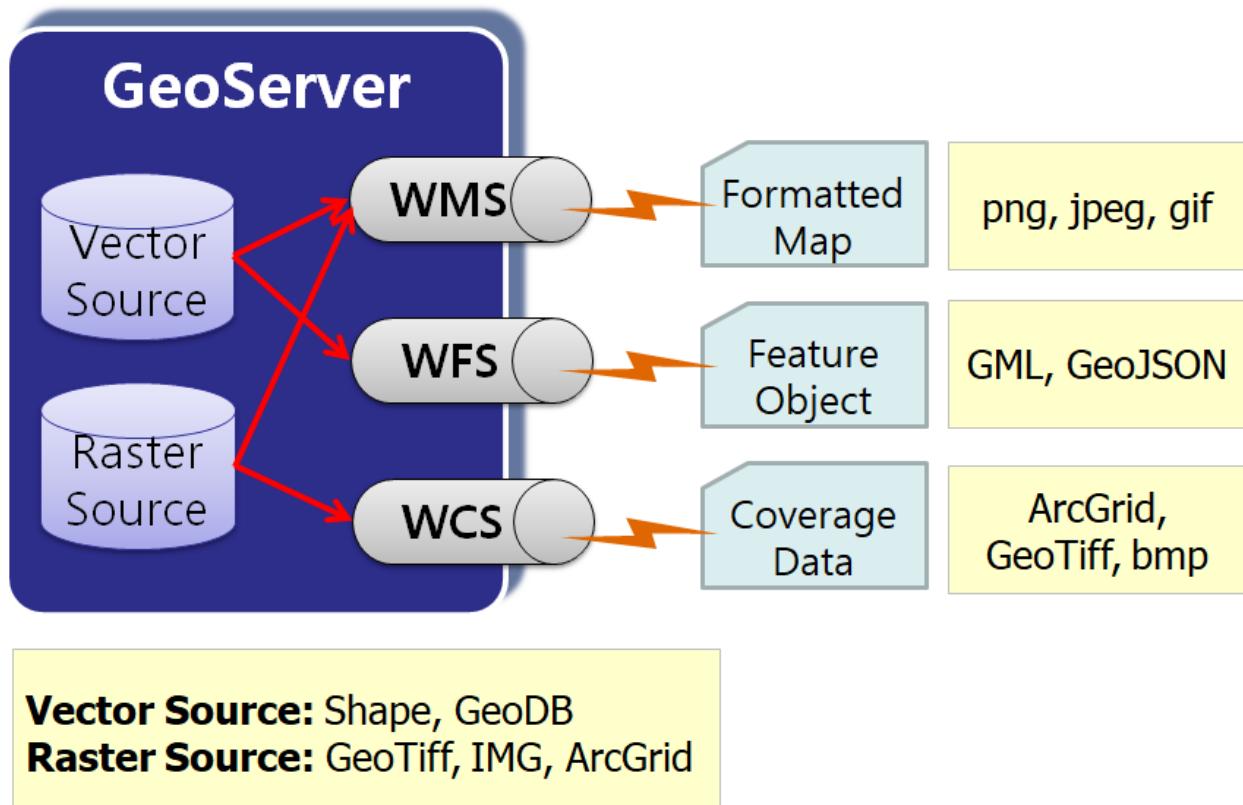
Scale = 1 : 279M  
Click on the map to get feature info

## 5x02 OGC 웹서비스 이해

지금 배우고 있는 GeoServer의 역할을 한 문장으로 기술하면 다음과 같습니다.

### Geospatial Gateway Server

다양한 공간 Data를 인터넷용 공간데이터 표준 인터페이스로 공급하는 프로그램이란 뜻입니다.



GeoServer가 제공하는 대표적인 공간정보 표준 인터페이스에는 WMS, WFS, WCS 등이 있습니다. 이 인터페이스를 통해 컴퓨터에 저장된 벡터 데이터와 래스터 데이터를 인터넷 상에서 활용하기 적합한 형태로 바꿔 서비스 해 줍니다.

벡터 데이터는 WMS와 WFS로 서비스 될 수 있고, 래스터 데이터는 WMS와 WCS로 서비스 될 수 있습니다.

이런 서비스 표준을 정한 기관이 OGC(Open Geospatial Consortium)입니다. 그래서 통칭해 OGC Web Service(OWS), 우리말로 OGC 웹서비스라 합니다.

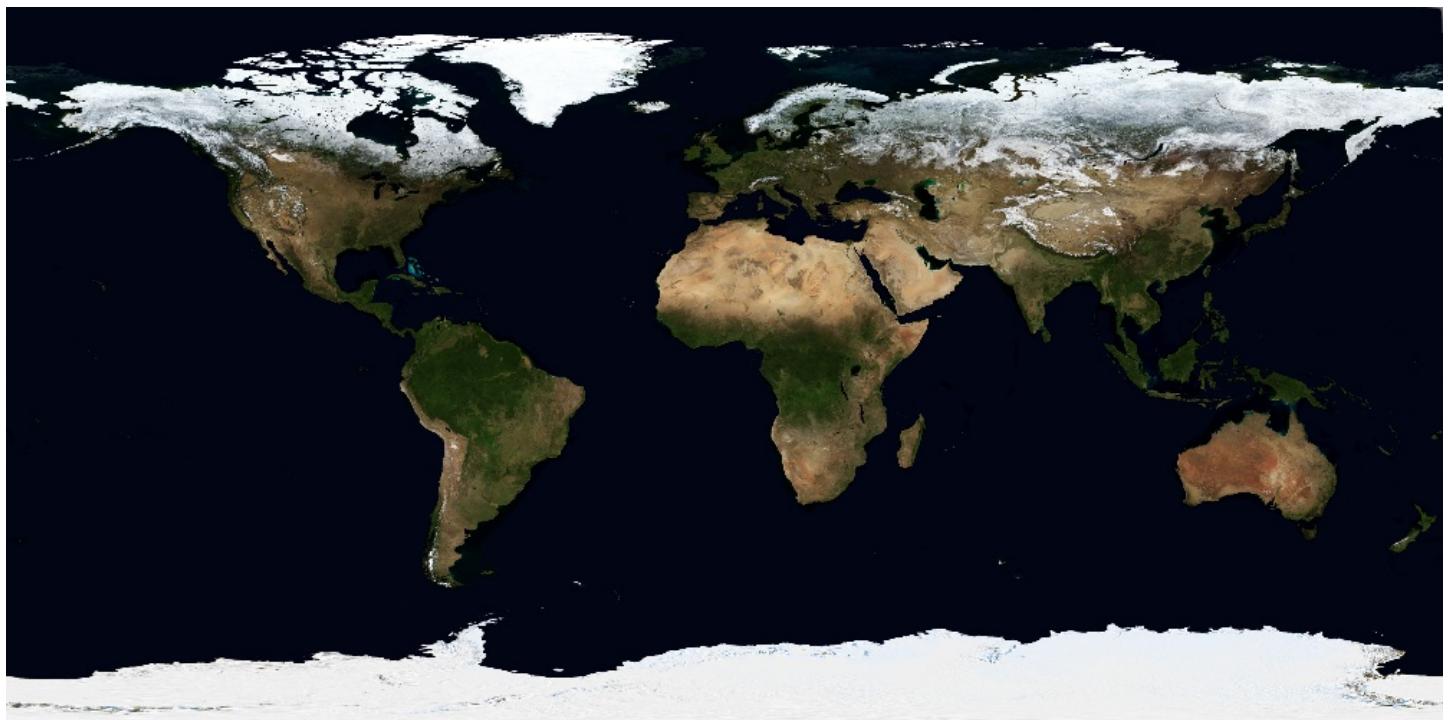
이제 서비스 인터페이스를 중심으로 하나씩 살펴보겠습니다.

## ■ Web Map Service (WMS)

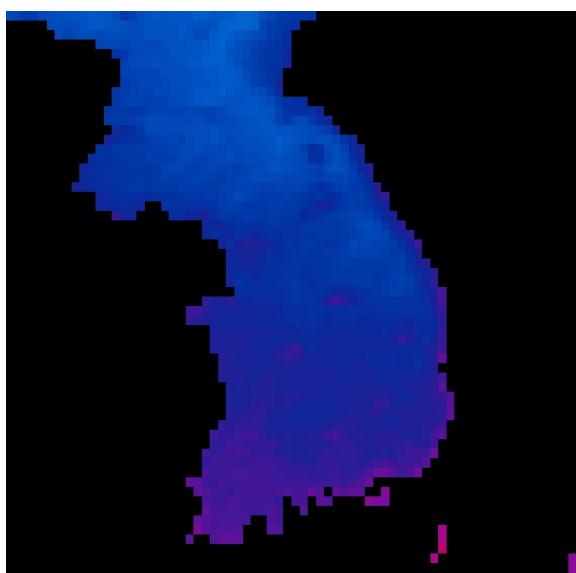
- 지도 이미지 등 스타일을 가진 공간데이터를 인터넷으로 서비스하는 표준입니다.
- 요청방법과 응답형식을 정의하고 있습니다.
- 보통 URI(웹주소+파라미터)로 요청하고 이미지로 응답을 받습니다.
- 지도요청, 카타로그조회, 속성조회 등이 가능합니다.
- 필수: GetCapabilities , GetMap
- 옵션: GetFeatureInfo, DescribeLayer, GetLegendGraphic

실제 사례를 중심으로 익혀보겠습니다.

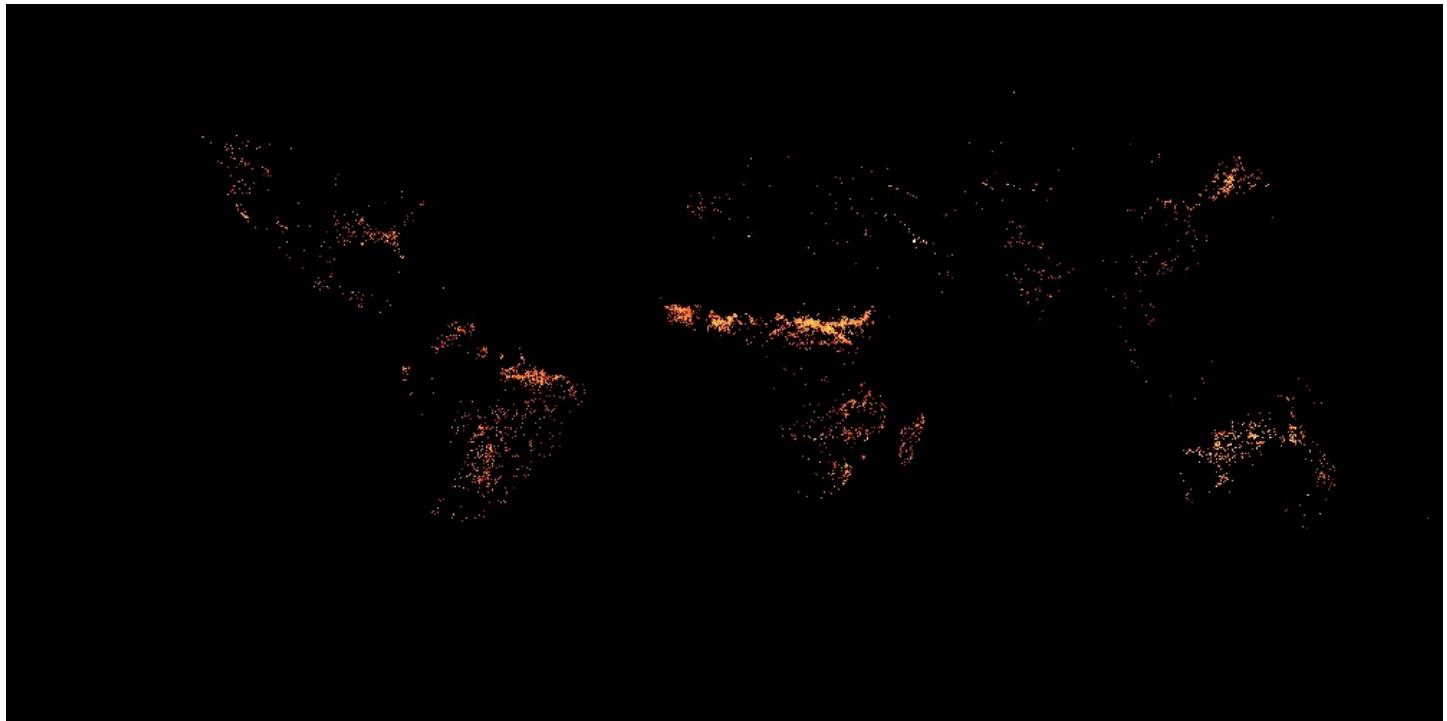
<http://neowms.sci.gsfc.nasa.gov/wms/wms?version=1.1.1&SERVICE=WMS&REQUEST=GetMap&LAYERS=BlueMarbleNG&FORMAT=image/jpeg&SRS=EPSG:4326&BBOX=-180,-90,180,90&WIDTH=1000&HEIGHT=500> [링크]



[http://neowms.sci.gsfc.nasa.gov/wms/wms?version=1.1.1&SERVICE=WMS&REQUEST=GetMap&LAYERS=MOD\\_LSTN\\_CLIM\\_M&FORMAT=image/jpeg&SRS=EPSG:4326&BBOX=124,34,131,40&WIDTH=600&HEIGHT=600](http://neowms.sci.gsfc.nasa.gov/wms/wms?version=1.1.1&SERVICE=WMS&REQUEST=GetMap&LAYERS=MOD_LSTN_CLIM_M&FORMAT=image/jpeg&SRS=EPSG:4326&BBOX=124,34,131,40&WIDTH=600&HEIGHT=600) [링크]



[\[링크\]](http://neowms.sci.gsfc.nasa.gov/wms/wms?version=1.1.1&SERVICE=WMS&REQUEST=GetMap&LAYERS=MOD14A1_M_FIRE&FORMAT=image/jpeg&SRS=EPSG:4326&BBOX=-180,-90,180,90&&WIDTH=1000&HEIGHT=500)



[\[링크\]](http://61.43.91.75:8088/MLTMServlet/wmts?SERVICE=WMTS&REQUEST=GetTile&VERSION=1.0.0&LAYER=STD_LINK&TILEMATRIXSET=EPSG:900913&TILEMATRIX=EPSG:900913:13&TILEROW=3173&TILECOL=6984&FORMAT=image/gif)



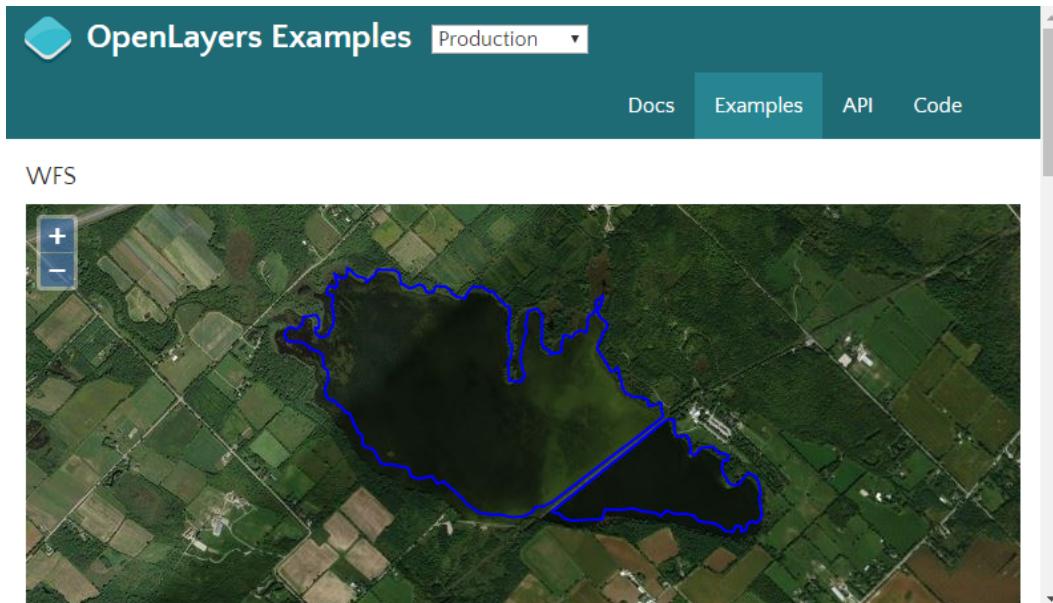
## ■ Web Feature Service (WFS)

- 벡터 형식의 공간정보 피처(Feature: Geometry+attribute)를 인터넷으로 서비스하는 표준입니다.
- 요청방법과 응답형식을 정의하고 있스빈다.
- 보통 URI로 요청하고 XML, GeoJSON 등으로 응답을 받습니다.
- 피처요청, 카타로그조회, 속성조회 등이 가능합니다.
- 필수: GetCapabilities , DescribeFeatureType , GetFeature
- 옵션: LockFeature, Transaction

실제 사례로 익혀보겠습니다.

다음 사이트에 가 보시면 WFS 사용 예가 있습니다.

<http://openlayers.org/en/latest/examples/vector-wfs.html?q=wfs>



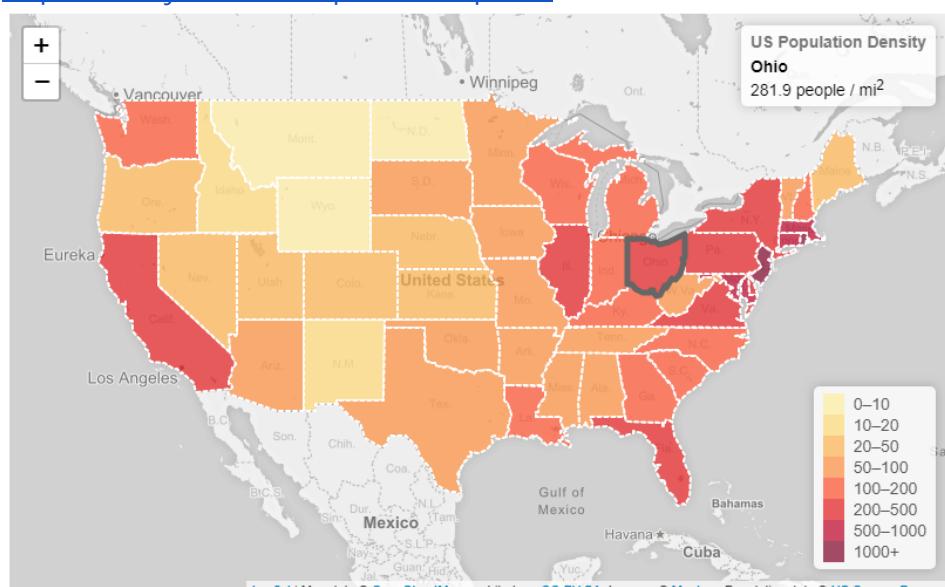
영상 위에 파란선들이 WFS로 받아온 데이터들입니다.

내부적으로는 이렇게 요청됩니다.

[https://ahocevar.com/geoserver/wfs?service=WFS&version=1.1.0&request=GetFeature&typename=osm:water\\_areas&outputFormat=application/json&srsname=EPNG:3857&bbox=-8937896.372928089,5370465.607316485,-8879880.668459637,5393396.715802036,EPNG:3857](https://ahocevar.com/geoserver/wfs?service=WFS&version=1.1.0&request=GetFeature&typename=osm:water_areas&outputFormat=application/json&srsname=EPSG:3857&bbox=-8937896.372928089,5370465.607316485,-8879880.668459637,5393396.715802036,EPNG:3857) [링크]

다음 샘플은 더 WFS를 잘 활용하고 있습니다.

<http://leafletjs.com/examples/choropleth/>

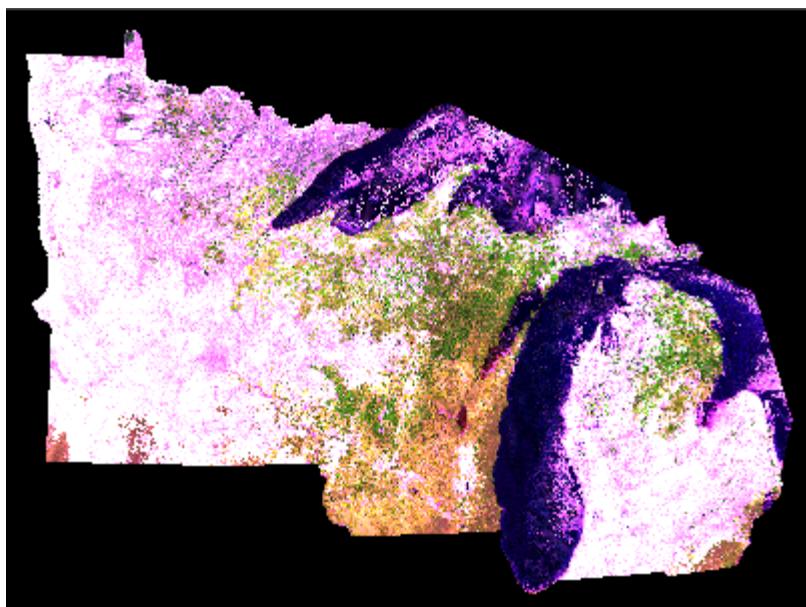


## ■ Web Coverage Service (WCS)

- 래스터 형태의 커버리지(좌표가 있는 래스터) 공간데이터를 인터넷으로 서비스하는 표준입니다.
- 요청방법과 응답형식을 정의하고 있습니다.
- 보통 URI로 요청하고 래스터 파일로 응답을 받습니다.
- 커버리지요청, 카타로그조회 등이 가능합니다.
- 필수: GetCapabilities , DescribeCoverage , GetCoverage

실제 사례를 보겠습니다.

