Introduction to PostgreSQL

PostgreSQL (or Postgres for short) is a popular open source object relational database server.

- Runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, macOS, Solaris, Tru64), and Windows.
- It is fully ACID compliant (standard database speak - Atomicity, Consistency, Isolation, Durability).
- PostgreSQL runs stored procedures in more than a dozen programming languages, including Java, Perl, Python, Ruby, Tcl, C/C++, and its own PL/pgSQL, which is similar to Oracle's PL/SQL

Term	Description
Relational	A relational database is a set of tables containing data fitted into predefined categories. Each table (which is sometimes called a relation) contains one or more data categories in columns. Each row contains a unique instance of data for the categories defined by the columns.
Object-Relational	An object relational database uses an object-oriented model. Objects, classes and inheritance are supported. Particularly in PosgreSQL, this includes table inheritance.
SQL	Structured Query Language. Different database servers use diferent 'flavours' of SQL. PostgreSQL SQL implementation strongly conforms to the ANSI-SQL:2008 standard.
Open Source	The PostgreSQL Licence gives you the freedom to use, modify and distribute PostgreSQL in any form you like, open or closed source. Any modifications, enhancements, or changes you make are yours to do with as you please.
Extensible	PostgreSQL includes a framework that allows developers to define and create their own custom data types along with supporting functions and operators that define their behavior. As a result, a host of advanced data types have been created that range from geometric and spatial primitives to network addresses to even ISBN/ISSN (International Standard Book Number/International Standard Serial Number) data types, all of which can be optionally added to the system.

Limits

There are some data limits within PostgreSQL but they are quite high.

Limit	Value
Maximum Database Size	Unlimited
Maximum Table Size	32 TB
Maximum Row Size	1.6 TB
Maximum Field Size	1 GB
Maximum Rows per Table	Unlimited
Maximum Columns per Table	250 - 1600 depending on column types
Maximum Indexes per Table	Unlimited

Installing PostgreSQL

PostgreSQL is free and open source. The recommended method of installing PostgreSQL is to use a packaged installer from EnterpriseDB. This includes download packages for multiple operating systems:

- Linux 32bit
- Linux 64bit
- Windows 32bit
- Windows 64bit
- Mac OSX

Download	Description
EnterpriseDB	Packaged versions of PostgreSQL, including admin client for multiple operating systems
Source code	Source code download for postgreSQL v9.6.2

Install using EnterpriseDB (Windows)

1. Run the installer
2. When prompted to create a password choose an appropriate password for the service and superuser account. Further accounts can be created later on if required.
3. The default port is 5432. Leave this as is unless you need to run on a different port.
4. Leave as the Default Locale
5. Deselect the option to run StackBuilder after the install.

PostgreSQL should now be installed! By default the EnterpriseDB package comes with pgAdmin 4. This is the current default client.

Turning off the server

When installed, by default the database server will run when the server (or PC) is started. To change this you can turn off the automatic startup. In Windows:

1. Open the start menu and search for 'services.msc'. Run this.
2. Scroll down the list of services and find Postgres.
3. Set Right click > Properties > Startup Type > Manual.
4. Within this properties window you can also start and stop the service.

Linked servers

When installed, PostgreSQL can be set up on a server to allow remote connection from clients. It can also handle connections from other database technologies, such as SQL Server. For example if you wanted to transfer data between clients, or set up databases that queried each other. This would be using SQL Server Linked Server technology.

For more info on connecting a PostgreSQL database as a linked server see SQL Server and

PostgreSQL linked server configuration.

PostgreSQL Clients

There are a selection of Graphical User Interfaces (GUIs) to use when working with PostgreSQL databases. We will show you three of these clients.

- pgAdmin III (https://www.pgadmin.org/include/doc.php?docset=1.8&docpage=index.html)
- pgAdmin 4 (https://www.pgadmin.org/download/windows4.php)
- Navicat (https://www.pgadmin.org/download/windows4.php)

pgAdmin 4

pgAdmin 4 is the latest version of the official PostgreSQL client. It comes bundled with the EnterpriseDB installer.

- Sign up to the PostgreSQL Community to see Issue Tracking.