[http://demo.mapserver.org/cgi-bin/wcs?SERVICE=wcs&VERSION=1.0.0&REQUEST=GetCoverage&COVERAGE=modis-001&CRS=EPSG:26915&BBOX=159707,4597395,1400707,5501395&WIDTH=400&HEIGHT=300&FORMAT=GEOTIFF\\_RGB](http://demo.mapserver.org/cgi-bin/wcs?SERVICE=wcs&VERSION=1.0.0&REQUEST=GetCoverage&COVERAGE=modis-001&CRS=EPSG:26915&BBOX=159707,4597395,1400707,5501395&WIDTH=400&HEIGHT=300&FORMAT=GEOTIFF_RGB) [\[링크\]](#)



<http://demo.mapserver.org/cgi-bin/wcs?SERVICE=wcs&VERSION=1.0.0&REQUEST=GetCapabilities> [\[링크\]](#)

<http://demo.mapserver.org/cgi-bin/wcs?SERVICE=wcs&VERSION=1.0.0&REQUEST=DescribeCoverage&COVERAGE=modis> [\[링크\]](#)

OWS 서비스에 대한 더 많은 정보는 다음 링크에서 찾으실 수 있습니다.

<http://www.nsdi.go.kr/?menuno=2926>

## 5x03 레이어 그룹과 스타일

앞에서 실습한 각각의 레이어들은 이제 인터넷에 서비스 되어 인터넷 연결만 되면 어디서든 불러서 활용할 수 있는 상태가 되었습니다.

하지만 아직은 전문적인 서비스로 보기에는 어설픈 점이 많습니다.

우선 이 3개 레이어를 묶어서 하나의 레이어처럼 서비스 해 봅시다.

GeoServer 관리자 화면에서 [레이어 그룹] 메뉴를 선택하고 [새로운 레이어 그룹 생성하기] 버튼을 누릅니다.

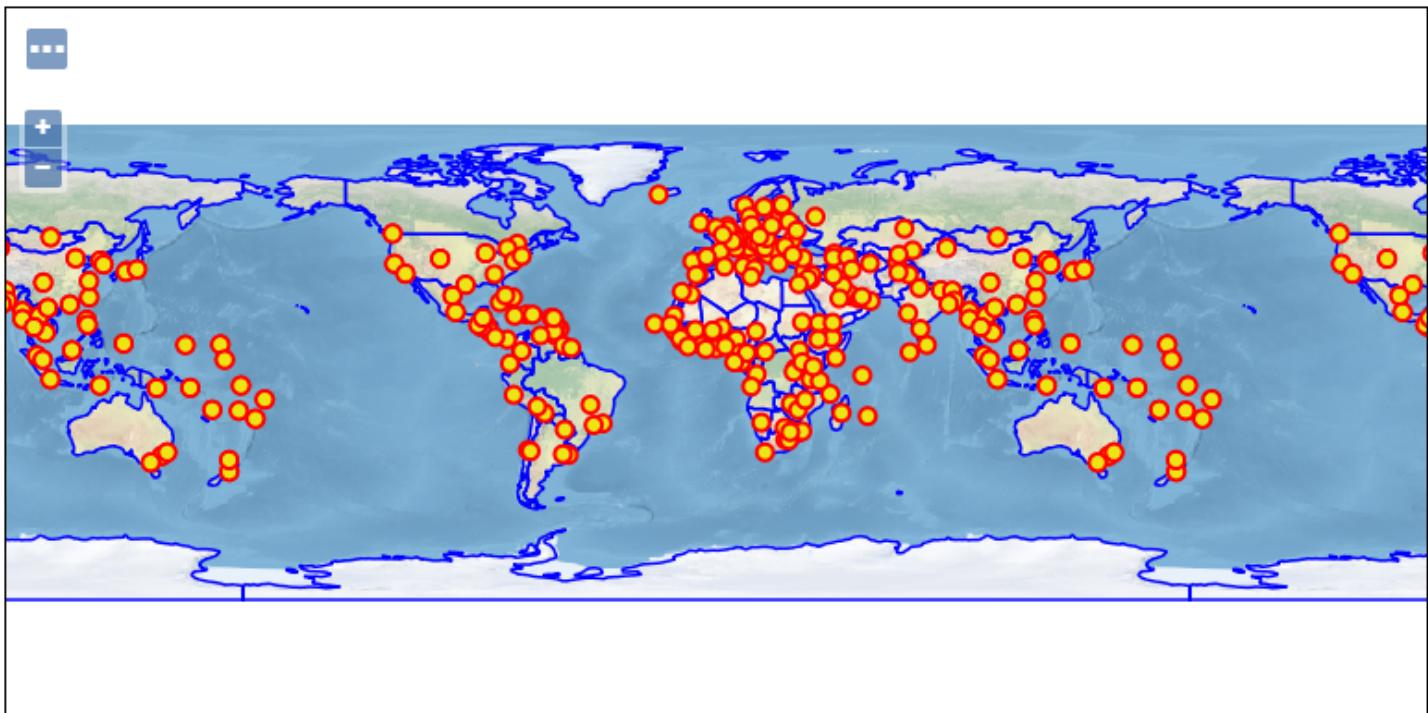
The screenshot shows the GeoServer Layer Manager interface. At the top, there are input fields for 'Min X' (-179.99999999999), 'Min Y' (-90.0000000000000), 'Max X' (180.0000000000000), and 'Max Y' (90). Below these are dropdown menus for 'EPSG:4326' and '검색...' (Search...), and another for 'EPSG:WGS 84...'. There are two buttons: '데이터 최소경계 영역 계산하기' (Calculate Data Boundary Area) and 'CRS로부터 최소경계 계산하기' (Calculate Boundary from CRS). Under the 'Mode' section, 'Single' is selected. In the 'Layer Group' section, '조회 가능' (Searchable) is checked. The 'Layer List' section contains three layers: 'worldmap:NE1\_50M\_SR\_W', 'worldmap:ne\_110m\_admin\_0\_countries', and 'worldmap:ne\_110m\_populated\_places'. Each layer has a '기본 스타일' (Basic Style) checkbox (unchecked), a '스타일' (Style) column showing 'raster', 'line', and 'poi' respectively, and a '제거' (Remove) button. At the bottom, there are navigation buttons (<<, <, 1, >, >>) and a message '결과: 1에서 3 ( 3 항목 중 )' (Result: 1 to 3 ( 3 items total )).

[데이터] 탭에서 [레이어 추가하기...] 버튼을 눌러 NE1\_50M\_SR\_W, ne\_110m\_admin\_0\_countries, ne\_110m\_populated\_places 레이어들을 차례로 추가합니다.

[데이터 최소경계 영역 계산하기] 버튼을 눌러 영역 값을 입력합니다.

이름에 'worldmap'을 입력하고 [저장]을 누릅니다.

[레이어 미리보기] 메뉴로 가서 worldmap 그룹을 지도로 봅시다.



Scale = 1 : 279M

Click on the map to get feature info

다 합치니 좀 나아 보이는군요.

하지만 우리가 목표로 한 <https://thetruesize.com/> 사이트와 거리는 아직 커 보이네요.

기능은 아직 웹 지도 콘트롤인 OpenLayers를 배우지 않았으니 어쩔 수 없지만 지도의 아름다움에서도 큰 차이가 납니다.

이제 이 차이를 좁혀보겠습니다.

지도의 스타일은 [스타일] 메뉴와 [레이어] 메뉴의 레이어 편집에서 [발행] 탭의 WMS 설정의 기본 스타일 부분을 조합해서 관리하게 됩니다.

먼저 국가경계를 진한 회색으로 바꾸고, 국가명도 표시해 보겠습니다.

[스타일] 메뉴로 가셔서 [새로운 스타일 추가하기...] 버튼을 선택합니다.

스타일

GeoServer에 의해 발행된 스타일을 관리합니다

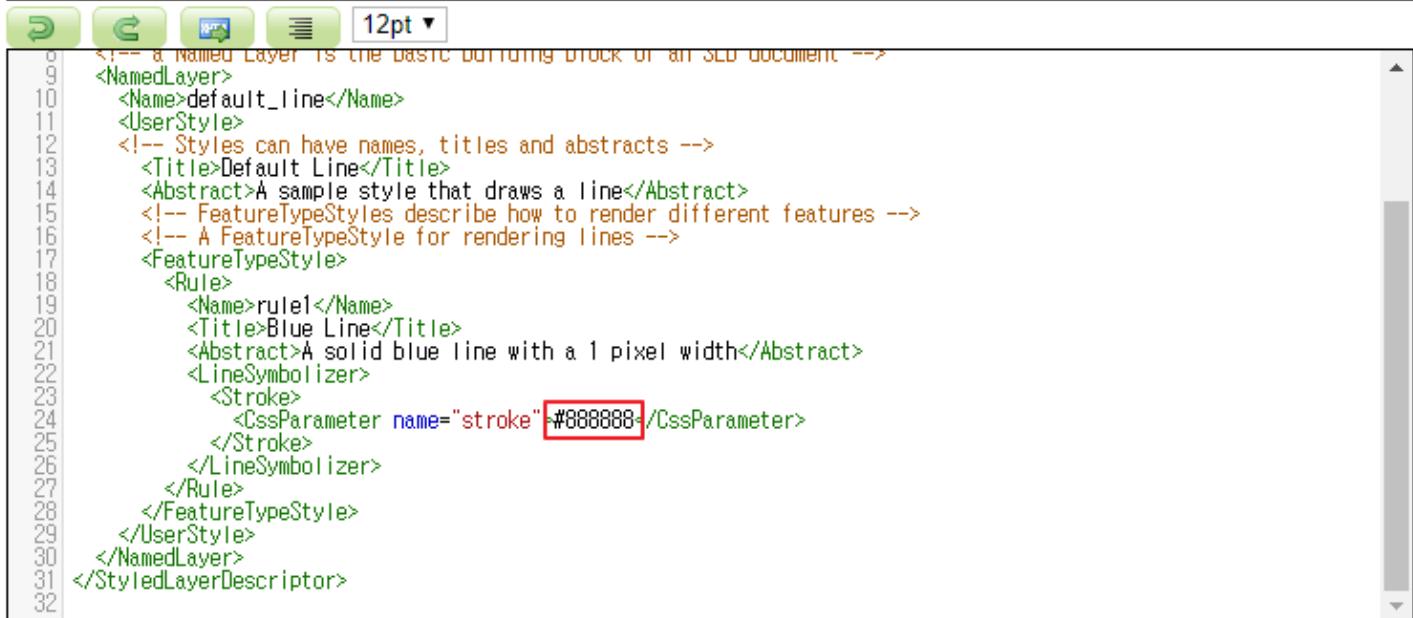
새로운 스타일 추가하기

선택된 스타일 제거하기

<< < 1 > >> 결과: 1에서 22 ( 22 항목 중)

스타일 이름	작업공간
burg	
capitals	
cite_lakes	
dem	

## 스타일 편집기



```
<!-- a Named Layer is the basic building block of an SLD document -->
<NamedLayer>
  <Name>default_line</Name>
  <UserStyle>
    <!-- Styles can have names, titles and abstracts -->
    <Title>Default Line</Title>
    <Abstract>A sample style that draws a line</Abstract>
    <!-- FeatureTypeStyles describe how to render different features -->
    <!-- A FeatureTypeStyle for rendering lines -->
    <FeatureTypeStyle>
      <Rule>
        <Name>rule1</Name>
        <Title>Blue Line</Title>
        <Abstract>A solid blue line with a 1 pixel width</Abstract>
        <LineSymbolizer>
          <Stroke>
            <CssParameter name="stroke">#888888</CssParameter>
          </Stroke>
        </LineSymbolizer>
      </Rule>
    </FeatureTypeStyle>
  </UserStyle>
</NamedLayer>
</StyledLayerDescriptor>
```

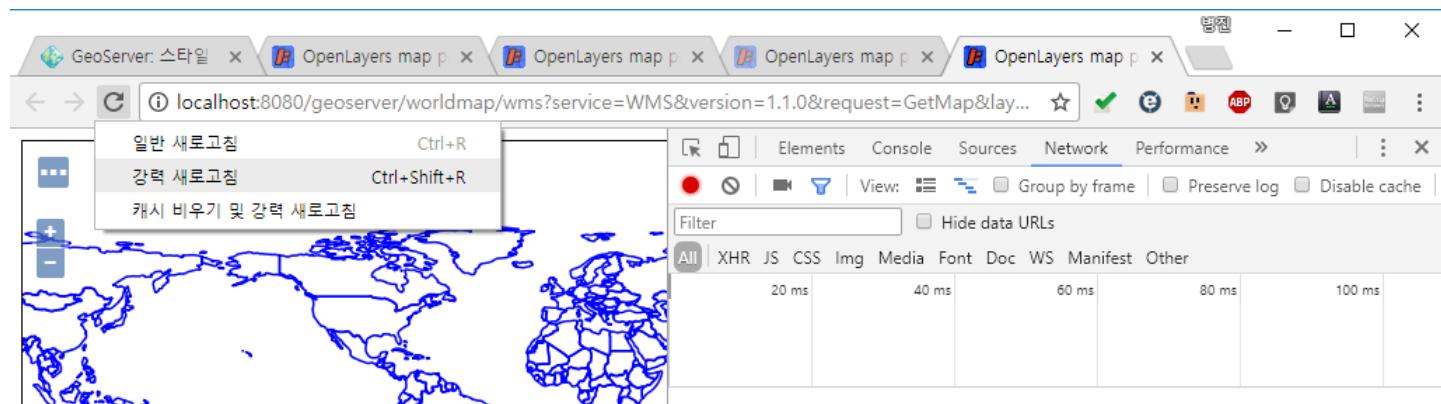
유효성 검증하기   적용하기   제출   취소

스타일 이름으로 world\_admin 을 입력하고,  
기본 스타일로 부터 복사합니다. 아래 콤보박스에서 Line을 선택 후, [복사하기...] 링크를 누릅니다.  
스타일 편집기에서 stroke 부분을 찾아 그 뒤의 값을 #888888 으로 바꾸고 [제출]을 눌러 world\_admin  
스타일 만들기를 끝냅니다.

[레이어] 메뉴를 선택하고, ne\_110m\_admin\_0\_countries 레이어를 찾아 선택하여 '레이어 편집' 화면으로  
갑니다.

[발행] 탭을 선택하고 WMS 설정 부분에서 기본스타일을 world\_admin으로 바꿉니다.  
[저장]을 눌러 변경사항을 저장합니다.

[레이어 미리보기]로 가서 ne\_110m\_admin\_0\_countries 레이어를 미리보기 합니다.  
색상이 안 변했다면, 이것은 캐시의 부작용입니다.  
더 확대해 보면 회색 선으로 보이는데 처음 화면은 아무리 새로고침을 해도 파란 선으로 보이네요.



Chrome에서 [F12]를 눌러 개발자 도구를 띄우고, 새로고침 버튼을 길게 누르면 추가 기능들이 나옵니다.  
여기서 강력 새로고침 메뉴를 선택하면 캐시를 지우고 다시 불러옵니다.  
이제 첫 화면부터 회색 선으로 보이지요?

그러나 아직 만족스럽지 않습니다. 나라 이름을 표시해야 하겠습니다.

스타일 편집이 UI에서 선택하기만 하면 되는 것이 아니라 XML을 직접 편집해야 하는 것이라 어찌 해야할지  
막막합니다.

다행이 다음 경로에서 좋은 샘플들을 많이 찾을 수 있습니다.

<http://docs.geoserver.org/stable/en/user/styling/sld/cookbook/>

SLD Cookbook에서 목차에 있는 Polygons 항목을 선택해 이동합니다.

그림들을 주욱 보며 아~ 저런 표현들도 가능하구나 하다 보니 글자가 폴리곤 중심에 찍혀 있는 그림이 보이네요.

조금 더 내리는 Label Halo 부분에 글자가 찍히고 주변에 테두리까지 있는 예제가 보입니다.

```
<FeatureTypeStyle>
  <Rule>
    <PolygonSymbolizer>
      <Fill>
        <CssParameter name="fill">#40FF40</CssParameter>
      </Fill>
      <Stroke>
        <CssParameter name="stroke">#FFFFFF</CssParameter>
        <CssParameter name="stroke-width">2</CssParameter>
      </Stroke>
    </PolygonSymbolizer>
    <TextSymbolizer>
      <Label>
        <ogc:PropertyName>name</ogc:PropertyName>
      </Label>
      <Halo>
        <Radius>3</Radius>
        <Fill>
          <CssParameter name="fill">#FFFFFF</CssParameter>
        </Fill>
      </Halo>
    </TextSymbolizer>
  </Rule>
</FeatureTypeStyle>
```

XML 부분을 모두 복사합니다.

다시 GeoServer 관리자 화면으로 가서 [스타일] 메뉴를 선택합니다.

world\_admin 스타일을 선택하고 스타일 편집기에서 17~28 행의 부분을 복사한 텍스트로 바꿉니다.

stroke 값은 #888888 으로 바꾸고

<Fill> ... </Fill> 부분은 삭제합니다.

<TextSymbolizer> 부분에서 <Label>의 <ogc:PropertyName>name</ogc:PropertyName> 부분이 텍스트를 표시할 컬럼의 이름을 의미합니다.

그런데 우리는 ne\_110m\_admin\_0\_countries 레이어에 어떤 컬럼들이 있는지 잘 모르네요.



Scale = 1 : 35M

#### ne\_110m\_admin\_0\_countries

fid	scalerank	featurecla	LABELRANK	SOVEREIGNT	SOV_A3	ADM0_DIF	LEVEL	TYPE	ADMIN
ne_110m_admin_0_countries.88	1	Admin-0 country	2.0	South Korea	KOR	0.0	2.0	Sovereign country	South Korea

아까 띄워 놓았던 ne\_110m\_admin\_0\_countries 레이어 미리보기 화면에서 화면을 확대하고 한국 부분을 클릭하면 해당 위치 객체의 속성을 볼 수 있습니다.

ADMIN라는 컬럼을 이용하는 것이 좋아 보이는군요.

다시 스타일 편집기로 돌아와 <ogc:PropertyName>의 값을 ADMIN으로 수정 후 [제출] 합니다.

수정된 XML은 다음과 같습니다.

```

<?xml version="1.0" encoding="UTF-8"?>
<StyledLayerDescriptor version="1.0.0"
xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
xmlns="http://www.opengis.net/sld"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <!-- a Named Layer is the basic building block of an SLD document -->
  <NamedLayer>
    <Name>default_line</Name>
    <UserStyle>
      <!-- Styles can have names, titles and abstracts -->
      <Title>Default Line</Title>
      <Abstract>A sample style that draws a line</Abstract>
      <!-- FeatureTypeStyles describe how to render different features -->
      <!-- A FeatureTypeStyle for rendering lines -->
      <FeatureTypeStyle>
        <Rule>
          <PolygonSymbolizer>
            <Stroke>
              <CssParameter name="stroke">#888888</CssParameter>

```

```

<CssParameter name="stroke-width">2</CssParameter>
</Stroke>
</PolygonSymbolizer>
<TextSymbolizer>
  <Label>
    <ogc:PropertyName>ADMIN</ogc:PropertyName>
  </Label>
  <Halo>
    <Radius>3</Radius>
    <Fill>
      <CssParameter name="fill">#FFFFFF</CssParameter>
    </Fill>
  </Halo>
</TextSymbolizer>
</Rule>
</FeatureTypeStyle>
</UserStyle>
</NamedLayer>
</StyledLayerDescriptor>

```

이제 미리보기 화면으로 돌아와 화면을 움직여 보면, 다음과 같이 회색선에 나라 이름이 나옴을 확인할 수 있습니다.



하지만 아직 만족스럽지는 않습니다. 표시되는 위치가 치우쳐 있고, 폰트도 마음에 안드네요. 폴리곤의 중앙에 나라이름이 오게 하고 폰트는 '맑은 고딕'으로 바꾸고 싶습니다.

SLD Cookbook으로 다시 가서 Polygon 스타일의 좀 더 아래 부분을 보니 Polygon with styled label 부분에 그나마 마음에 드는 예제가 있습니다.

이 예제의 코드 부분에서 <Font> 부분과 <LabelPlacement> 부분을 복사해 옵니다.