Pros	Description
Open Source	Developed as open source, this has a good developer community, forums, and free licence terms.
Free	gAdmin is made available for free under the PostgreSQL Licence.
Modern UI	pgAdmin 4 is a rewrite of the pgAdmin interface, uses web development styling and more modern UI.
Official	The most popular and feature rich Open Source administration and development platform for PostgreSQL. pgAdmin is a project that is part of the PostgreSQL project.
Active development	Features will be addded and improved.
Installed	As pgAdmin 4 is included in the EnterpriseDB installer package, no additional setup is necessary.

Cons	Description
Buggy	While fairly early in development there are many bugs in pgAdmin 4. New releases are frequent though, with a considerable number of bug fixes in each one.
Incomplete	There are some features within pgAdmin 4 that don't appear to be complete, or are not fully tested.

pgAdmin III

pgAdmin III is the previous version of the pgAdmin client. As pgAdmin 4 is a major rewrite, and still in fairly early development, this is still worth considering.

Pros	Description
Open Source	Developed as open source, this has a good developer community, forums, and free licence terms.
Free	gAdmin is made available for free under the PostgreSQL Licence.
Reliable	pgAdmin III has been through many iterations and is more reliable than the latest version.
Familiar	For many people, the style of pgAdmin III is familiar, and will be the most commonly used environment.

Pros	Description
Official	The most popular and feature rich Open Source administration and development platform for PostgreSQL. pgAdmin is a project that is part of the PostgreSQL project.
Fully featured	pgAdmin III covers all popular features of creating and administering databases in PostgreSQL up to a recent version of PostgreSQL

Cons	Description
Unsupported	pgAdmin III is no longer supported. It is likely to stop working in future versions of the database server.
Discontinued	There is no longer development work on pgAdmin III.
Old UI	Slightly outdated user interface, logos, and experience

Navicat

Navicat for PostgreSQL is one of many Navicat products for different database servers, and is a commercial offering.

- Free 14 day trial available at Navicat free trial

Pros	Description
Support	Navicat is a commercially supported product
Modern UI	Good looking, modern user experience
Streamlined UI	Feels better for SQL development

Cons	Description
Cost	Licence cost varies

Try out different tools and make your own decision! See this Reddit discussion for various opinions.

Creating PostgreSQL databases

It is easy to create databases in postgreSQL using either the GUI, or by directly running SQL queries.

Creating a database

- In pgAdmin 4 Connect to the database server.
- In the server tree select the postgres database.
- Open up the query tool. Tools > Query Tool.
- Create a new database using the following query. Once you've written the query click the Lightning icon to run.
- CREATE DATABASE documentation

```
CREATE DATABASE training;
```

On refreshing the server tree by right clicking and selecting 'Refresh' you will see the new database. From now on we'll do everything on this database.

An alternative to this would be to use the GUI. Right click on databases and select Create.

Creating a table

- CREATE TABLE documentation

Databases are made up of tables. To create these we can either run SQL statements or use the GUI.

```
CREATE TABLE postcodes (
    postcode varchar(8),
    positional_quality_indicator integer,
    po_box_indicator char(1),
    total_number_of_delivery_points integer,
    delivery_points_cplc integer,
    domestic_delivery_points integer,
    non_domestic_delivery_points integer,
    po_box_delivery_points integer,
    matched_address_premises integer,
    unmatched_delivery_points integer,
    country_code varchar(9),
    nhs_regional_ha_code varchar(9),
    nhs_ha_code varchar(9),
    admin_county_code varchar(9),
    admin_district_code varchar(9),
    admin_ward_code varchar(9),
    postcode_type char(1),
    wkt text,
    CONSTRAINT pk_postcodes_postcode PRIMARY KEY(postcode)
);
```

Data types

One of the most important considerations in creating databases is the data types used in tables. Here are a selection of commonly used PostgreSQL data types. When creating database tables PostgreSQL will handle a number of common ways of specifying data types. For example, character(10) is the same as char(10).

Type	Description
character varying(10)/varchar(10)	variable-length characters with a specified limit
character(10)/char(10)	fixed-length set of characters
text	variable unlimited length
integer	non decimal number -2147483648 to +2147483647
numeric	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
serial	auto-incrementing integer
date	a date value
timestamp	a date and time value
integer array/integer[]	2 dimensional and 3 dimensional arrays of a specified type

Loading data

Data can be manually loaded into PostgreSQL using SQL commands.

- INSERT INTO documentation

```
INSERT INTO postcodes
VALUES('EX1
1EE',10,'N',32,32,18,14,0,15,0,'E92000001','E19000002','E18000010','E10000008','E07
000041','E05003502','S','POINT (292079 92307)');
```

Postgres also allows for loading in data using the COPY command. Typically this will be loading into individual tables from CSV data.

A tip for combining CSV files in Windows is to use the command prompt copy command. For example, in a directory full of postcode CSV files run:

- Microsoft copy syntax

```
copy *.csv postcodes.csv
```

Now, taking an Ordnance Survey sample from code point we'll use the PostgreSQL COPY command to import all postcodes from a CSV file.

Before we do this we need to clear down the postcode added previously. For this we can just wipe all data from the table.

- TRUNCATE documentation

```
TRUNCATE postcodes;
```

Then we can load in the postcodes from Code Point.

- COPY documentation

```
COPY postcodes FROM
'C:\Development\DaveBathnes\PostgreSQL-Training\data\codepoint.csv' HEADER CSV;
```

PostgreSQL Common Queries

Brief overview of selecting and querying data in PostgreSQL. Many of these commands will be similar/identical to commands run in other database servers e.g. SQL Server, Oracle.

Using the pgAdmin 4 GUI

1. In the pgAdmin explorer navigate in the tree to Databases > Training > Schemas > Public > Tables
2. Right click on the postcodes table and select View All Rows

Selecting data

Select particular columns from the data table. In the case below, just the postcode and positional quality indicator.

- SELECT statement documentation

```
SELECT postcode, positional_quality_indicator FROM postcodes;
```

Where clauses

Filter the data by specifying values that columns must conform to. In the case below, where the postcode begins 'EX'.

- WHERE clause featured in SELECT documentation

```
SELECT * FROM postcodes WHERE postcode LIKE 'EX%';
```

Order By

Modify the order in which the data is returned. In the case below, postcodes sorted in reverse alpabetical order.

- ORDER BY documentation

```
SELECT postcode FROM postcode ORDER BY postcode DESC;
```

Group By

Commonly used for aggregating and counting data. In the case below, count postcodes by first two letters.

- GROUP BY featured in SELECT documentation

```
SELECT SUBSTRING(postcode, 1, 3), COUNT(*) FROM postcodes GROUP BY
SUBSTRING(postcode, 1, 3);
```

Update Data

SQL Update commands will update data in a database table. It is important to use the WHERE clause with update statements as they will otherwise update every row. For example, updating a postcode to another postcode:

- UPDATE statement documentation

```
UPDATE postcodes
SET postcode = 'EX1 4BA'
WHERE postcode = 'EX1 4BB'
```

GIS data types

Format	Description
GeoJSON	Based on JSON, often used in web development.
KML	XML based open standard.
GML	XML based. An open spatial data schema to enable interoperability between independent GIS applications and with internet GIS in mind
Shapefile	Proprietary and popular ESRI file format.
CSV	Data held in traditional CSV files

GeoJSON

Aspect	Details
JSON	JavaScript Object Notation. Based on the JavaScript programming language and ideal for use in web maps.
Lightweight	Text based, and not too much markup
Readable	Relatively easy to open in a text editor and read

KML

Aspect	Details
XML	eXtensible Markup Language. Based upon XML.
Google	Developed for use with Google Earth, which was originally named Keyhole Earth Viewer
Open	KML became an international standard of the Open Geospatial Consortium in 2008

Shapefiles

Aspect	Details
Proprietary	ESRI based standard
Common	
Integration with PostGIS	Easy loading and exporting from PostGIS

GML

Aspect	Details
XML	eXtensible Markup Language. Based upon XML.
OS	Used by Ordance Survey as part of certain products such as MasterMap.
Tricky	Can be tricky to use and import into databases while maintaining all the data

CSV

Aspect	Details
Comma separated values	Columns are separated (typically) by commas. Data that could itself contain a comma is normally wrapped in quotes. e.g. and address "110 The Laggar, Corsham"

Aspect	Details
Tabular	Can be opened in a variety of software that takes table data, such as Excel and Access
WKT	GIS data is often included in CSV through the use of 'Well Known Text' standard, or alternatively simple X/Y values in separate columns
Readable	Easy to open in a text editor and read

```
EX1
1EE,10,N,32,32,18,14,0,15,0,E92000001,E19000002,E18000010,E10000008,E07000041,E0500
3502,S,POINT (292079 92307)
```

Introduction to PostGIS

PostGIS is a set of GIS based extensions for PostgreSQL databases. As PostgreSQL is extensible, PostGIS is simply one of many extensions, but one of the most popular.