GeoServer 관리자 화면으로 와서 [스타일] 메뉴에서 world\_admin 스타일을 선택해 다시 편집합니다.

</Halo> 다음에 복사해온 내용을 붙여 넣고, font-family 값을 '맑은 고딕'으로 바꿉니다.

font-size 값은 13으로 바꿉니다.

스타일 편집기의 XML 중 첫행에 있는 encoding 값을 CP949로 바꿉니다.

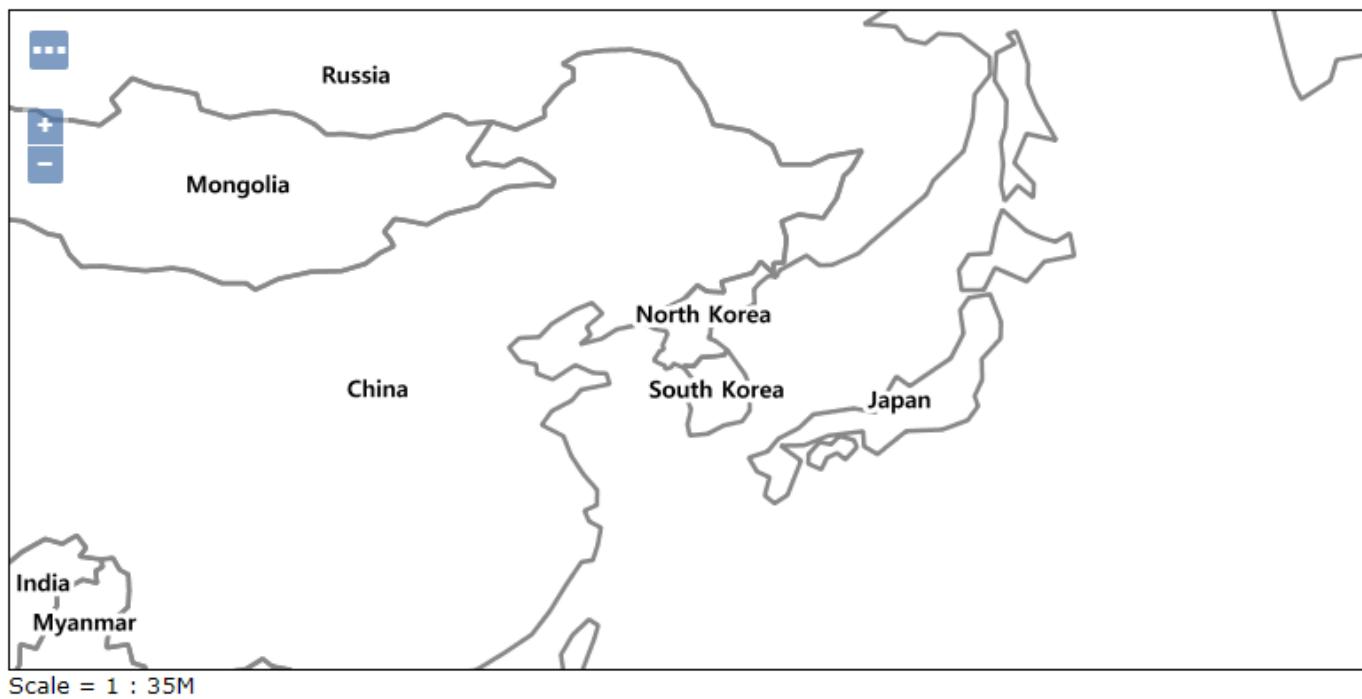
[적용하기]를 눌러 반영합니다.

이 때 encoding 값을 CP949로 바꾸는 것은 GeoServer의 버그에 대응하기 위한 일종의 꽁수입니다. 언젠가는 수정되어 다른 방법으로 대응해야 할지도 모릅니다.  
현재는 한글이 SLD에 들어갈 경우 이렇게 대응할 수 밖에 없습니다.

```
<?xml version="1.0" encoding="CP949"?>
<StyledLayerDescriptor version="1.0.0"
xsi:schemaLocation="http://www.opengis.net/sld StyledLayerDescriptor.xsd"
xmlns="http://www.opengis.net/sld"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <!-- a Named Layer is the basic building block of an SLD document -->
  <NamedLayer>
    <Name>default_line</Name>
    <UserStyle>
      <!-- Styles can have names, titles and abstracts -->
      <Title>Default Line</Title>
      <Abstract>A sample style that draws a line</Abstract>
      <!-- FeatureTypeStyles describe how to render different features -->
      <!-- A FeatureTypeStyle for rendering lines -->
      <FeatureTypeStyle>
        <Rule>
          <PolygonSymbolizer>
            <Stroke>
              <CssParameter name="stroke">#888888</CssParameter>
              <CssParameter name="stroke-width">2</CssParameter>
            </Stroke>
          </PolygonSymbolizer>
          <TextSymbolizer>
            <Label>
              <ogc:PropertyName>ADMIN</ogc:PropertyName>
            </Label>
            <Halo>
              <Radius>3</Radius>
              <Fill>
                <CssParameter name="fill">#FFFFFF</CssParameter>
              </Fill>
            </Halo>
            <Font>
              <CssParameter name="font-family">맑은 고딕</CssParameter>
              <CssParameter name="font-size">13</CssParameter>
              <CssParameter name="font-style">normal</CssParameter>
              <CssParameter name="font-weight">bold</CssParameter>
            </Font>
            <LabelPlacement>
              <PointPlacement>
                <AnchorPoint>
                  <AnchorPointX>0.5</AnchorPointX>
                  <AnchorPointY>0.5</AnchorPointY>
                </AnchorPoint>
              </PointPlacement>
            </LabelPlacement>
          </TextSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
```

```
</StyledLayerDescriptor>
```

이제 미리보기 화면에서 지도를 이동해 보면 그럭저럭 원했던 모습을 볼 수 있습니다.



지도를 아름답게 표현하는 것은 생각보다 어렵습니다.

미적 감각과 기술이 모두 필요합니다.

기술은 특히 속성을 문자로 표현하거나, 값에 따라 표현을 달리하는 등에서 많이 필요합니다.

GeoServer가 사용하고 있는 SLD(Styled Layer Descriptor)는 역시 OGC가 지정한 국제 표준이며, 기술적인 면에서 지도표현에 필요한 내용을 아주 상세히 정의할 수 있습니다.

하지만 지금 해 보셨다시피 이 복잡한 문서를 사람 손으로 수정하는 것이 결코 만만한 작업은 아닙니다. 그래도 처음부터 만들라면 못 하지만, 이미 있는 예를 보고 살짝씩 수정하는 것은 할만 합니다.

SLD에 대한 자세한 정보는 다음 링크를 참조하세요.

<http://docs.geoserver.org/stable/en/user/styling/sld/reference/index.html#sld-reference>

## 5x04 uDig을 이용한 순쉬운 스타일링

이번 시간에는 앞에서 만들었던 SLD를 좀더 손쉽게 시각적인 도구를 이용하여 만드는 방법을 익히도록 하겠습니다.

국제 표준이기에 SLD를 지원하는 툴은 많고, 우리가 흔히 사용하는 QGIS에서도 SLD를 읽고 만들 수 있습니다.

하지만 아쉽게도 QGIS에서 만든 SLD는 GeoServer에서는 잘 적용되지 않습니다.

이것은 QGIS나 GeoServer가 표준을 지키지 않아 생기는 것이 아니라, 똑같은 웹페이지도 웹브라우저에 따라 조금씩 달리 보이는 것 처럼 각 프로그램에 어느정도 구현의 자유가 있기 때문입니다.

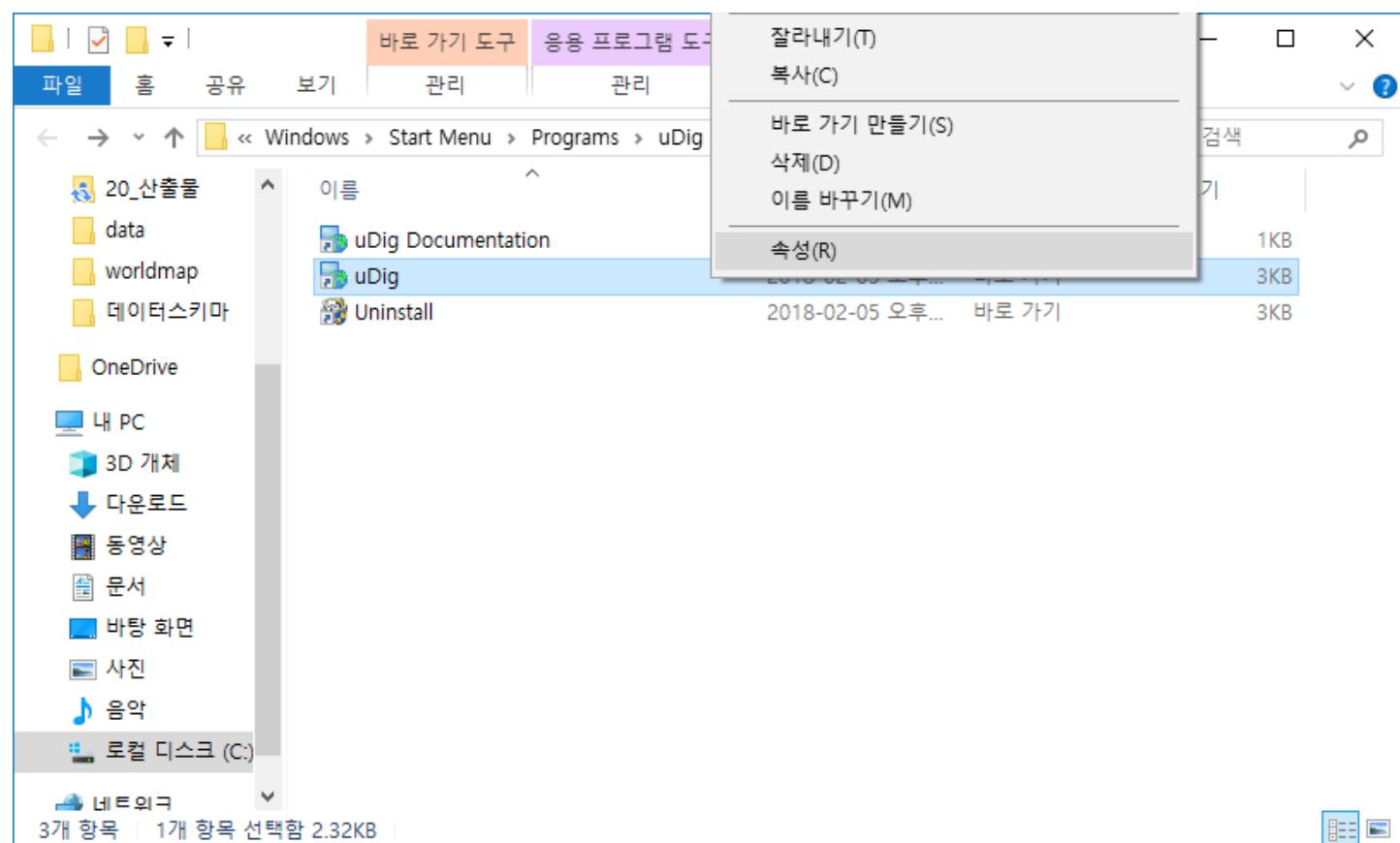
그래서 GeoServer를 위한 SLD를 가장 잘 다루는 툴은, GeoTools를 핵심 엔진으로 사용하는 uDig 입니다.

uDig은 좋은 데스크탑 GIS 프로그램입니다. 하지만, 다른 유명 오픈소스 GIS 프로그램들에 비하면 약간 덜 다듬어져 보입니다.

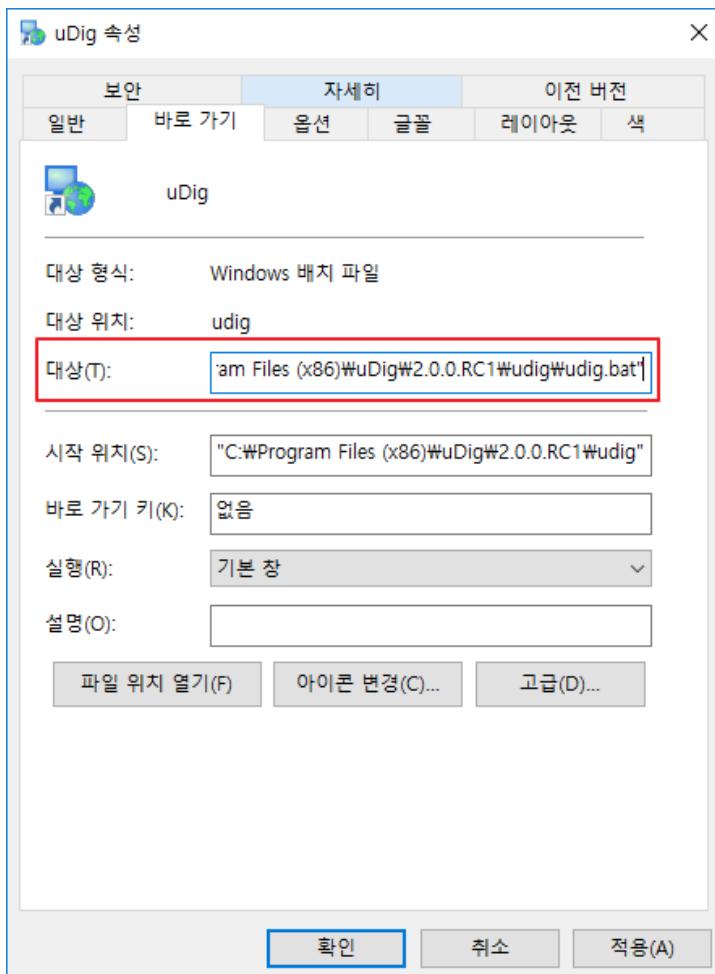
다른 툴들에 비해 속도가 좀 늦은 편이고, 특히 프로그램의 시작이 잘 되지 않는다는 문제가 있습니다.

만약 설치 후 아이콘을 찾아 클릭해도 시작이 안 된다면 다음과 같이 하시면 됩니다.

윈도우 [시작] 메뉴에서 uDig 아이콘을 찾아 파일 위치로 가기 해서 아이콘이 설치된 폴더로 갑니다.

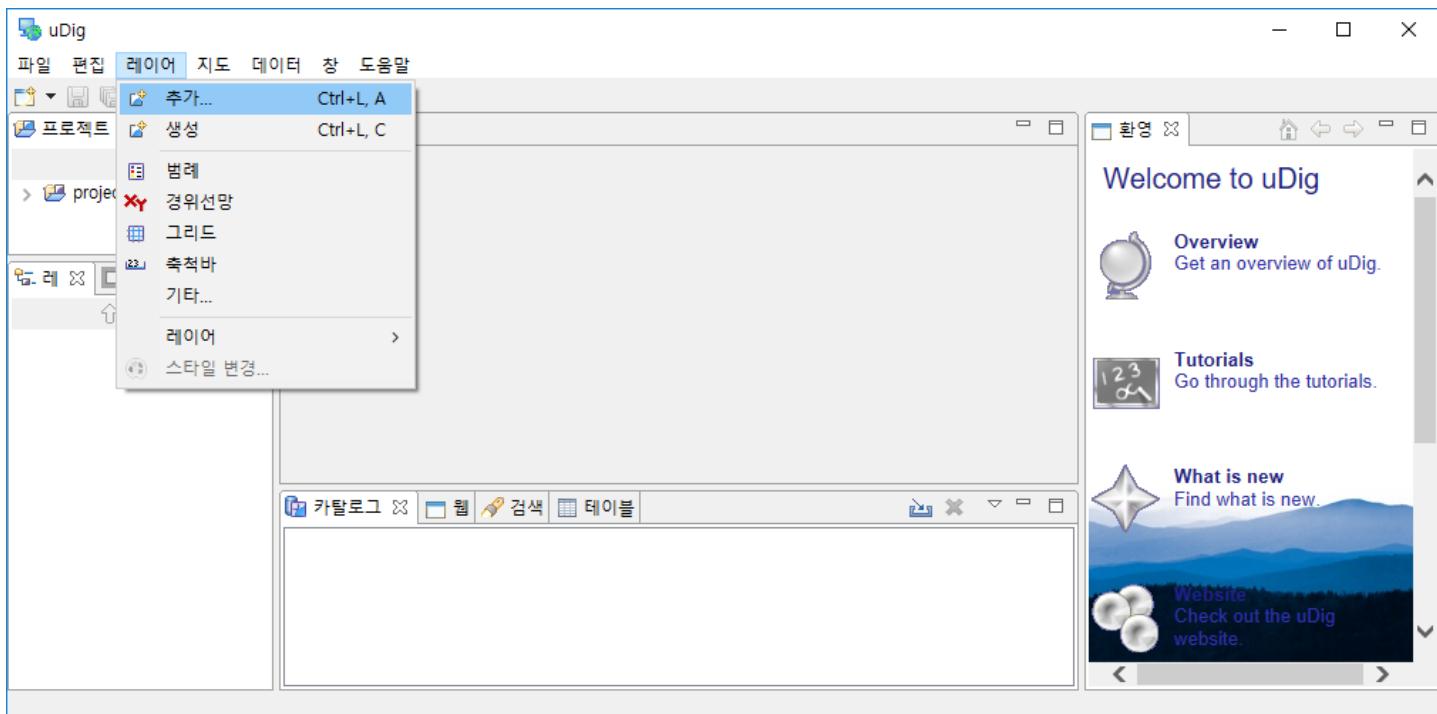


uDig 아이콘을 오른쪽 클릭하고 [속성] 메뉴를 선택합니다.

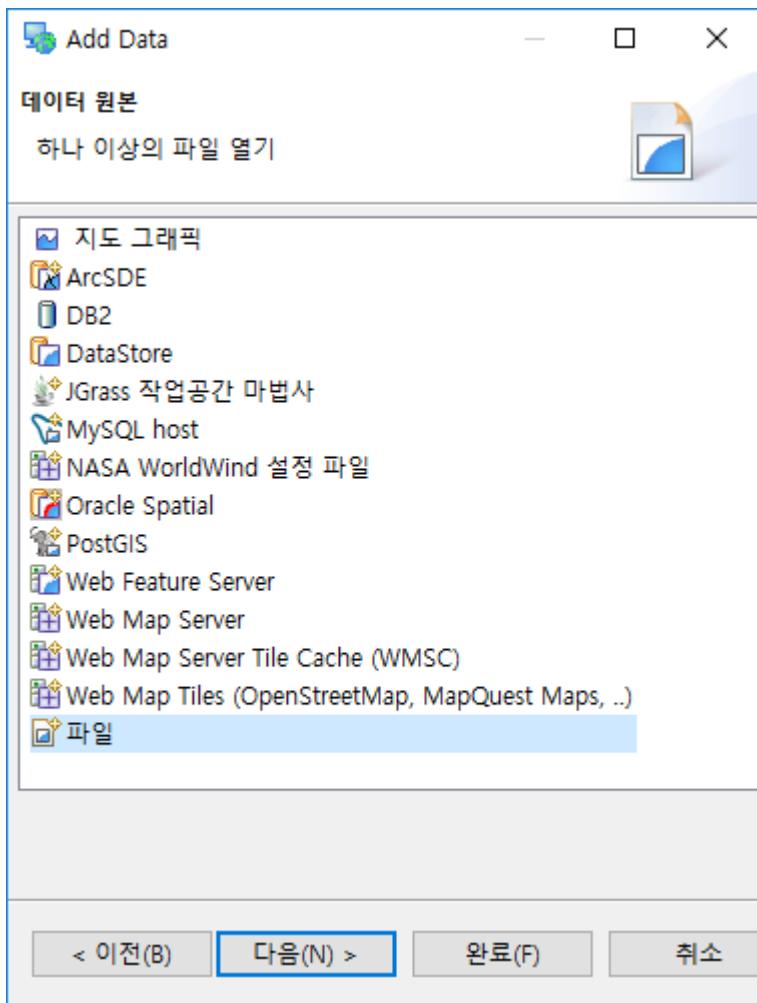


대상 항목에서 "C:\Program Files (x86)\uDig\2.0.0.RC1\udig\udig.bat" 다음에 있는 내용들을 다 지우고 [확인]을 눌러 저장합니다.

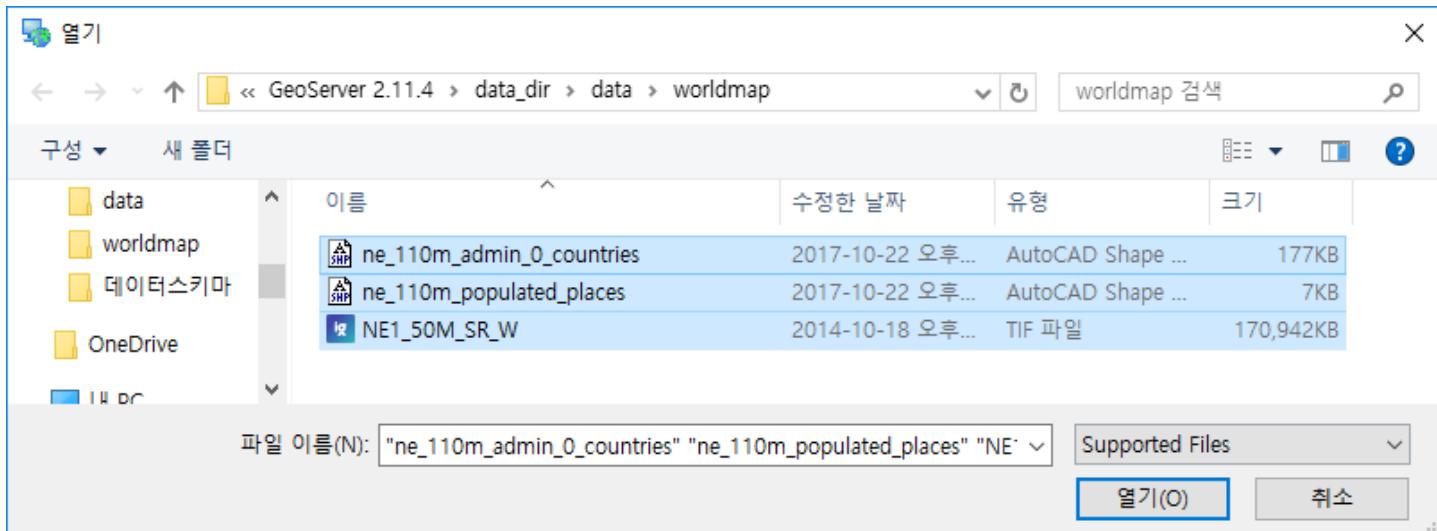
이제 다시 uDig 아이콘을 더블 클릭하면 시작이 됩니다.  
시작 하는 것이 조금 오래 걸리니 기다려 주세요.



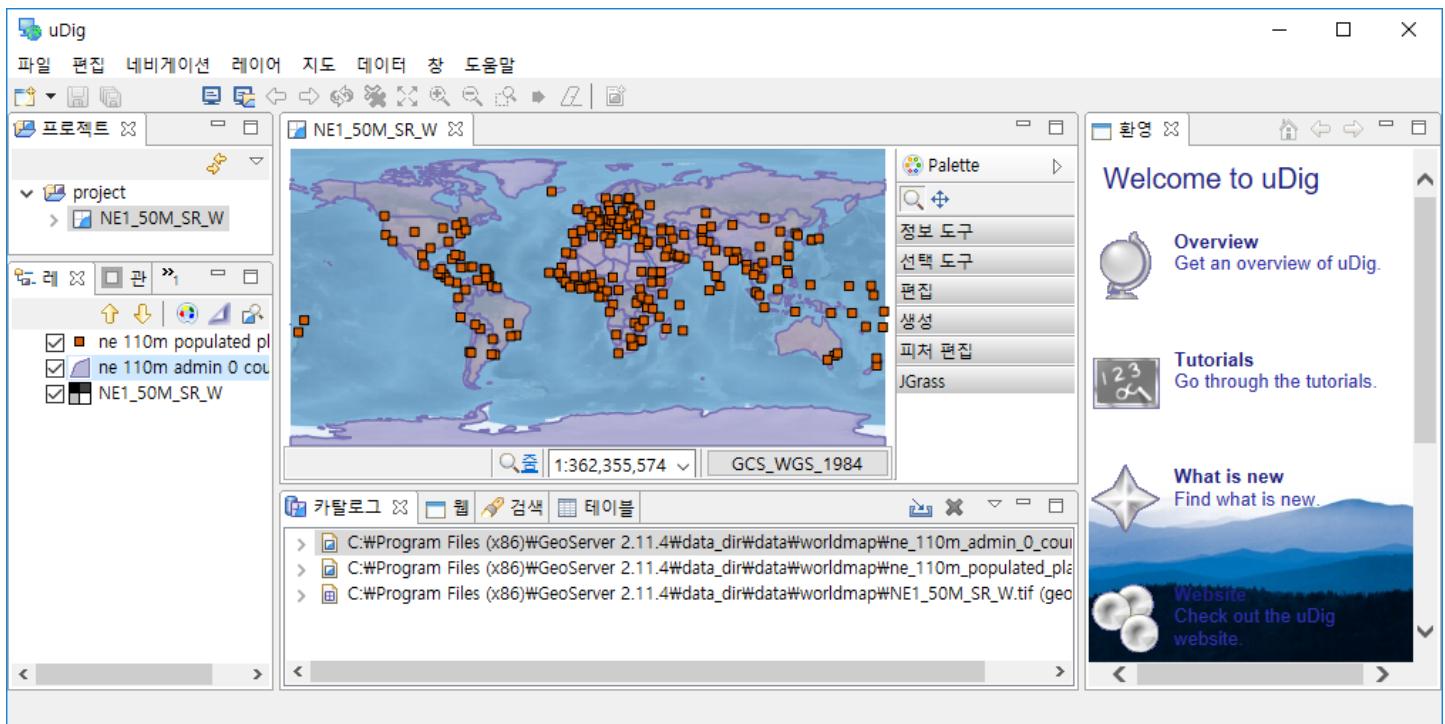
시작이 되면 [ 레이어 - 추가... ] 메뉴를 선택합니다.



데이터 원본 중 파일 항목을 선택하고 [다음>] 버튼을 누릅니다.



GeoServer의 data\_dir 폴더로 가서 data/worldmap 폴더 안의 3개 자료를 모두 선택하고 [열기] 버튼을 누릅니다.



레이어 창에서 포인트, 폴리곤, 영상 순서가 되게 조정해 줍니다.

먼저 지난 시간에 했던 국가경계 폴리곤의 심볼을 만드는 것부터 해 보겠습니다.  
레이어 리스트에서 폴리곤 레이어를 선택하고 [스타일 변경...] 메뉴를 누릅니다.

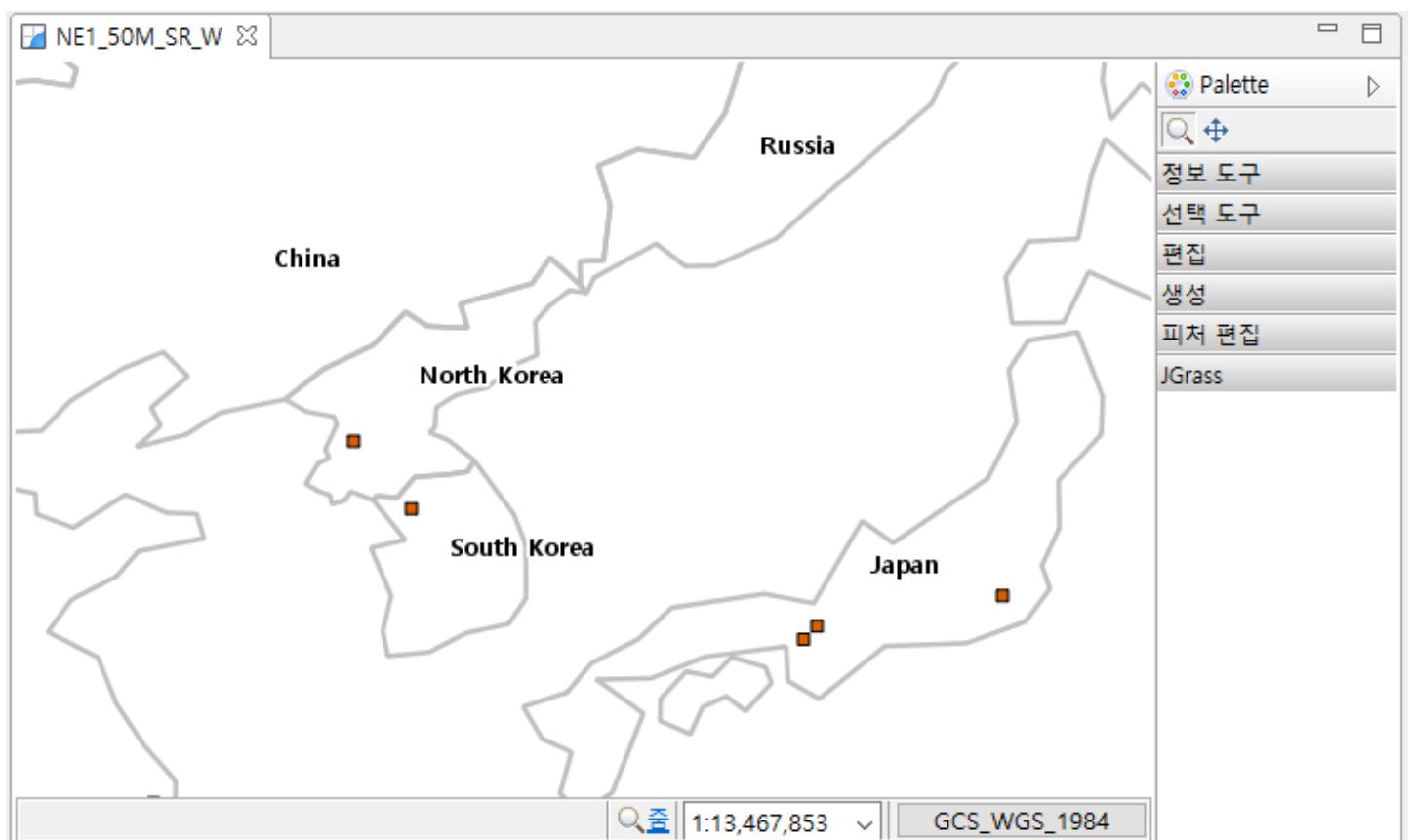
채우기를 비활성화 합니다.

테두리의 색상을 적절한 회색으로 바꾸고 두께도 2로 바꿉니다.

레이블을 활성화하고, 레이블로 ADMIN 컬럼을 선택합니다.

글꼴을 '맑은 고딕' 13 폰트로 지정하고, 헤일로는 흰색의 2 포인트로 지정합니다.

[적용] 버튼을 눌러 지도에 변경한 심볼을 반영합니다.



대략 다음과 같이 보이면 됩니다.

변경된 심볼이 마음에 들면 스타일 편집기 창에서 [내보내기] 버튼을 눌러 SLD 파일로 저장하고, 이를 GeoServer에서 불러다 쓰면 됩니다.

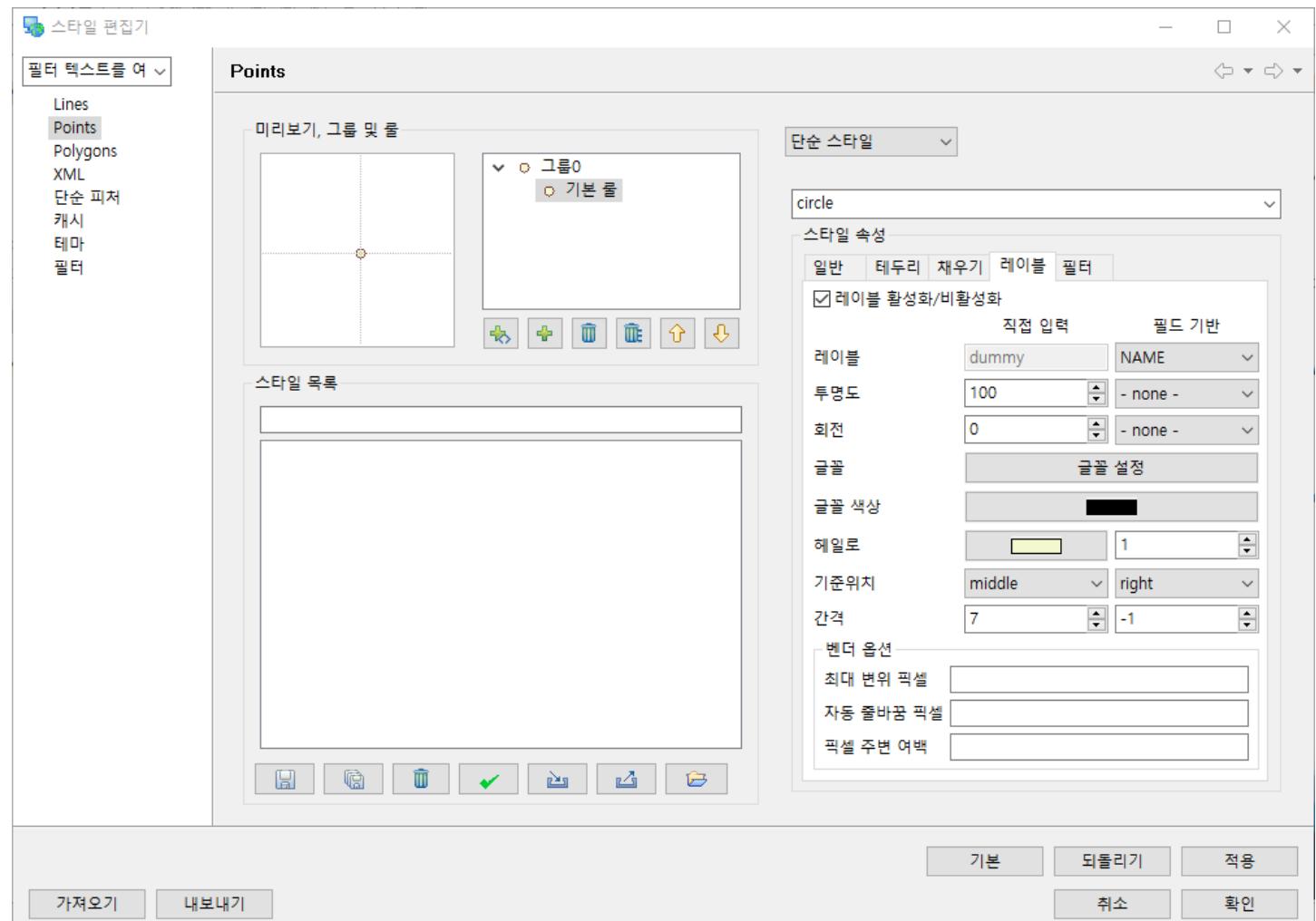
이전 시간에 에디터에서 한 작업과 유사한 스타일 정의 작업을 참고자료를 찾는 등 없이 손쉽게 해 냈습니다. 하지만, uDig에는 폴리곤의 라벨을 중앙에 가게 하는 옵션은 없네요.

이제 주요도시 포인트를 꾸며 보겠습니다.

포인트 레이어를 선택하고 [스타일 변경...]을 선택합니다.

포인트의 형태를 Circle, 크기 7, 선색으로 갈색 계열 색, 채움색에 밝은 노란색선택합니다.

레이블에 NAME 선택하고, 맑은 고딕 10포인트, 헤일로 밝은 노랑 두께 1 기준위치 middle, right 선택합니다. [적용]을 누르고 지도를 보면서 원하는 모습이 나올 때까지 조정합니다.



심볼을 다 조정했으면, [내보내기] 버튼을 눌러 world\_city.sld로 c:\data 폴더에 저장합니다.

이제 다시 GeoServer 관리자 화면으로 갑니다.

<http://localhost:8080/geoserver>

[스타일] 메뉴를 선택하고, [새로운 스타일 추가하기] 버튼을 누릅니다.

스타일 파일을 업로드 합니다. 아래의 [파일 선택] 버튼을 누르고, 조금 전에 저장한 world\_city.sld파일을 선택합니다.

[업로드...] 버튼을 누르면 SLD 파일이 읽혀져 스타일 편집기에 들어옵니다.

## 스타일 편집기

```

1 <?xml version="1.0" encoding="UTF-8"?><sld:StyledLayerDescriptor xmlns:sld="http://www.opengis.net/sld" xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://www.opengis.net/ogc" version="1.0.0">
2   <sld:UserLayer>
3     <sld:LayerFeatureConstraints>
4       <sld:FeatureTypeConstraint/>
5     </sld:LayerFeatureConstraints>
6   <sld:UserStyle>
7     <sld:Name>ne_110m_populated_places</sld:Name>
8     <sld:FeatureTypeStyle>
9       <sld:Name>点</sld:Name>
10      <sld:FeatureTypeName>Feature</sld:FeatureTypeName>
11      <sld:SemanticTypeIdentifier>generic:geometry</sld:SemanticTypeIdentifier>
12      <sld:SemanticTypeIdentifier>simple</sld:SemanticTypeIdentifier>
13    <sld:Rule>
14      <sld:Name>点</sld:Name>
15      <sld:PointSymbolizer>
16        <sld:Graphic>
17          <sld:Mark>
18            <sld:WellKnownName>circle</sld:WellKnownName>
19            <sld:Fill>
20              <sld:CssParameter name="fill">#F4F8CF</sld:CssParameter>
21            <sld:Stroke>
22              <sld:CssParameter name="stroke">#804040</sld:CssParameter>
23

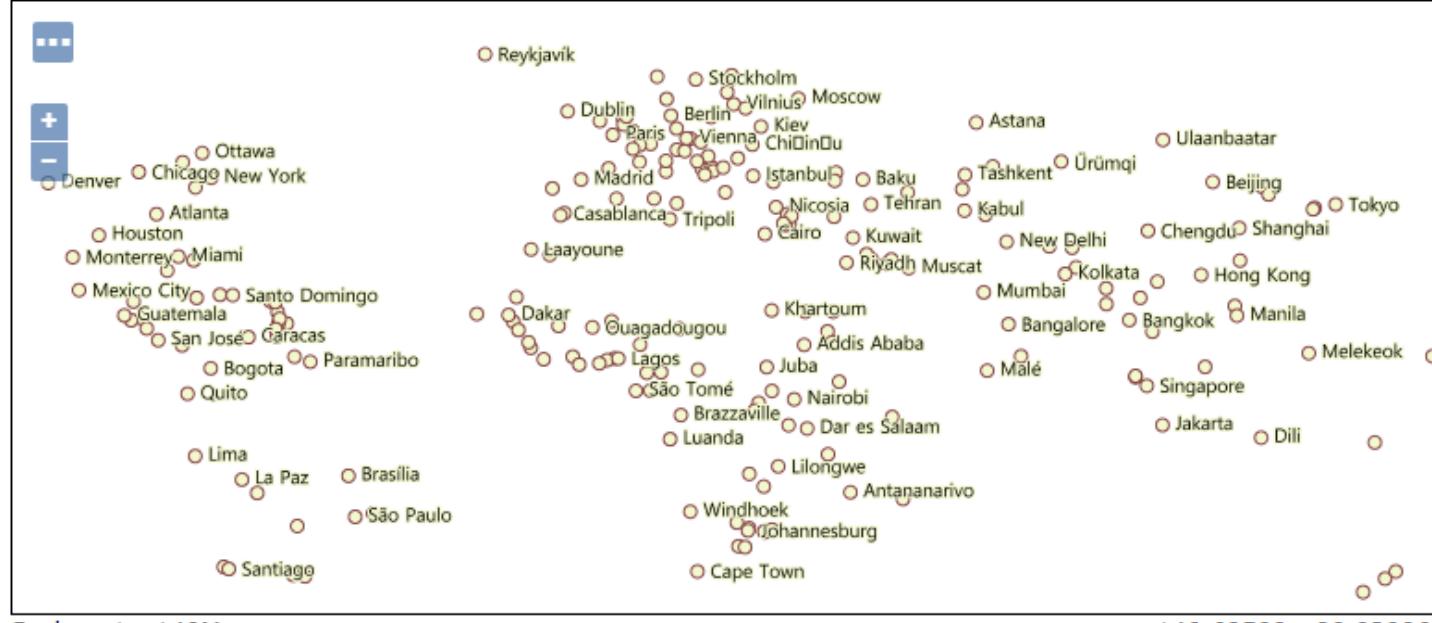
```

그런데 좀 문제가 있습니다. 한글이 다 깨져 보이네요.

첫 줄의 encoding을 "CP949"로 바꾸고 Name 부분의 한글이 깨진 곳을 world\_city로 바꿉니다.  
font-family 값을 '맑은 고딕'으로 바꾸고 [제출] 을 눌러 저장합니다.

[레이어] 메뉴로 가서 ne\_110m\_populated\_places 레이어를 선택합니다.  
[발행] 탭을 선택하고 기본 스타일을 world\_city로 바꾸고 [저장] 합니다.

[레이어 미리보기]로 가서 ne\_110m\_populated\_places 레이어를 확인합니다.