Aspect	Description
Open Source	Released under the GNU General Public License (GPLv2 or later)
Documentation	Excellent documentation on the documentation site
Features	Many features rarely found in other competing spatial databases. See PostGIS Feature list

Installing PostGIS

PostGIS needs to be installed separately to the main PostgreSQL install.

The recommended way of installing PostGIS is via the PostGIS site. Be careful to choose a version that is compatible with your version of PostgreSQL. If you are downloading the latest of both PostGIS and PostgreSQL those should be compatible.

Install

1. Download the latest PostGIS Windows Installer.
2. Run through the installer selecting all the defaults.

PostGIS is now installed!

Creating a PostGIS database

Creating a POstGIS database involves the same steps as a normal POstgreSQL database. Enabling POstGIS extensions on the database gives a greater variety of dat atypes and fucntions.

Create a normal database

For this purpose we're going to use our training database so we can skip this step.

- CREATE DATABASE documentation

```
CREATE DATABASE training;
```

Add PostGIS extensions

Making a spatial database involves initially adding PostGIS extensions.

On the new database, run the command to add PostGIS extensions.

- CREATE EXTENSION documentation

```
CREATE EXTENSION postgis;
```

Create a normal table

Run a create table statement.

- CREATE TABLE documentation

```
CREATE TABLE postcodes_geo (
    postcode varchar(8),
    positional_quality_indicator integer,
    po_box_indicator char(1),
    total_number_of_delivery_points integer,
    delivery_points_cplc integer,
    domestic_delivery_points integer,
    non_domestic_delivery_points integer,
    po_box_delivery_points integer,
    matched_address_premises integer,
    unmatched_delivery_points integer,
    country_code varchar(9),
    nhs_regional_ha_code varchar(9),
    nhs_ha_code varchar(9),
    admin_county_code varchar(9),
    admin_district_code varchar(9),
    admin_ward_code varchar(9),
    postcode_type char(1),
    CONSTRAINT pk_postcodesgeo_postcode PRIMARY KEY(postcode)
);
```

Add a geometry column

- AddGeometryColumn documentation

```
SELECT AddGeometryColumn ('postcodes_geo','geom',0,'POINT',2);
```

Loading CSV data

Loading CSV data can be done in the same way as we did previously. We know that the final column includes valid well known text POINT data.

- COPY documentation

```
COPY postcodes_geo FROM
'C:\Development\DaveBathnes\PostgreSQL-Training\data\codepoint.csv' HEADER CSV;
```

Set the SRID of the column

To save the Spatial Reference System being used in the column we can use a PostGIS function to set this value.

The data is British National Grid so we can set the SRID to 27700.

- UpdateGeometrySRID documentation

```
SELECT UpdateGeometrySRID('postcodes_geo','geom',27700);
```

PostGIS Data loading

We have already demonstrated loading data into a spatial database by using CSV data where the geographic data is in Well Known Text Form.

Now we need to look at loading various other data formats into PostGIS.

For these exercises we are going to be taking sample data from the Ordnance Survey, available at.

Ordnance Survey Sample Data

By downloading any of the following datasets you agree to the terms of our Data Exploration Licence. It lets you use the data to research and develop ideas and propositions, create working prototypes, trial as part of your business use for up to 3 months, take part in hackathons and share knowledge.

For convenience, the data in these exercises is included in the data directory of these notes.

Load AddressBase

We will be taking the AddressBase Premium data, available in GML format.

PostGIS will not directly import GML data. Astun Technology have written an open source Loader for GML data.

Follow the installation instructions here.

Once that it installed we need to use the following configuration in loader.config

```
# Use the OS AddressBase Premium preparation logic
prep_cmd=python prepgml4ogr.py $file_path prep_osgml.prep_addressbase_premium

# OS AddressBase Premium gfs file to specify appropriate schema for the data
gfs_file=../gfs/addressbase_premium.gfs
```

From the python directory Astun Loader has been downloaded to, run the command:

```
python loader.py loader.config
```

The data ends up succesfully loaded into our PostGIS database.

Shapefile loader

PostGIS comes with a convenient shapefile loading tool.