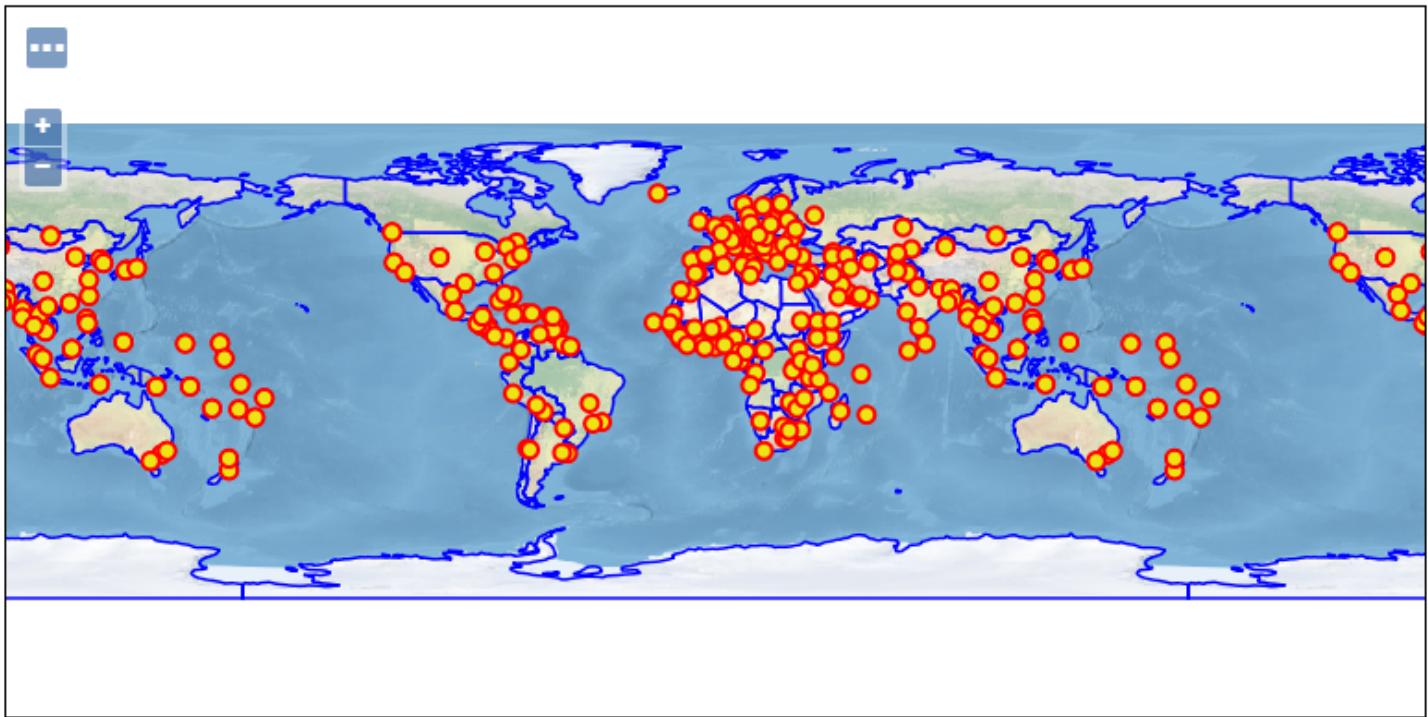


Scale = 1 : 140M

[Click on the map to get feature info](#)

140.62500, -20.03906

이제 [레이어 미리보기]로 가서 worldmap 레이어 그룹을 확인해 보면 조금 이상한 현상이 보입니다.



Scale = 1 : 279M

*Click on the map to get feature info*

열심히 심볼을 빼줬는데 아직 모든 레이어가 그대로네요.

이는 그룹 레이어인 경우 처음에 그룹 레이어를 만들 때 지정된 기본 스타일을 기억하고 있기 때문입니다.  
이를 바꿔주어야 합니다.

[레이어 그룹] 메뉴를 선택하고, worldmap 레이어 그룹을 선택합니다.

worldmap:ne\_110m\_admin\_0\_countries 레이어의 스타일로 world\_admin,

worldmap:ne\_110m\_populated\_places 레이어의 스타일로 world\_city를 선택하고 [저장]을 누릅니다.

#### 레이어 목록



레이어 추가하기...

레이어 그룹 추가하기...

그리기 순서	레이어	기본 스타일	스타일	제거
1	worldmap:NE1_50M_SR_W	<input type="checkbox"/>	raster	
2	worldmap:ne_110m_admin_0_countries	<input type="checkbox"/>	world_admin	
3	worldmap:ne_110m_populated_places	<input type="checkbox"/>	world_city	

1 결과: 1에서 3 ( 3 항목 중)

아까 띄워 놓은 worldmap 레이어 그룹의 미리보기 화면에 가서 화면을 이동하면 이제 심볼이 바뀌어 보입니다.



Scale = 1 : 140M

[Click on the map to get feature info](#)

조금 더 조정이 필요해 보이는 것은 도시들이 표시되는 것 때문에 나라 이름들이 몇개 안 보이네요.  
그럼 도시 이름을 이정도 축척 까지만 나오도록 조정해 보겠습니다.



Scale = 1 : 17M

[Click on the map to get feature info](#)

저는 이 정도 확대된 정도까지만 도시들이 표시 되었으면 합니다.

이 때 중요한 정보가 미리보기 화면 좌하단에 있습니다.

현재 축척이 1 : 17M, 즉 1 : 17,000,000 이네요.

저는 1 : 20,000,000 부터는 도시들이 사라지게 설정하겠습니다.

[스타일] 메뉴로 가, world\_city 스타일을 선택하고, 스타일 편집기에서 <sld:PointSymbolizer>가 시작하는 부분을 찾아 그 앞에 다음 행을 넣어 줍니다.

```
<sld:MaxScaleDenominator>20000000</sld:MaxScaleDenominator>
```

그리고 [제출] 하시면 Point 들이 보이게 될 최소 축척의 지정이 끝납니다.

### 스타일 편집기



```
1 <?xml version="1.0" encoding="CP949"?><sld:StyledLayerDescriptor xmlns="http://www.opengis.net/sld" xmlns:sld="http://www.opengis.net/sld" xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://www.opengis.net/ogc" version="1.0.0">
  2   <sld:UserLayer>
  3     <sld:LayerFeatureConstraints>
  4       <sld:FeatureTypeConstraint/>
  5     </sld:LayerFeatureConstraints>
  6   <sld:UserStyle>
  7     <sld:Name>ne_110m_populated_places</sld:Name>
  8     <sld:FeatureTypeStyle>
  9       <sld:Name>world_city</sld:Name>
10       <sld:FeatureTypeName>Feature</sld:FeatureTypeName>
11       <sld:SemanticTypeIdentifier>generic</sld:SemanticTypeIdentifier>
12       <sld:SemanticTypeIdentifier>simple</sld:SemanticTypeIdentifier>
13       <sld:Rule>
14         <sld:Name>world_city</sld:Name>
15         <sld:MaxScaleDenominator>20000000</sld:MaxScaleDenominator>
16       <sld:PointSymbolizer>
17         <sld:Graphic>
18           <sld:Mark>
19             <sld:WellKnownName>circle</sld:WellKnownName>
20             <sld:Fill>
21               <sld:CssParameter name="fill">#F4F8CF</sld:CssParameter>
22             </sld:Fill>
23             <sld:Stroke>
```

이렇게 GeoServer는 각 레이어가 언제 보이고 사라질지를 스타일에서 지정해야 합니다.

많은 GIS 툴들이 레이어에서 가시성 설정을 하는 것과 비교해 보면 좀 특이해 보이기는 합니다.

그리고 첫 화면이 절대 바뀌지 않은 캐시의 강력함은 너무 원망말아 주세요. 이 강력함이 곧 도움이 될 것입니다.



Scale = 1 : 140M  
Click on the map to get feature info



Scale = 1 : 17M  
Click on the map to get feature info

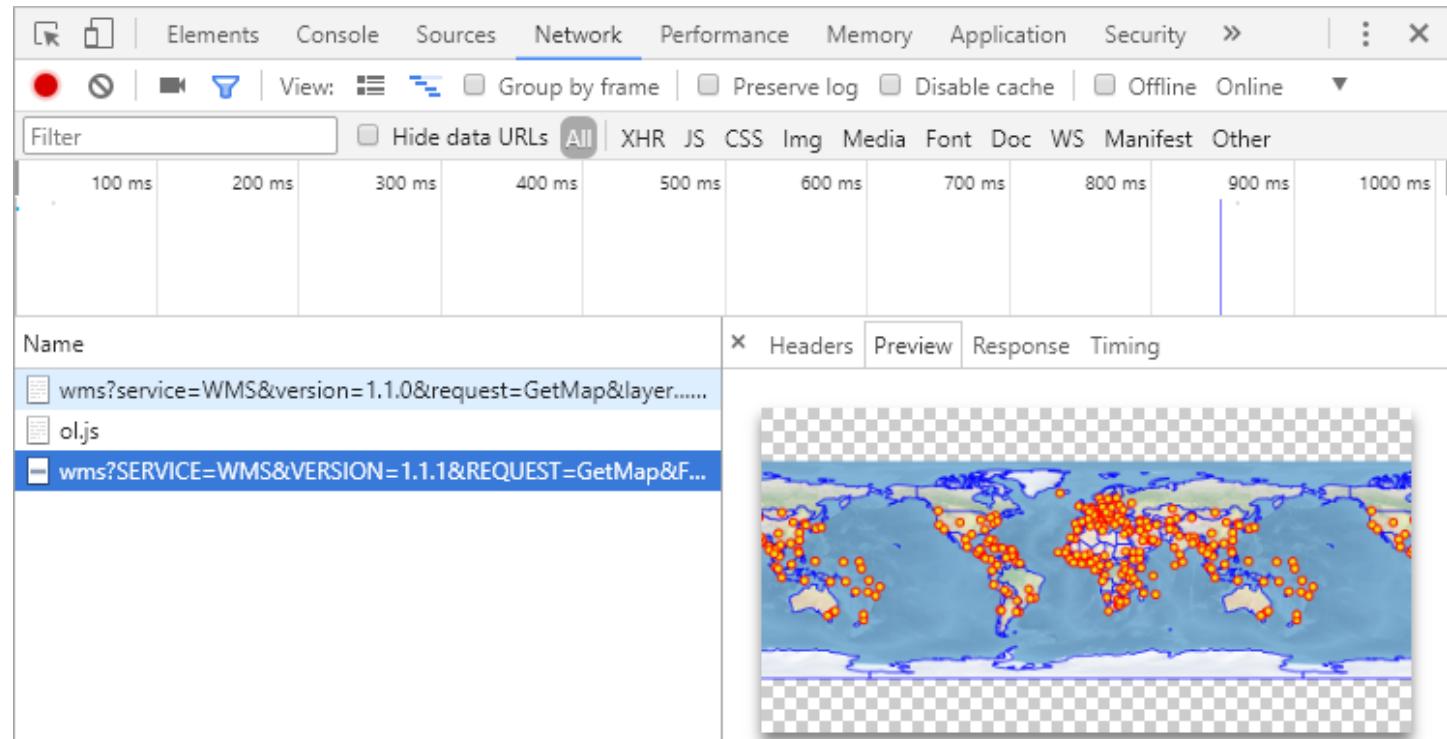
## 5x05 GWC를 이용한 캐시

우리가 지금까지 무심코 사용했던 GeoServer가 제공하는 레이어 미리보기 기능에는 생각보다 많은 기능이 들어있습니다.

화면의 크기를 바꿀 수도 있고, 받아올 이미지의 포맷도 변경할 수 있고, WMS 버전 등의 상세한 정보도 바꿔 줄 수 있으며, 심지어 받아올 객체의 조건을 주는 것 까지도 가능합니다.

이번에 배워볼 캐시와 관련된 중요한 기능인 타일 형태로 지도를 불러오는 방법도 여기서 지정 가능합니다.

현재 미리보기 화면을 크롬의 개발자 도구에서 네트워크 분석 해 보면 지도 전체를 한 장의 이미지로 받아오고 있는 것을 확인할 수 있습니다.



이렇게 웹페이지의 지도 부분에 들어갈 이미지를 한 장으로 받아오는 것은 캐시라는 측면에서 불리한 면이 있습니다,

보통 지도화면은 웹 페이지의 크기 변화에 따라 자연스럽게 변화하고, 그러면 지도화면에 가득 차는 지도 이미지의 크기도 바뀌게 됩니다.

이렇게 크기가 바뀌면 웹 브라우저 입장에서는 똑같은 지도 이미지가 아니라 다른 컨텐츠이기에 서버에 다시 요청해야 하고, 그러면 서버인 GeoServer는 거의 같은 그림을 다시 그려야만 합니다.

레이어 미리보기 화면의 오른쪽 상단에 있는 [...] 버튼을 선택하면 여러가지 옵션을 지정할 수 있도록 확장되는데 이 중 Width/Height를 600/600으로 바꿔 봅시다.

WMS version: 1.1.1 Tiling: Single tile Antialias: Full Format: PNG 24bit  
Styles: Default Width/Height: 600 600  
Filter: CQL Apply  
Reset

Name

- wms?service=WMS&version=1.1.0&request=GetMap&layer.....
- ol.js
- wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&F...
- wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&F...

Headers Preview Response

100000 ms 200000 ms 300000 ms 400000 ms 500000 ms

4 requests | 227 KB transferred | Finish: 7.4 min | DOMContent... 222 KB | 600 × 600 | image/png

이제야 우리를 괴롭히던 강력한 캐시 효과에서 벗어나 드디어 지도가 바꾼 스타일로 변경되었네요. 지도 이미지는 여전히 하나의 통 이미지로 오고 있습니다.

이제 Tiling 항목을 Single Tile에서 Tiled로 바꿔 봅시다.

WMS version: 1.1.1 Tiling: Tiled Antialias: Full Format: PNG 24bit  
Styles: Default Width/Height: 600 600  
Filter: CQL Apply  
Reset

Name

- wms?service=WMS&version=1.1.0&request=GetMap&layer.....
- ol.js
- wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&F...
- wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&F...
- wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&F...
- wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&F...
- wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&F...
- wms?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&F...

Headers Preview Response Timing

100000 ms 200000 ms 300000 ms 400000 ms 500000 ms 600000 ms 700000 ms 800000 ms

8 requests | 227 KB transferred | Finish: 11.8 min | DOMContent... 53.7 KB | 256 × 256 | image/png

지도 화면에 변화가 없는 것처럼 보이지만, 네트워크 분석을 보면 4장의 256\*256 크기의 타일로 분할되어 지도 이미지가 움직일 수 있습니다.

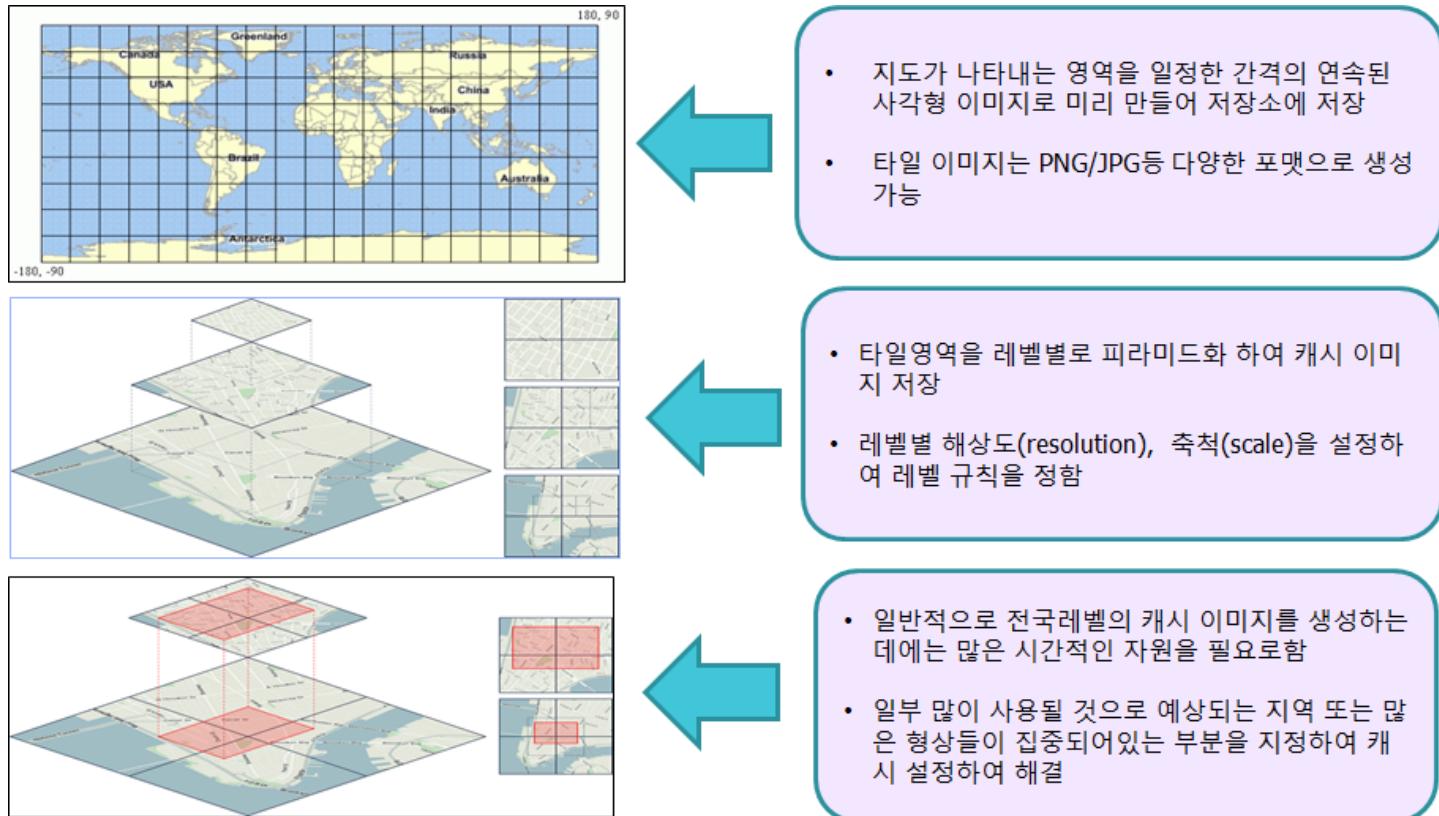
심지어 화면 크기가 600\*600인데 256\*256크기의 타일 4장만 왔습니다.

그럼 좀 비는 나머지 이미지 부분은 어떻게 처리된 것일까요?

비밀은 태평양에 있는 날짜변경선에서 나뉘어진 타일이 오른쪽 왼쪽으로 재사용 된 것입니다.

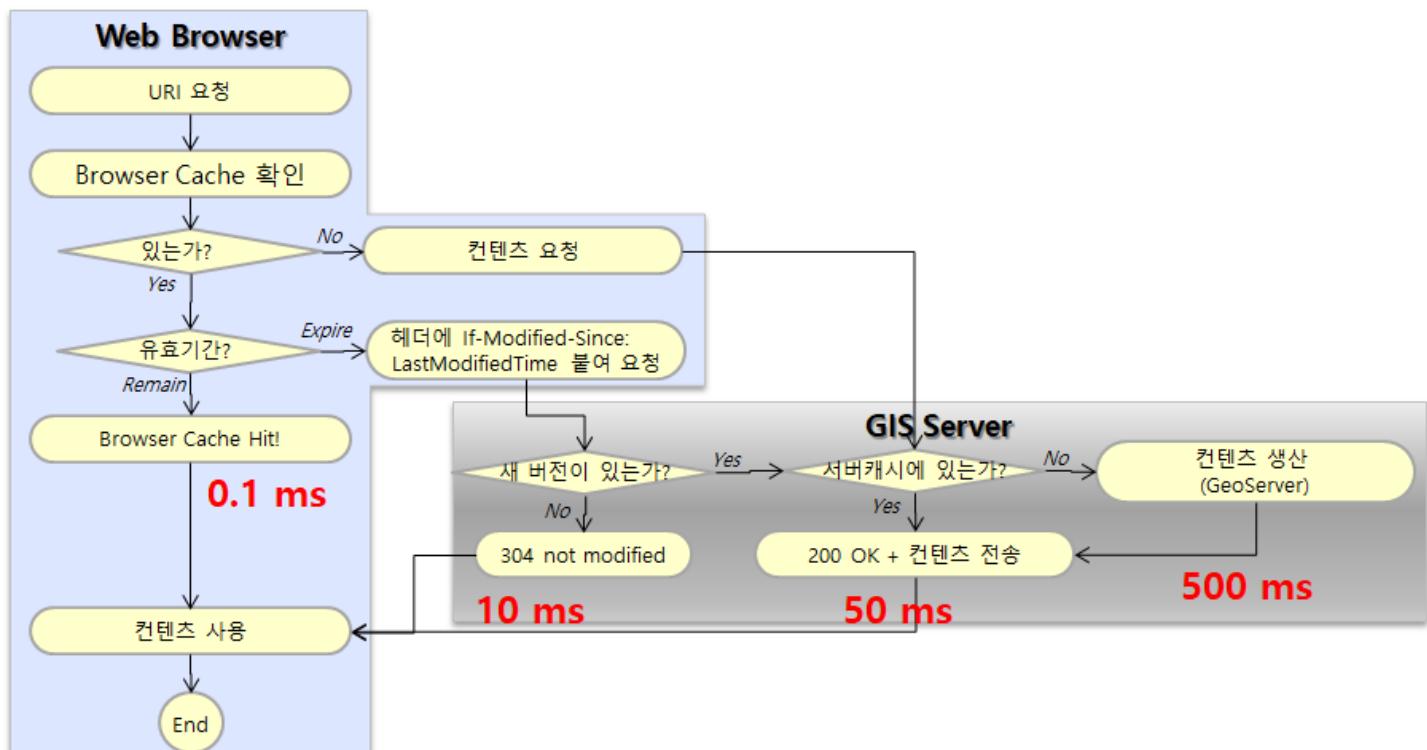
지도를 좀 더 확대해 보면 더 많은 타일들이 오는 것을 확인하실 수 있습니다.

그런데 중요한 것은 이런 타일들이 잘리는 자리를 항상 같다는 것입니다.  
때문에 이 타일들을 캐시해서 재활용 하면, 웹페이지의 크기가 바뀌어 지도 크기가 바뀌어도 캐시된 자료를 그냥 사용할 수 있어 매우 효율적입니다.



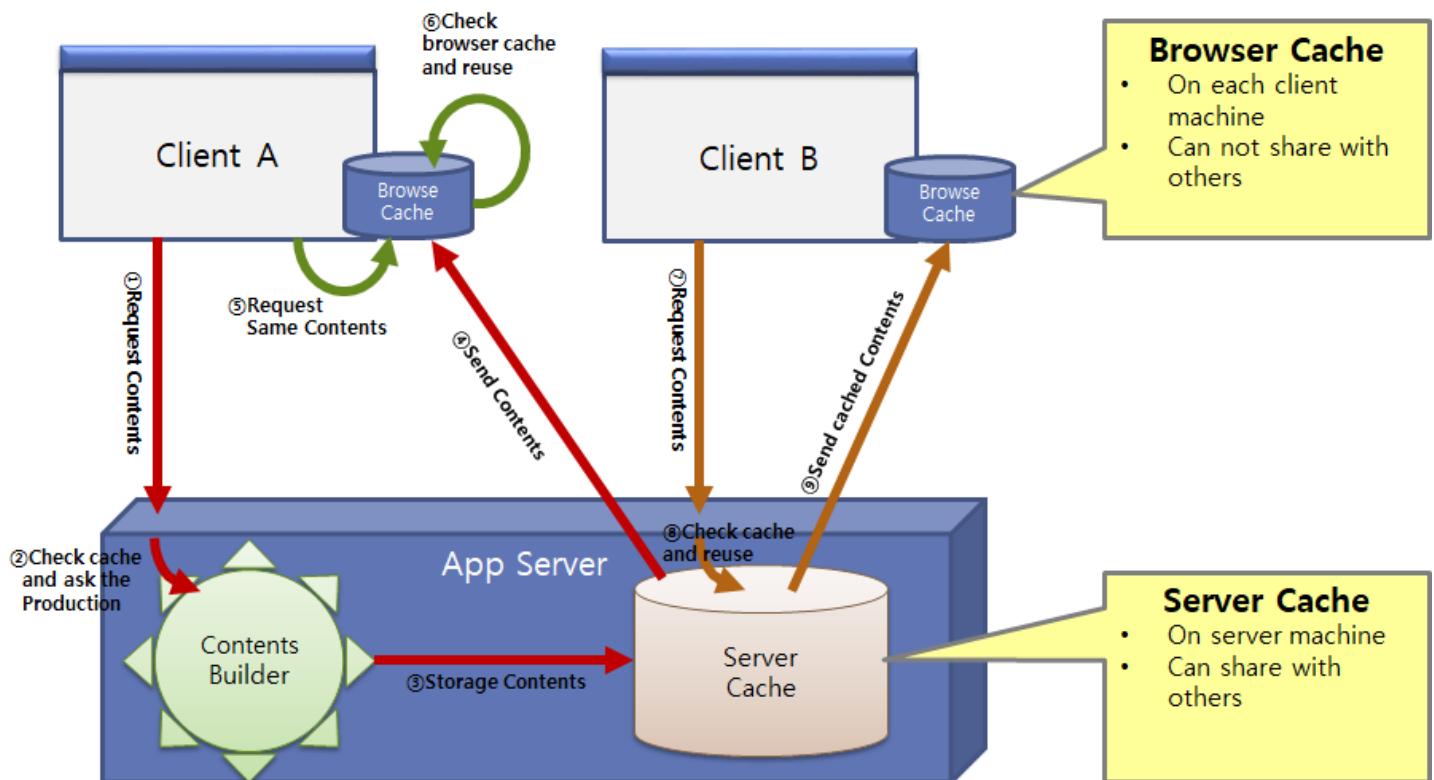
이렇게 타일들이 잘리는 자리를 전 지구적으로 명확히 정의해 놓은 것을 그리드셋이라고 합니다.

이런 지도 타일을 이용한 캐시 기법은 단지 나의 웹브라우저의 캐시에만 효율적인 것이 아니라, 서버 차원에서도 누가 요청하던 미리 만들어 둔 지도 타일이 있는 경우 다시 만들지 않고 이를 서비스에 사용할 수 있다는 커다란 장점이 있습니다.



지도 타일 같은 웹 컨텐츠의 캐시 방법은 HTTP가 설계 될 때 이미 잘 설계되어 지금도 이 규칙을 그대로 사용 중입니다.

이 규칙에 따라 각 클라이언트와 서버의 캐시가 어찌 생산되고 활용될지가 결정되고, 이에 따라 엄청난 속도 차이를 보이게 됩니다.



간혹 왜 브라우저 캐시가 있는데 불필요하게 서버 캐시를 또 만드냐고 하는 분이 있는데, 이는 내 동료의 브라우저에 있는 캐시를 내가 쓸 수 있다면, 보안적으로 문제가 되기 때문이라 할 수 있습니다.  
하지만, 동료를 주기 위해 만들었던 캐시를 내가 서버에서 다시 받아 사용하는 것은 아무런 문제도 되지 않습니다.

단지 서버가 두번 일하지 않게 해 줄 수 있을 뿐이지요. 가장 안정적이고 좋은 서버는, 거의 일을 하지 않는 서버입니다. 이런 철학으로 만든 서비스는 아주 훌륭한 속도와 성능을 보입니다.

이제 GeoServer 관리화면에서 캐시를 관리하는 방법을 배워 보겠습니다.

메뉴의 아랫쪽을 보면 타일캐시 그룹이 있습니다.

GeoServer의 타일캐시 그룹에 있는 설정들은 모두 서버캐시에 관한 설정입니다.

브라우저 캐시를 위한 설정은 이미 우리가 레이어를 만들때 배웠었습니다.

캐시 헤더 설정 부분이 바로 그것입니다.

GeoServer에서 지원하는 지도타일 서버캐시를 GeoWebCache 줄여서 GWC라 많이 부릅니다.

서버캐시 설정을 위해 먼저 [캐시 기본 설정] 메뉴를 먼저 눌러보겠습니다.

이 중에서 꼭 살펴봐야 할 부분이 '기본 타일 이미지 포맷' 부분과 그리드셋 부분입니다.

### 기본 타일 이미지 포맷:

#### 벡터 레이어

- application/json;type=utfgrid
- image/gif
- image/jpeg
- image/png
- image/png8
- image/vnd.jpeg-png

#### 래스터 레이어

- image/gif
- image/jpeg
- image/png
- image/png8
- image/vnd.jpeg-png

#### 레이어 그룹

- application/json;type=utfgrid
- image/gif
- image/jpeg
- image/png
- image/png8
- image/vnd.jpeg-png

### 인 메모리 Blob 저장소 옵션

#### 활성화



### 기본 캐시된 그리드셋

그리드셋	CRS	타일 Dimensions	줌 레벨	사용량	삭제
EPSG:4326	EPSG:4326	256 x 256	22	0.0B	
EPSG:900913	EPSG:900913	256 x 256	31	0.0B	
기본 그리드셋 추가 <input type="button" value="선택"/>					

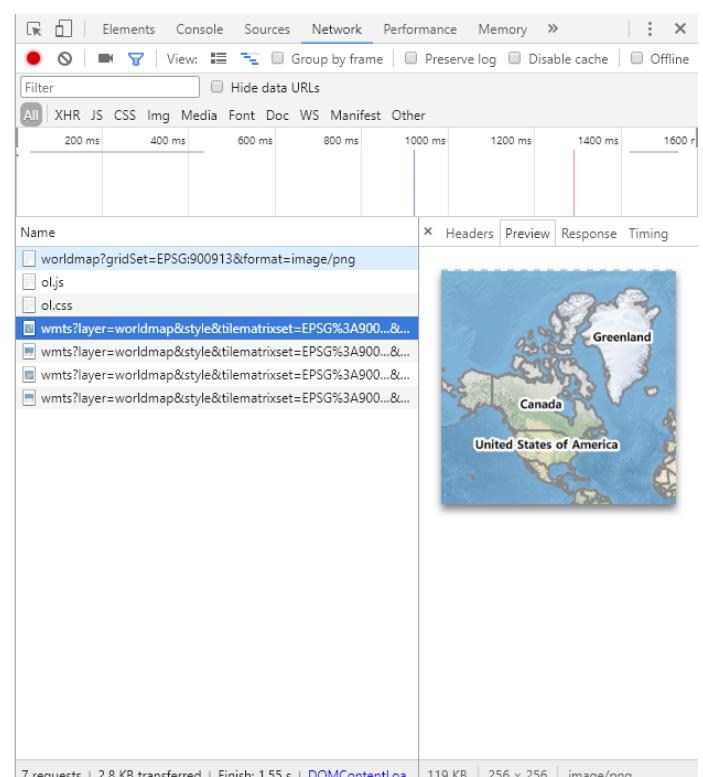
기본적으로 jpg와 png 포맷의 이미지만 캐시가 되도록 되어 있습니다. 이것은 gif, tif 등으로 요청시 서버캐시를 타지 않는다는 의미입니다.

가장 아래 부분을 보면 '기본 캐시된 그리드셋'이라는 부분이 있습니다.

아까 말했던 타일이 잘리는 자리인 그리드셋이 바로 이것입니다.

그리드셋은 보통 좌표계마다 정의되고 기본적으로 EPSG:4326(경위도)와 EPSG:900913(구글좌표계 혹은 메르카토르 좌표계, 정식명칭 EPSG:3857)좌표계에 대한 그리드셋이 정의되어 있습니다.

때문에 기본적으로는 경위도와 구글좌표계로 요청하는 타일들만 서버캐시가 생성됩니다.



이런 기본 좌표계를 쓰지 않는 지도들을 타일로 캐시하려면, [그리드셋] 메뉴를 이용해 직접 정의해 줄 수 있습니다.

하지만, 이는 입문과정 수업 범위를 벗어납니다.

추가적으로 볼 메뉴는 [타일레이어] 메뉴입니다.

여기서 레이어와 좌표계를 선택해 타일형태로 요청되는 지도들을 미리보기 해볼 수도 있고, 캐시를 미리 생성하거나 지울 수 있고, 각 레이어별 캐시 할당량을 지정할 수도 있습니다.

The screenshot shows the GeoWebCache interface. At the top, there's a logo of a globe with a grid pattern and the text "GeoWebCache". Below it, there's a "List" button followed by a dropdown menu set to "this Layer tasks" with the note "(there are no tasks for other Layers)". There's also a "Kill" button with a dropdown menu set to "all", a "Tasks for Layer 'worldmap'" button, and a "Submit" button. Underneath, there's a section titled "List of currently executing tasks:" which contains a bullet point "• none". Below this is a link "Refresh list".

#### **List of currently executing tasks:**

- *none*

[Refresh list](#)

#### **Please note:**

- This minimalistic interface does not check for correctness.
- Seeding past zoomlevel 20 is usually not recommended.
- Truncating KML will also truncate all KMZ archives.
- Please check the logs of the container to look for error messages and progress indicators.

Here are the max bounds, if you do not specify bounds these will be used.

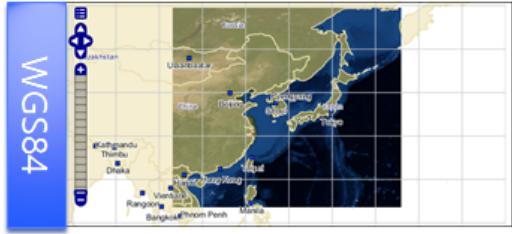
- EPSG:4326: -180.0,-90.0,180.0,90.0
- EPSG:900913: -20037508.34,-20037508.34,20037508.34,20037508.34

#### **Create a new task:**

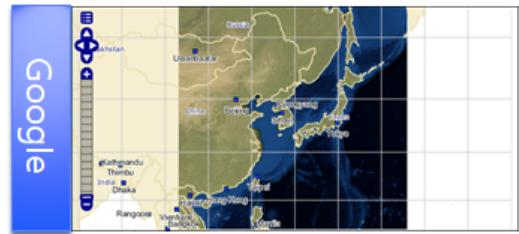
Number of tasks to use:	<input type="text" value="01"/>
Type of operation:	<input type="text" value="Seed - generate missing tiles"/>
Grid Set:	<input type="text" value="EPSG:4326"/>
Format:	<input type="text" value="image/png"/>
Zoom start:	<input type="text" value="00"/>
Zoom stop:	<input type="text" value="15"/>

보통 이 화면에서 가장 많이 하는 작업은 이용률이 많을 것으로 예상되는 레이어의 캐시를 미리 만들어 두는 작업입니다.

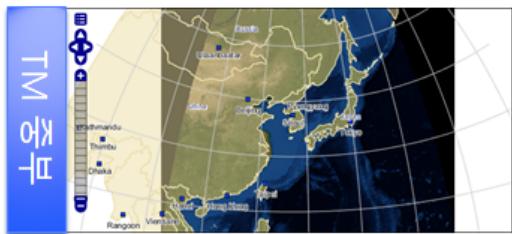
<http://localhost:8080/geoserver/wms?service=WMS&version=1.1.0&request=GetMap&layers=world&styles=&width=660&height=330&format=application/openlayers&srs=EPSG:4326&bbox=110,20,150,50>



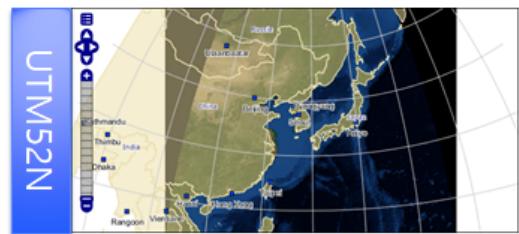
<http://localhost:8080/geoserver/wms?service=WMS&version=1.1.0&request=GetMap&layers=world&styles=&width=660&height=330&format=application/openlayers&srs=EPSG:900913&bbox=12245143.9872601,2273030.92698769,16697923.618991,6446275.84101716>



<http://localhost:8080/geoserver/wms?service=WMS&version=1.1.0&request=GetMap&layers=world&styles=&width=660&height=330&format=application/openlayers&srs=EPSG:2097&bbox=-1599161.20365349,-1401708.62964815,1839996.14666927,2091714.20629109>



<http://localhost:8080/geoserver/wms?service=WMS&version=1.1.0&request=GetMap&layers=world&styles=&width=660&height=330&format=application/openlayers&srs=EPSG:32652&bbox=-1516012.385514,2328845.36574506,1998458.72659911,5753394.63053658>



앞의 예에서도 보았듯이 GeoServer는 강력한 좌표계 변환기능이 있어서 벡터, 래스터 혹은 모두 합쳐진 지도를 실시간 좌표변환 해 줄 수 있습니다.

하지만, 실시간 좌표계 변환 기능은 매우 부하가 큰 기능입니다.

때문에 상용 서비스에서는 가능한 한 서비스할 좌표계로 모든 자료를 변환해 놓고 좌표계 변환 없이 서비스될 수 있도록 해 줍니다.

## 6x01 OpenLayers 소개

OpenLayers는 간단히 말해 웹에 지도를 붙여주는 라이브러리입니다. 흔히 웹지도 콘트롤이라고도 불립니다.

OpenLayers가 자체적으로 지도를 제공하는 것은 아니고, OSM(Open Street Map), Google Map, Bing Map, 다음지도, 네이버지도 등에서 제공하는 지도데이터를 받아 이를 웹에서 자유롭게 이동/확대 등이 가능한 동적 지도 형태로 기능을 제공합니다.

사용하는 공간 데이터는 앞에서 배운 WMS, WMTS, WFS, WCS 등 OWS 인터페이스를 제공하는 것이라면 대부분 사용할 수 있고, 이 외에도 타일 이미지 형태로 지도를 공급하는 OSM, 구글지도 등도 대부분 사용할 수 있습니다.

이런 특성을 이용해 우리가 만든 GeoServer 세계지도를 데이터로 활용하는 서비스를 만들어 볼 것입니다.

또한 단순히 지도를 보여주는 기능 외에도 여러 레이어를 관리하고, 다양한 형태로 가시화하고, 좌표계변환을 하며, 객체를 선택하고, 편집할 수 있는 등, 웹GIS라 불리는 여러가지 기능들을 제공하고 있습니다.

이런 기능들이 모두 Javascript와 CSS로 구현되어 HTML5를 지원하는 웹 브라우저에서 추가 컨트롤 없이 효율적으로 동작하게 되어 있습니다.

때문에 데스크탑 웹에서 뿐 아니라 모바일에서도 잘 동작하는 지도서비스를 쉽고 강력하게 만들 수 있습니다.

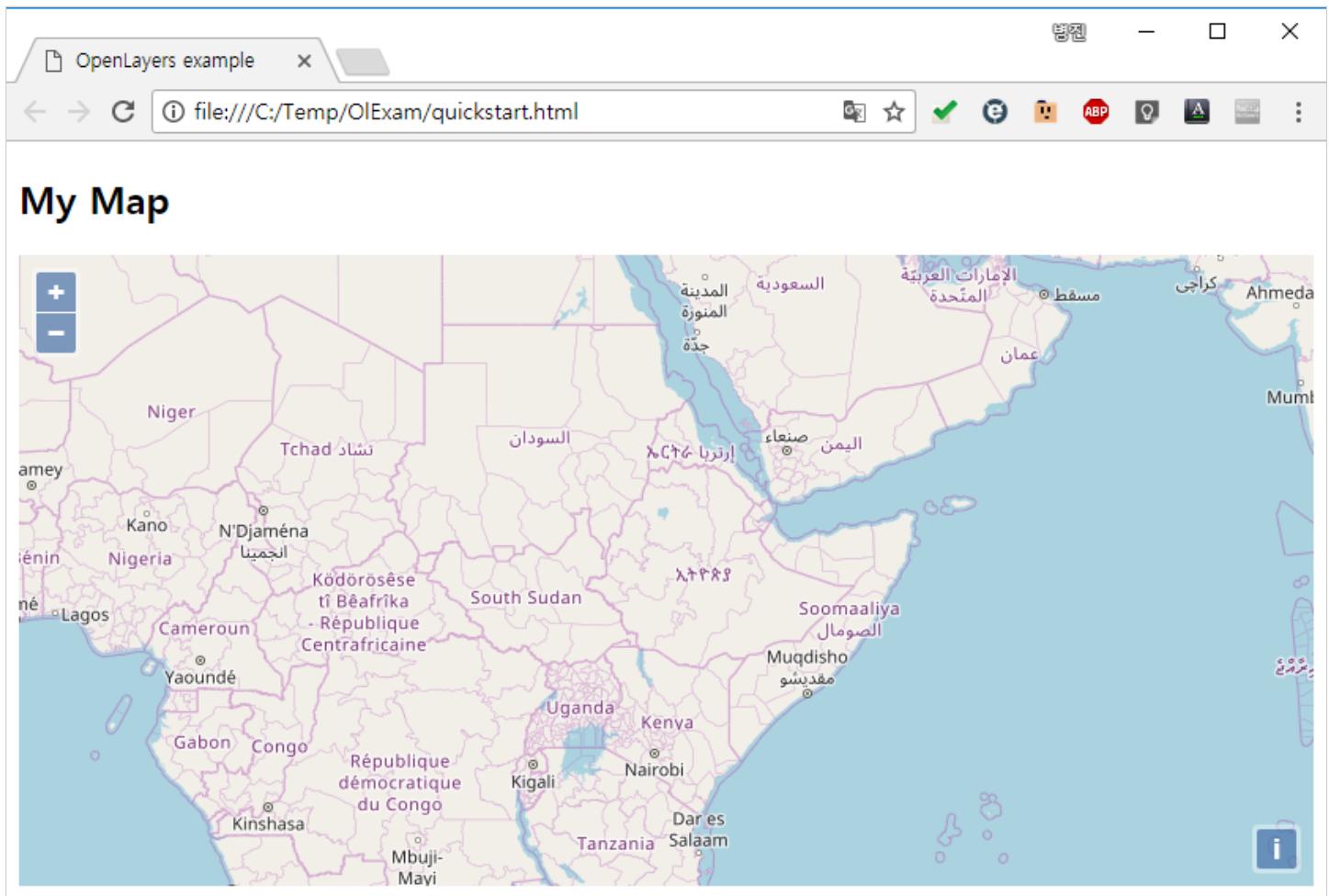
먼저 무작정 지도를 띄워 보도록 하겠습니다.