1. Launch PostGIS 2.0 Shapefile and DBF Loader.

2. Fill out the connection details for the database.
3. Select the file. In this case we'll use county_region.shp (from OS Boundary Line Open). Modify the SRID to 27700.

Click Import - the data is then imported into a new table called county_region. We can then check the data has been loaded correctly.

```PLpgSQL SELECT * FROM county_region WHERE name = 'Somerset County'; ``
```

PostGIS Common Queries

There are many spatial queries that can be run within PostGIS. A definitive list is held at:

- PostGIS Functions Index

ST_X

Returns the X coordinate of the geometry.

- ST_X documentation

```
SELECT ST_X(geom) FROM postcodes_geo WHERE postcode = 'EX1 1EE';
```

ST_Y

Returns the Y coordinate of the geometry.

- ST_Y documentation

```
select ST_Y(geom) from postcodes_geo where postcode = 'EX1 1EE';
```

ST_Transform

Transforms a geometry into the specified spatial reference system, by ID. In PostGIS databases spatial reference systems are defined in the spatial_ref_sys table. ST_Transform relies on PostGIS knowing which SRS the geometry is currently in (in section 9 we used the UpdateGeometrySRID function to set this on a column).

- ST_Transform documentation

```
SELECT ST_Transform(geom, 4326) FROM postcodes_geo WHERE postcode  = 'EX1 1EE';
```

Using that to extract the Lng/Lat:

```
SELECT ST_X(ST_Transform(geom, 4326)), ST_Y(ST_Transform(geom, 4326)) FROM
postcodes_geo WHERE postcode  = 'EX1 1EE';
```

ST_DWithin

Returns results that are within a specified distance. In the case below, returns results that are within 100 metres of a certain point.

- ST_DWithin documentation

```
select postcode from postcodes_geo where ST_DWithin(geom,
ST_SetSRID(ST_MakePoint(292079, 92307), 27700), 100);
```

ST_Intersects

Returns results that share the same space. In the case below, returns postcodes that are within Devon.

- ST_Intersects documentation

```
SELECT postcode FROM postcodes_geo WHERE ST_Intersects(
    geom,
    (SELECT geom FROM county_region WHERE name = 'Devon County')
);
```

ST_Centroid

Returns the centre point of a geometry.

- ST_Centroid documentation

```
select ST_X(ST_Centroid(wkb_geometry)), ST_X(ST_Centroid(wkb_geometry)) from street
where usrn = 14202557
```

PostGIS Exporting Data

Shapefile exporter

The Shapefile Loader we used in the data loading section can also be used to export data to shapefile.

1. Launch PostGIS 2.0 Shapefile and DBF Loader.
2. Fill out the connection details for the database.
3. Change from Import to Export.
4. Click Add Table. Select the Table (e.g. postcodes_geo) and click OK.
5. Click Export and select an output folder.

The shapefile files (cpg,dbf,prj,shp,shx) will be exported into the folder.

Shapefile exporter command line

The shapefile exporter allows you to select tables and database views to export these as shapefiles. But what about custom database queries that either select certain fields, or that do some geoprocessing (such as cookie-cutting data). These could be transferred into a new table but they can also be exported to shapefile using a command line tool pgsql2shp.

The tool is installed as part of PostGIS and is installed at C:\Program Files\PostgreSQL\9.6\bin\pgsql2shp.exe

- pgsql2shp documentation

The tool can be used to run complex SQL commands. These could be scheduled and set to run whenever appropriate.

```
"C:\Program Files\PostgreSQL\9.6\bin\pgsql2shp.exe" -f "C:\output" -u postgres
training "select distinct a.* from ((select * from postcodes_geo where
ST_DWithin(geom, ST_SetSRID(ST_MakePoint(292079, 92307), 27700), 100)) union all
(select * from postcodes_geo where ST_DWithin(geom, ST_SetSRID(ST_MakePoint(292279,
92507), 27700), 100))) a"
```

COPY

We have primarily used the COPY command in PostgreSQL to import data, but it can also be used for exporting data. It is best used for exporting tabular (CSV) data.

```
COPY postcodes_geo to 'C:\output\postcodes_geo.csv' CSV;
```

That's exporting a single table to a CSV file, the same can be done with a custom query.

```
COPY (SELECT postcode, ST_X(geom), ST_Y(geom) FROM postcodes_geo) TO
'C:\output\postcode_custom.csv' CSV;
```