<http://openlayers.org/en/latest/doc/quickstart.html>

Notepad++에서 새문서를 열고, 다음 코드를 붙여넣고, C:\Temp\OlExam\quickstart.html이라는 이름으로 저장하고, 저장된 quickstart.html을 실행하면 지도가 보입니다.

```
<!doctype html>
<html lang="en">
<head>
<link rel="stylesheet" href="https://openlayers.org/en/v4.6.4/css/ol.css" type="text/css">
<style>
.map {
  height: 400px;
  width: 100%;
}
</style>
<script src="https://openlayers.org/en/v4.6.4/build/ol.js" type="text/javascript"></script>
<title>OpenLayers example</title>
</head>
<body>
<h2>My Map</h2>
<div id="map" class="map"></div>
<script type="text/javascript">
var map = new ol.Map({
  target: 'map',
  layers: [
    new ol.layer.Tile({
      source: new ol.source.OSM()
    })
  ],
  view: new ol.View({
    center: ol.proj.fromLonLat([37.41, 8.82]),
    zoom: 4
  })
});
```

```
</script>
</body>
</html>
```



30줄이 조금 넘어 길어 보이지만, 실제 소스 부분은 Javascript 부분인 다음이 전부입니다.

```
var map = new ol.Map({
  target: 'map',
  layers: [ new ol.layer.Tile({ source: new ol.source.OSM() }) ],
  view: new ol.View({ center: ol.proj.fromLonLat([37.41, 8.82]), zoom: 4 })
});
```

원 소스를 좀 더 보기 좋게 줄여 놓은 이 4줄도 실은 1줄로 써도 됩니다.

한 줄씩 살펴 보면 첫 줄에서 **new ol.Map()** 으로 OpenLayers를 객체로 생성해 map이라는 변수에 저장하네요.

그 다음 3줄은 ol.Map() 함수가 받는 옵션값 들입니다. 옵션은 상당히 다양하지만, 이 3가지는 필수입니다.

두번째 줄의 target: 'map' 은 지도 콘트롤을 'map' 이라는 DIV에 넣겠다는 옵션이며, HTML 부분에서 **<div id="map" class="map"></div>** 이라 지정된 부분을 쓰겠다는 것입니다.

또 이 DIV의 크기는 Style 부분에서 **map { height: 400px; width: 100%; }** 으로 지정되어 높이 400에 넓이는 가득 차게 설정되었네요.

세번째 줄은 지도의 데이터로 사용할 레이어를 지정하는 부분인데, Tile 형태 지도 중에서도 OSM을 소스로 이용하고 있군요.

이 부분을 바꾸면 구글지도나 우리가 만든 GeoServer가 제공하는 지도로도 서비스 할 수 있습니다.

또 layers가 복수이기에 한 레이어만이 아니라 여러 레이어를 리스트로 주면 겹쳐서 사용할 수도 있습니다.

네번째 줄은 세번째 줄에서 설정한 지도 데이터들을 어느 좌표를 중심으로, 어떤 축척으로 볼지 view를 설정하는 부분입니다. 여기서 사용자가 원하는 좌표계를 지정해 주면 지도 데이터의 좌표계에서 자동으로 좌표계 변환한 지도를 보여 줄 수 있습니다.

즉 지도자료는 경위도지만, 이를 좀 더 현실에 가까운 구글좌표계로 볼 수 있는 것이지요.

그리고 OpenLayers를 쓰기 위해 꼭 필요한 부분이 HTML의 Header 부분에 들어있는 다음 2줄입니다.

```
<link rel="stylesheet" href="https://openlayers.org/en/v4.6.4/css/ol.css" type="text/css">
<script src="https://openlayers.org/en/v4.6.4/build/ol.js" type="text/javascript"></script>
```

필요한 라이브러리를 임포트 하는 부분이라 생각하시면 됩니다.

OpenLayers가 CSS와 Javascript로 만들어져서 css 파일과 js 파일을 임포트 하고 있네요.

이 css 파일과 js 파일은 이렇게 인터넷에서 받아와도 되지만 로컬 파일로 저장해서 써도 됩니다.

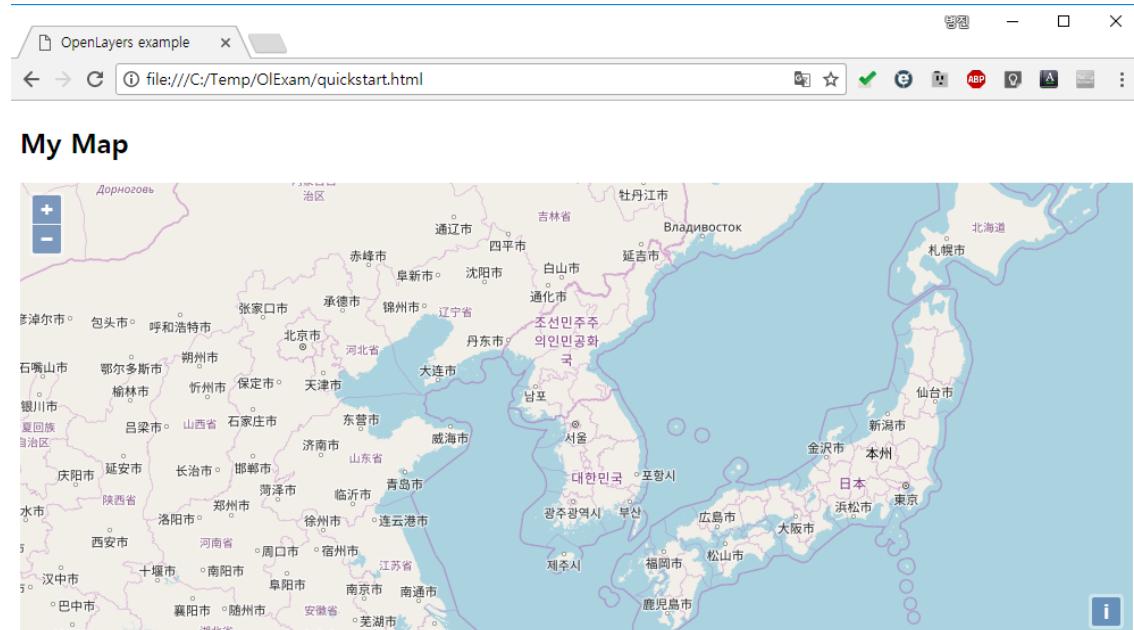
참고로 `ol/Map()` 생성시에 줄 수 있는 옵션은 다음과 같은 것들이 있습니다.

```
var map = new ol.Map({
    view: new ol.View({ center: [0, 0], zoom: 1 }),
    layers: [],
    target: 'map',
    logo: '', // 지도에 표시 되는 로고로써 표시 하지 않으려면 false를,
              // 객체를 넣어 링크와 이미지를 넣을수 있습니다.
              // 기본값은 Openlayers 3 로고 입니다.
    overlays: '', // 기본적으로 맵에 넣을 overlay를 설정합니다.
    renderer: '', // canas, 'dom', 'webgl' 을 사용할수 있습니다.
    interactions: [], // 기본적으로 넣을 ol.interaction을 설정합니다.
    controls: [] // 기본적으로 넣을 ol.control을 설정합니다.
});
```

이렇게 단 몇줄 만으로 웹페이지에 지도를 넣을 수 있지만, OpenLayers가 제공하는 기능은 굉장히 많습니다. 매뉴얼이 잘 되어 있어 개발시 많은 도움이 되지만, 이보다는 샘플을 찾아보고 내가 원하는 기능과 비슷한 것을 찾아 조금 수정하는 형태로 개발하는 것이 더 쉽습니다.

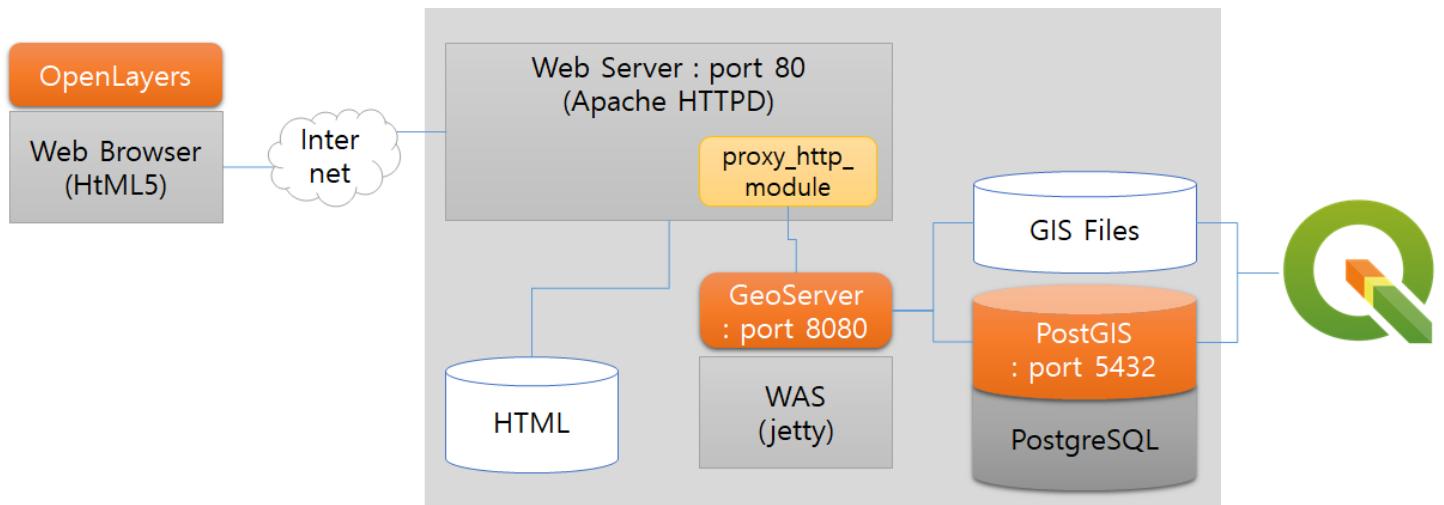
직접 해 봅시다.

- 시작 위치가 한반도가 되게 수정해 봅시다.
- 지도가 400\*400의 고정 크기를 갖게 하고, 지도 왼쪽에 지도에 대한 설명이 들어가 있는 DIV가 있도록 페이지를 만들어 봅시다.



## 6x02 서비스를 위한 Stack 구성

상용 환경에서 웹 지도 서비스를 위한 일반적인 구성은 아래 그림과 같습니다.



OpenLayers를 지도 콘트롤로 하는 웹페이지가 웹서버를 통해 서비스에 접근하고, 웹서버는 정적인 HTML 데이터를 제공하고, 공간정보는 Proxy HTTP 모듈을 통해 GeoServer에 요청합니다.  
이 때 사용자의 인터넷을 통한 요청은 80포트로 통합되어 서비스 됩니다.

GeoServer는 공간정보 파일이나 PostGIS에서 데이터를 가져와 서비스를 해 줍니다.  
이 때 공간정보는 QGIS 등 데스크탑 GIS 툴을 이용해 가공하고 관리합니다.

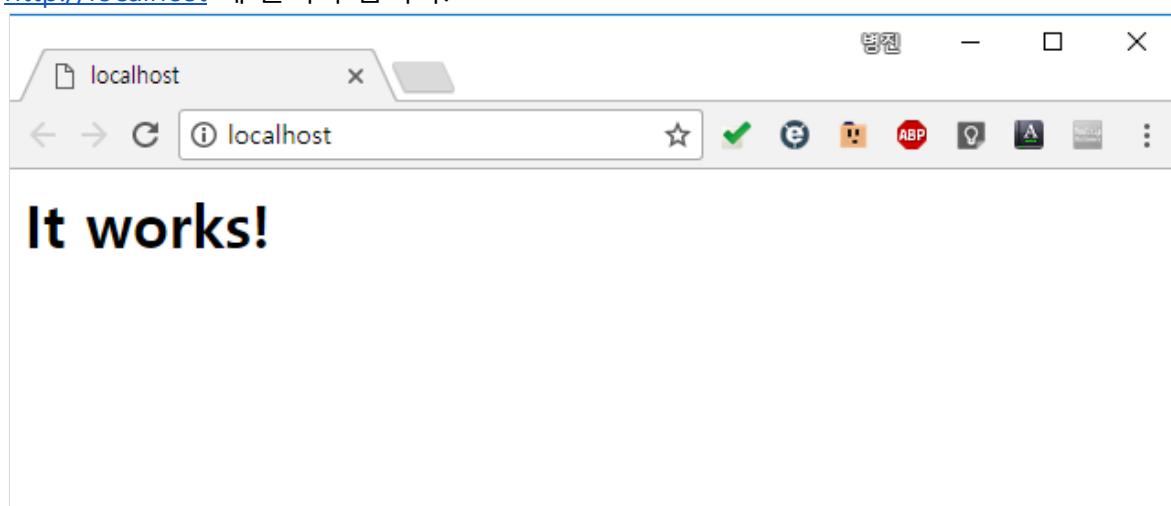
이런 서비스 스택의 구성을 이제 거의 다 배웠습니다.  
아직 안 배운 것이 웹서비스를 통해 여러가지 서비스를 80포트로 통합하는 것입니다.  
이런 기법은 보안정책 준수와 Cross Domain 오류 극복을 위해 필요합니다.

많이 사용되는 웹서버인 Apache HTTPD를 설치해 봅시다.

<https://www.apachelounge.com/download/>

Apache 2.4의 win64용 zip 파일을 받아 C:\Apache24 폴더에 압축풀어 주는 것만으로 설치는 끝납니다.  
이 때 Apache24 폴더가 2중으로 생기지 않게 주의해 주세요,  
만약 다른 경로에 설치하려면 설정파일을 여기저기 수정해 주어야 하는 불편함이 있습니다.

잘 설치되었는지 확인하기 위해 C:\Apache24\bin\httpd.exe 파일을 실행 후 웹브라우저를 띄우고 <http://localhost>에 들어가 봅시다.



이제 Apache HTTPD와 GeoServer의 연결을 위해 설정파일을 수정해 주어야 합니다.

Apache HTTPD의 설정 파일은 conf/httpd.conf 파일입니다.

이 파일의 설정으로 웹서버의 거의 모든 동작을 제어할 수 있는 아주 중요한 파일입니다.

이제 우리 목적에 맞게 수정할 것인데, 그 전에 꼭 원본을 복사해 두세요.

만약 실수로 이 설정파일이 잘못되면 웹서비스가 구동하지 않게 되기에 이렇게 수정 전에 복사해 두는 습관을 들이는 것이 좋습니다.

Notepad++로 httpd.conf 파일을 엽니다.

먼저 필요한 모듈들을 활성화 시켜야 하는데 mod\_proxy.so, mod\_proxy\_http.so 2개 모듈을 찾아 앞의 주석마크(#)을 삭제해 활성화합니다. 이 때 간혹 실수로 mod\_proxy\_html.so 모듈을 실행시키는 분이 있으니 주의하세요.

이제 mod\_proxy\_http 모듈의 동작 설정을 해 주어야 합니다.

httpd.conf 파일의 제일 마지막에 다음 내용을 추가해 줍니다.

```
<IfModule proxy_http_module>
ProxyPass /geoserver http://localhost:8080/geoserver
ProxyPassReverse /geoserver http://localhost:8080/geoserver
</IfModule>
```

내용은 외부에서 http로 /geoserver 라는 내용이 들어오면 이를 내부적으로 <http://localhost:8080/geoserver>라는 경로로 바꿔주고, 내부에서 나가는 내용에 <http://localhost:8080/geoserver>라는 내용이 있으면 /geoserver로 바꿔 내보내 주라는 것입니다.

잘 되는지 확인해 보기 위해 httpd를 종료하고 다시 시작해 봅시다.

그리고 <http://localhost/geoserver> 를 호출해 봅시다.

이렇게 :8080 포트 지정 없이 geoserver를 호출할 수 있으면 성공입니다.

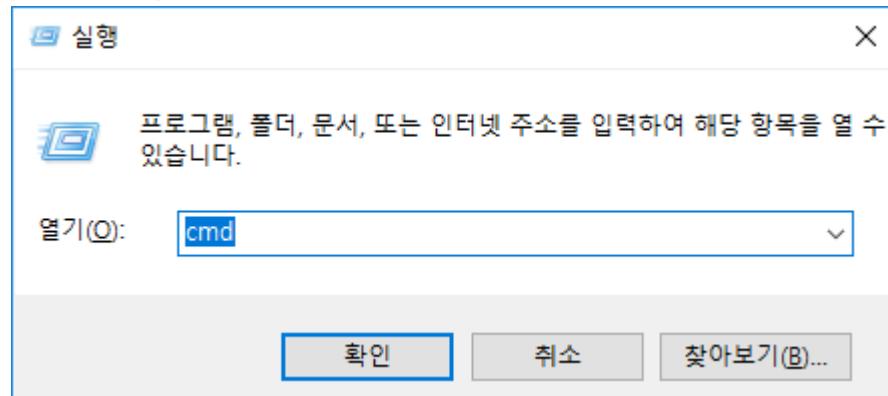
이제 웹 포트인 80 포트로 GeoServer까지 통합된 것입니다.

PostGIS는 GeoServer를 통해 연결할 수 있으니 웹서버 설정은 여기까지만 하면 우리가 원하는 스택을 모두 구성할 수 있습니다.

이번 실습에서 우리는 개발 소스를 C:\temp\olExam 폴더에 만들 것입니다.  
이 폴더를 웹서비스에서 볼 수 있도록 설정해 보겠습니다.

DocumentRoot로 지정된 폴더에 심볼릭 링크를 만들면, 여러 폴더를 통합해 줄 수 있습니다.  
httpd.conf 파일에서 DocumentRoot 부분을 수정해 HTML의 루트 경로를 바꿀 수도 있지만, 그보다는 심볼릭 링크를 이용하는 것이 여러 폴더를 체계적으로 관리하는데 유리합니다.

[윈도우 - R] 키를 눌러 실행 창을 띄우고 cmd 입력해 콘솔창을 띄웁니다.



cd 명령으로 \Apache24\htdocs 폴더로 갑니다.

윈도우에서 심볼릭 링크를 만들 수 있는 명령인 mklink를 이용해 심볼릭 링크를 생성해 줍니다.

```
mklink /D olExam c:\temp\olExam
```

cd 명령으로 olExam 폴더로 갑니다.

들어가지고 dir 명령으로 내용을 확인할 수 있으면 성공입니다.

A screenshot of a Windows Command Prompt window titled 'C:\WINDOWS\system32\cmd.exe'. The window shows the following command history:

```
C:\Windows\system32>Microsoft Windows [Version 10.0.16299.192]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\jangbi>cd \Apache24\htdocs
C:\Apache24\htdocs>mklink /D olExam c:\temp\olExam
olExam <<===> c:\temp\olExam에 대한 기호화된 링크를 만들었습니다.

C:\Apache24\htdocs>cd olExam
C:\Apache24\htdocs\olExam>
```

The 'olExam' directory was successfully created as a symbolic link pointing to 'c:\temp\olExam'.

<http://localhost/olExam/quickstart.html> 을 호출해 확인해 봅시다.

## 6x03 최종 목표 사이트 개발

최종 목표를 상기하기 위해 <https://thetruesize.com> 을 다시 보도록 합시다.

우리는 다음과 같은 샘플이 필요합니다.

- 배경지도를 GeoServer를 통해 WMS 타일로 받아와 서비스 하는 샘플
- The True Size of 서비스처럼 벡터를 선택해 이리저리 이동시키는 샘플
- 국가경계들을 WFS로 받오는 샘플
- 벡터를 위치에 따라 scale 변경하는 샘플

이에 맞는 샘플들을 찾아 봅시다.

샘플은 보통 다음 사이트에서 찾습니다.

<http://openlayers.org/en/latest/examples/>

The screenshot shows a web browser window displaying the 'OpenLayers Examples' website. The URL in the address bar is <http://openlayers.org/en/latest/examples/?q=rotate>. The page contains a grid of nine cards, each representing a different map example:

- Accessible Map** ([accessible.html](#))  
Example of an accessible map.
- View Animation** ([animation.html](#))  
Demonstrates animated pan, zoom, and rotation.
- Image ArcGIS MapServer** ([arcgis-image.html](#))  
Example of an image ArcGIS layer.
- Tiled ArcGIS MapServer** ([arcgis-tiled.html](#))  
Example of a tiled ArcGIS layer.
- Attributions** ([attribution.html](#))  
Example of attributions visibly change on map resize, to collapse them on small maps.
- Bing Maps** ([bing-maps.html](#))  
Example of a Bing Maps layer.
- Blend Modes** ([blend-modes.html](#))  
Shows how to change the canvas compositing / blending mode in post- and precompose eventhandlers.
- Box Selection** ([box-selection.html](#))  
Using a DragBox interaction to select features.
- Custom Tooltips** ([button-title.html](#))  
This example shows how to customize the buttons tooltips with Bootstrap.

## ■ 배경지도를 GeoServer를 통해 WMS 타일로 받아와 서비스 하는 샘플 → wms.html

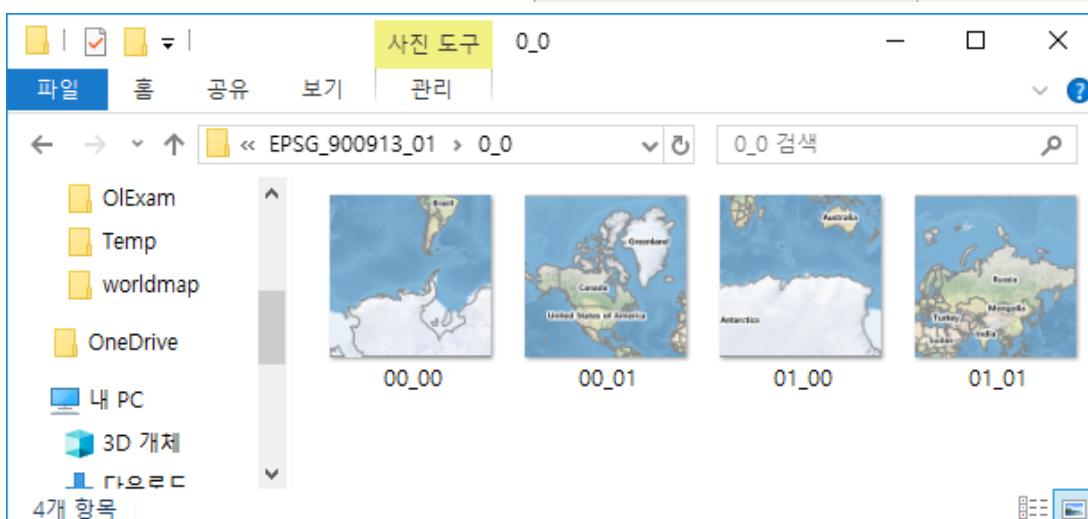
- <http://openlayers.org/en/latest/examples/wms-tiled.html>
- 앞의 quickstart.html을 **wms.html**로 저장합니다.
- script 부분을 샘플에서 복사에 내 소스에 넣어줍니다.
- 잘 되는지 확인합니다.
- WMS를 받아오는 url을 내 localhost의 GeoServer로 바꿔줍니다.  
url: 'http://localhost:8080/geoserver/wms'
- 잘 되는지 확인합니다. 개발자 도구의 Network 분석을 통해 꼭 localhost에서 오는지 확인합니다.
- WMS 요청 레이어를 'worldmap'으로 변경해 주세요.
- WMS 레이어 범위가 전세계가 되게 수정해 주세요.
- 시작시 전세계가 보이게 수정해 주세요.
- 기본 레이어를 localhost가 서비스하는 'worldmap' 레이어가 되게 수정해 주세요.
- 네트워크 분석을 통해 브라우저 캐시와 서버 캐시가 정상 동작중인지 확인해 주세요.
- WMS를 받아오는 url을 gwc가 제공하는 경로로 바꿔줍니다.  
url: '<http://localhost:8080/geoserver/gwc/service/wms>'
- '400: No SRS specified' 오류를 해결해 봅시다.  
"SRS": "EPSG:3857"
- 네트워크 분석에서 Response Header의 geowebcache-tile-index 값을 확인해 봅시다.
- 이 레이어의 좌표계를 명확히 지정해 봅시다.  
projection: 'EPSG:3857'

## WMS



Screenshot of the Chrome DevTools Network tab showing the request for 'wms.html'. The request details are as follows:

- Name: wms.html
- Request Method: GET
- Status Code: 200 OK (from memory cache)
- Remote Address: [::1]:8080
- Referrer Policy: no-referrer-when-downgrade
- Response Headers:
  - Cache-Control: max-age=31536000, must-revalidate
  - Content-Disposition: inline; filename=geoserver-di.h.image
  - Content-Length: 149315
  - Content-Type: image/png
  - Expires: Fri, 08 Feb 2019 07:10:24 GMT
  - geowebcache-cache-result: HIT
  - geowebcache-crs: EPSG:900913
  - geowebcache-gridset: EPSG:900913
  - geowebcache-tile-bounds: 10018754.169999998,0.0,208.34,10018754.169999998
  - geowebcache-tile-index: [3, 2, 2], [3, 2, 2]
  - Last-Modified: Thu, 08 Feb 2018 07:10:14 GMT
- 14 requests | 0 B transferred | Finish: 486 ms...



## ■ The True Size of 서비스처럼 벡터를 선택해 이리저리 이동시키는 샘플 → translate.html

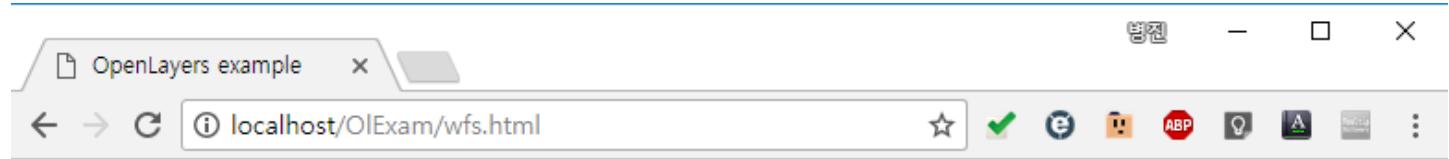
- <http://openlayers.org/en/latest/examples/translate-features.html>
- 앞의 wms.html을 **translate.html**로 저장합니다.
- 샘플에서 raster, vector, select, translate 변수 부분을 복사해 와서 translate.html의 스크립트 부분에 넣어줍니다.
- raster 변수의 레이어를 내 GeoServer에서 ol.source.TileWMS로 받아오던 레이어로 바꿔줍니다.
- layers 변수를 [raster, vector]로 바꿉니다.
- 동작을 확인합니다.
- map 변수를 초기화 하는 부분에 interactions 을 샘플을 참고해 추가합니다.
- 동작을 확인합니다.
- vector 레이어의 GeoJSON을 받아오는 url을 우리 GeoServer의 worldmap:ne\_110m\_admin\_0\_countries에서 받아오도록 변경합니다.  
[http://localhost:8080/geoserver/wfs?service=WFS&version=1.1.0&request=GetFeature&typename=worldmap:ne\\_110m\\_admin\\_0\\_countries&outputFormat=application/json&srsname=EPSG:4326&bbox=-180,-90,180,90,EPNG:4326](http://localhost:8080/geoserver/wfs?service=WFS&version=1.1.0&request=GetFeature&typename=worldmap:ne_110m_admin_0_countries&outputFormat=application/json&srsname=EPSG:4326&bbox=-180,-90,180,90,EPNG:4326)
- 정상동작 않음을 확인합니다.
- raster와 vector 레이어의 url에서 'http://localhost:8080'을 제거해 줍니다.
- Apache Web Server를 통해 같은 html을 불러 봅시다.
- <http://localhost/OIExam/translate.html>
- 동작을 확인합니다.
- CORS의 개념을 알아봅시다.
- [https://developer.mozilla.org/ko/docs/Web/HTTP/Access\\_control\\_CORS](https://developer.mozilla.org/ko/docs/Web/HTTP/Access_control_CORS)

## Translate



## ■ WFS로 벡터데이터를 받아오는 샘플

- <http://openlayers.org/en/latest/examples/vector-wfs.html>
- 앞의 translate.html을 wfs.html로 저장합니다.
- <script> ... </script> 부분을 복사해 와 wms.html에 교체합니다.
- 로컬에서 실행해 동작을 확인합니다.  
file:///C:/Temp/OIExam/wfs.html
- 개발자 도구를 통해 오동작의 원인을 확인합니다.
- Key가 없어 동작하지 않는 BingMap 대신 우리 GeoServer 배경맵으로 바꿔봅시다.
- 권한이 없어 받아올 수 없는 wfs를 우리 wfs로 바꿔봅시다.
- 웹서버를 통해 실행해 동작을 확인합니다.



## WFS

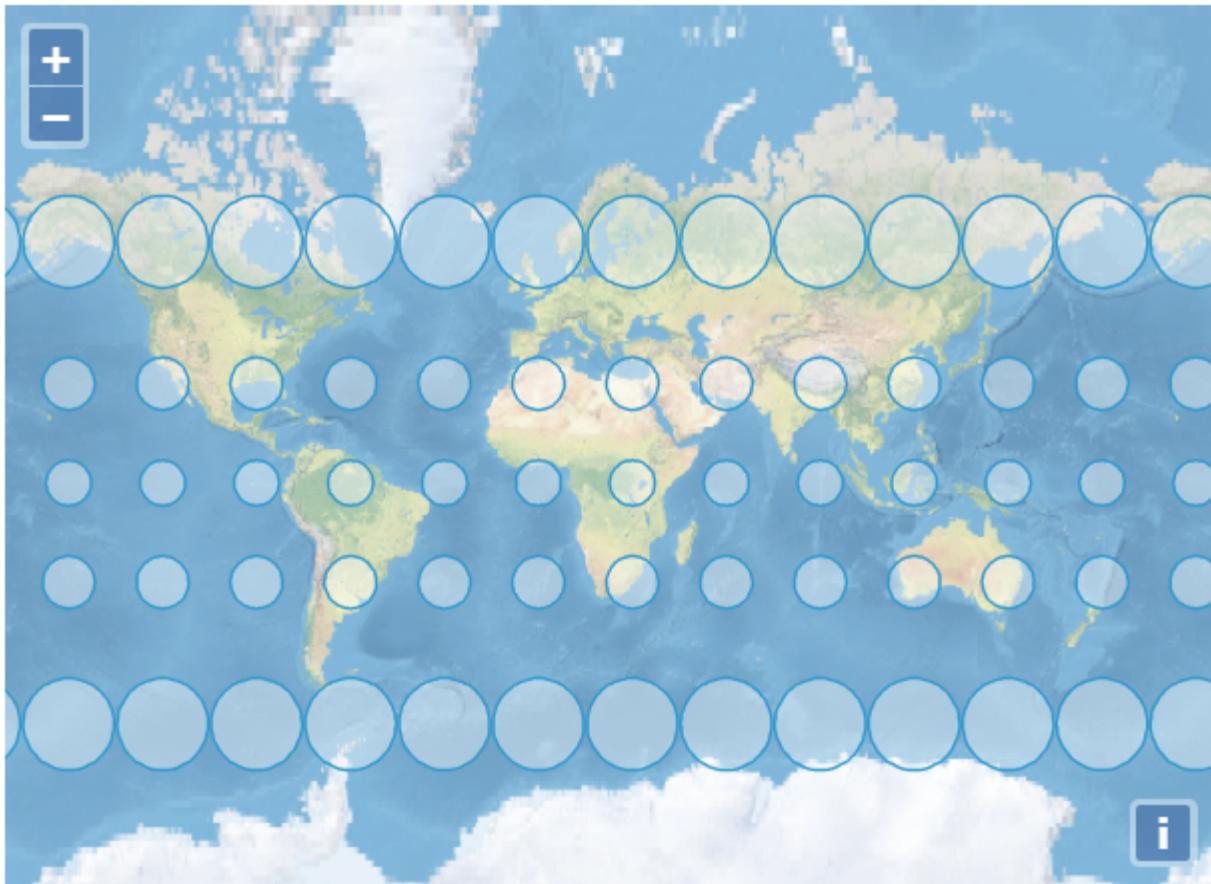


## ■ 벡터를 위치에 따라 scale 변경하는 샘플

- <http://openlayers.org/en/latest/examples/tissot.html>
- <http://openlayers.org/en/latest/apidoc/ol.interaction.Translate.html>
- 앞의 translate.html (wfs.html 아님)을 trueSize.html로 저장합니다.

구글에서 사용하는 좌표계인 EPSG:3857의 메르카토르 도법을 이해해 봅시다.

이 도법에서는 지구본 상의 원 모양은 유지되지만 원의 크기는 극지로 갈 수록 엄청나게 커집니다.



과연 어떤 비율로 커지는 것일까요?

각 위도에서의 길이 변화를 보면 적도(0도)상 크기가 1이라면, 30도는  $\sqrt{3}/2$ , 45도는  $\sqrt{2}/2$ , 60도는  $2/1$ 로 변합니다.

고등학교 수학시간에 이와 비슷한 함수를 배운 기억이 나세요?

$1/\cos \theta$ 의 비율로 변하고 있는 것입니다.

그렇다면, 메르카토르 도법에서 위도에 따라 확대되어 있는 객체를 원래대로 바꾸려면  $\cos \theta$ 를 곱해주면 되겠네요!

- ol.interaction.Translate interaction에 추가적인 이벤트를 붙일 수 있습니다.
- 변형이 시작되는 이벤트는 'translatestart'이고 여기서 선택된 객체의 중심 위도를 계산할 수 있습니다.
- 변형중에는 'translating' 이벤트가 계속 발생하고 이 때 객체를 선택된 위도와 현재 위도를 이용해 scale() 함수로 크기를 조정해 주어야 합니다.
- 변수의 값을 확인해 보는 좋은 방법 중 하나가 console.log() 함수로 값을 찍어보는 것입니다.
- 이벤트 객체에서 features.getArray()[0]하면 선택된 객체를 받을 수 있습니다.
- feature.getGeometry().getExtent() 함수로 객체의 범위를 찾을 수 있습니다.
- ol.extent.getCenter() 함수로 범위의 중앙점을 찾을 수 있습니다.
- ol.proj.transform(center\_3857, 'EPSG:3857', 'EPSG:4326') 함수로 구글좌표계를 경위도로 변환할 수 있습니다.
- feature.getGeometry().scale() 함수로 객체의 크기를 조정할 수 있습니다.



## True Size



## 6x04 개발 소스

---

### wms.html

```

<!doctype html>
<html lang="en">
<head>
<link rel="stylesheet" href="https://openlayers.org/en/v4.6.4/css/ol.css" type="text/css">
<style>
.map {
  height: 400px;
  width: 100%;
}
</style>
<script src="https://openlayers.org/en/v4.6.4/build/ol.js" type="text/javascript"></script>
<title>OpenLayers example</title>
</head>
<body>
<h2>WMS</h2>
<div id="map" class="map"></div>
<script type="text/javascript">
var layers = [
  new ol.layer.Tile({
    // extent: [-13884991, 2870341, -7455066, 6338219],
    source: new ol.source.TileWMS({
      url: 'http://localhost:8080/geoserver/gwc/service/wms',
      params: {'LAYERS': 'worldmap', 'TILED': true, "SRS": "EPSG:3857"},
      serverType: 'geoserver',
      transition: 0
    })
  })
];
var map = new ol.Map({
  layers: layers,
  target: 'map',
  view: new ol.View({
    center: [10000000, 5000000],
    zoom: 2
  })
});
</script>
</body>
</html>

```

## translate.html

```
<!doctype html>
<html lang="en">
<head>
<link rel="stylesheet" href="https://openlayers.org/en/v4.6.4/css/ol.css" type="text/css">
<style>
.map {
  height: 400px;
  width: 100%;
}
</style>
<script src="https://openlayers.org/en/v4.6.4/build/ol.js" type="text/javascript"></script>
<title>OpenLayers example</title>
</head>
<body>
<h2>Translate</h2>
<div id="map" class="map"></div>
<script type="text/javascript">
var raster = new ol.layer.Tile({
  source: new ol.source.TileWMS({
    url: '/geoserver/gwc/service/wms',
    params: {'LAYERS': 'worldmap', 'TILED': true, "SRS": "EPSG:3857"},
    serverType: 'geoserver',
    transition: 0
  })
});

var vector = new ol.layer.Vector({
  source: new ol.source.Vector({
    url:
'/geoserver/wfs?service=WFS&version=1.1.0&request=GetFeature&typename=worldmap:ne_11_0m_admin_0_countries&outputFormat=application/json&srsname=EPSG:4326&bbox=-180,-90,180,90,EPSC:4326',
    format: new ol.format.GeoJSON()
  })
});

var select = new ol.interaction.Select();

var translate = new ol.interaction.Translate({
  features: select.getFeatures()
});

var layers = [ raster, vector ];
var map = new ol.Map({
  interactions: ol.interaction.defaults().extend([select, translate]),
  layers: layers,
  target: 'map',
  view: new ol.View({
    center: [10000000, 5000000],
    zoom: 2
  })
});
</script>
</body>
</html>
```

## wfs.html

```
<!doctype html>
<html lang="en">
<head>
<link rel="stylesheet" href="https://openlayers.org/en/v4.6.4/css/ol.css" type="text/css">
<style>
.map {
  height: 400px;
  width: 100%;
}
</style>
<script src="https://openlayers.org/en/v4.6.4/build/ol.js" type="text/javascript"></script>
<title>OpenLayers example</title>
</head>
<body>
<h2>WFS</h2>
<div id="map" class="map"></div>
<script>
var vectorSource = new ol.source.Vector({
  format: new ol.format.GeoJSON(),
  url: function(extent) {
    return '/geoserver/wfs?service=WFS&' +
'version=1.1.0&request=GetFeature&typename=worldmap:ne_110m_admin_0_countries&' +
'outputFormat=application/json&srsname=EPSG:3857&' +
'bbox=' + extent.join(',') + ',EPSG:3857';
},
  strategy: ol.loadingstrategy.bbox
});

var vector = new ol.layer.Vector({
  source: vectorSource,
  style: new ol.style.Style({
    stroke: new ol.style.Stroke({
      color: 'rgba(0, 0, 255, 1.0)',
      width: 2
    })
  })
});

var raster = new ol.layer.Tile({
  source: new ol.source.TileWMS({
    url: '/geoserver/gwc/service/wms',
    params: {'LAYERS': 'worldmap', 'TILED': true, "SRS": "EPSG:3857"},
    serverType: 'geoserver',
    transition: 0
  })
});

var map = new ol.Map({
  layers: [raster, vector],
  target: document.getElementById('map'),
  view: new ol.View({
    center: [-8908887.277395891, 5381918.072437216],
    maxZoom: 19,
    zoom: 12
  })
});
```

```

        });
    });
</script>
</body>
</html>

```

## trueSize.html

```

<!doctype html>
<html lang="en">
<head>
<link rel="stylesheet" href="https://openlayers.org/en/v4.6.4/css/ol.css" type="text/css">
<style>
.map {
  height: 400px;
  width: 100%;
}
</style>
<script src="https://openlayers.org/en/v4.6.4/build/ol.js" type="text/javascript"></script>
<title>OpenLayers example</title>
</head>
<body>
<h2>True Size</h2>
<div id="map" class="map"></div>
<script type="text/javascript">
var raster = new ol.layer.Tile({
  source: new ol.source.TileWMS({
    url: '/geoserver/gwc/service/wms',
    params: {'LAYERS': 'worldmap', 'TILED': true, "SRS": "EPSG:3857"},
    serverType: 'geoserver',
    transition: 0
  })
});

var vector = new ol.layer.Vector({
  source: new ol.source.Vector({
    url:
      '/geoserver/wfs?service=WFS&version=1.1.0&request=GetFeature&typename=worldmap:ne_110m_admin_0_countries&outputFormat=application/json&srsname=EPSG:4326&bbox=-180,-90,180,90,EPSC:4326',
    format: new ol.format.GeoJSON()
  })
});

var select = new ol.interaction.Select();

var translate = new ol.interaction.Translate({
  features: select.getFeatures()
});

var cut_lat = null;
translate.on('translatestart', function(evt){
  // 현재위도
  var selected_feature = evt.features.getArray()[0];

```

```

cur_lat = getLatFromFeatureCenter(selected_feature);
console.log(cur_lat);
});

translate.on('translating', function(evt){
// 이전위도
var old_lat = cur_lat;

// 현재위도
var selected_feature = evt.features.getArray()[0];
cur_lat = getLatFromFeatureCenter(selected_feature);

// scale 적용
var crr_scale = 1/(cur_lat/old_lat);
selected_feature.setGeometry().scale(crr_scale);
});

function getLatFromFeatureCenter(feature) {
var feature_extent = feature.getGeometry().getExtent();
var center_3857 = ol.extent.getCenter(feature_extent);
var center_4326 = ol.proj.transform(center_3857, 'EPSG:3857', 'EPSG:4326');
return Math.cos(center_4326[1] * Math.PI / 180.0); // radian -> degree
}

var layers = [ raster, vector ];
var map = new ol.Map({
interactions: ol.interaction.defaults().extend([select, translate]),
layers: layers,
target: 'map',
view: new ol.View({
center: [10000000, 5000000],
zoom: 2
})
});
</script>
</body>
</html>

```