Open-source GIS guide

5. A guide to using a PostGIS database

A guide to loading and using spatial data in a PostGIS database

Pre- and Post-Sales Support
White paper
Unrestricted
Responsibility for this document

Pre- and Post-Sales Support is responsible for the content of this document.

Change history

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Summary of change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Mar 2013</td>
<td>First issue</td>
</tr>
</tbody>
</table>

The impacts of the process described in this document have been assessed and where appropriate, changed, in accordance with the requirements set out in Ordnance Survey's Equality Scheme.

Approval for issue

Issued by Pre- and Post-Sales Support, Ordnance Survey

Trademarks

OS VectorMap, OS MasterMap, AddressBase, Code-Point, Strategi, Ordnance Survey and the OS Symbol are registered trademarks of Ordnance Survey, the national mapping agency of Great Britain.

Disclaimer

This document is only a guide to the installation and use of open source software by members of the Public Sector Mapping Agreement (PSMA) and One Scotland Mapping Agreement (OSMA). Open source software and this guide is provided ‘as is’. Ordnance Survey makes no warranties, express or implied or arising by custom, as to the accuracy or fitness for any purpose of this guide or that open source software will be uninterrupted or error free. Except by operation of law, under no circumstances will Ordnance Survey be liable for any costs, liabilities, claims, damages or other losses, whether foreseeable or unforeseeable, either to members or any third party arising out of any breach or failure of this guide or any open source software or otherwise.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page no</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>4</td>
</tr>
<tr>
<td>2.1.1</td>
<td>4</td>
</tr>
<tr>
<td>2.1.2</td>
<td>5</td>
</tr>
<tr>
<td>2.1.3</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>8</td>
</tr>
<tr>
<td>3.2</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>4.1</td>
<td>11</td>
</tr>
<tr>
<td>4.2</td>
<td>13</td>
</tr>
<tr>
<td>4.2.1</td>
<td>13</td>
</tr>
<tr>
<td>4.2.2</td>
<td>13</td>
</tr>
<tr>
<td>4.2.3</td>
<td>15</td>
</tr>
<tr>
<td>4.2.4</td>
<td>15</td>
</tr>
<tr>
<td>4.3</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>6.1</td>
<td>18</td>
</tr>
<tr>
<td>6.2</td>
<td>19</td>
</tr>
<tr>
<td>6.3</td>
<td>21</td>
</tr>
<tr>
<td>6.3.1</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>24</td>
</tr>
<tr>
<td>7.1</td>
<td>24</td>
</tr>
<tr>
<td>7.2</td>
<td>26</td>
</tr>
</tbody>
</table>
1 Introduction

This guide forms part of a series of documents describing the implementation and use of an open-source Geographic Information System (GIS). PostGIS is the spatial data extension to the open-source PostgreSQL database. Advice on installing both applications can be found in guide 2 of this series.

In this guide we look at how to load data from a variety of sources, how to query the data and present the results in a GIS. As QGIS is the most commonly used open-source GIS, it is has been used as the GI system described in this document. However other software packages, both open-source and commercial can connect to PostGIS and it is recommended you check with your vendor on compatibility.

It has been assumed that the user of this document has some understanding of database practices and Standard Query Language (SQL) before using this guide. It is also assumed that an ODBC connection has been set up for use with PostgreSQL.

This guide has been written using version 9.1.4 of PostgreSQL and 1.5 of PostGIS.

2 Data loading

Data can be loaded into PostgreSQL using a variety of methods. In this section we will look at loading data directly into the database using SQL and tools provided with the database. However Astun Technology has made available SQL loaders for some Ordnance Survey data on their Labs web pages (http://astuntechnology.com/labs/). These include OS MasterMap®, OS VectorMap® Local and AddressBase®. Additionally, some GIS applications can connect to PostGIS and can be used to load data directly into the database. For further details please refer to your software provider.

2.1 Loading text files and converting to spatial data

The first step in loading data into the database is to create the table structure in which the data will be held. This is done using standard SQL, either within pgAdmin III or from the SQL command prompt. The example provided loads AddressBase from a single CSV file into a database table. Further details on manipulating AddressBase data can be found in the AddressBase Getting Started Guide on the Ordnance Survey website.

The PostgreSQL utility pgAdmin III is part of the install and can be found on the Windows Start menu: > All Programs > PostgreSQL x.x. If PostgreSQL is not installed further details on how to download and install can be found in the installation guide.

Also note that although PostgreSQL is not case sensitive it is best to use lower case for schema, table and columns names. Any of these names that created in either upper or mixed case will need to be within “double quotes” within any queries.

2.1.1 Creating a table to store the data

The first step is to create a table to store the data. This is done using the SQL CREATE TABLE command by following these steps;

1. From the Windows Start menu open pgAdmin III.
2. Connect to your database and select the schema you want to use to store the data (the default will be the public schema).
3. Click on the schema name and then click on the SQL query tool to open the query window.
4 In the SQL Editor, enter the following SQL statement. This will create an empty table, called postal in the default (public) schema. The syntax for this is: CREATE TABLE schema.tablename (column_name1 type (width), column_name2 type (width), ……). Therefore, if you want to create the table in a different schema use: CREATE TABLE 'myschema'.‘mytablename’.

CREATE TABLE postal
( uprn bigint NOT NULL,
oos_address_toid character varying(20),
rm_udprn bigint,
organisation_name character varying(60) DEFAULT NULL::character varying,
department_name character varying(60) DEFAULT NULL::character varying,
NULL::character varying,
sub_building_name character varying(60) DEFAULT NULL::character varying,
building_number integer,
dependent_thoroughfare_name character varying(80) DEFAULT NULL::character varying,
thoroughfare_name character varying(80) DEFAULT NULL::character varying,
post_town character varying(30) DEFAULT NULL::character varying,
double_dependent_locality character varying(35) DEFAULT NULL::character varying,
dependent_locality character varying(35) DEFAULT NULL::character varying,
postcode character varying(8) DEFAULT NULL::character varying,
postcode_type character varying(5) DEFAULT NULL::character varying,
x_coordinate numeric(10,3),
y_coordinate numeric(10,3),
rpc integer,
change_type character(1),
start_date date,
end_date date,
last_update_date date,
entry_date date,
"class" character(1),
process_date date)

2.1.2 Loading data

Once you have created the table you are then ready to load data into the table. This can be done either using the SQL COPY command, for which you will need administrator rights to your database or /copy from the pqsql prompt.

To load CSV data into the table created in the previous Steps use the COPY command in the SQL window enter the following: COPY postal FROM 'file_path/filename.txt' DELIMITER ',' CSV HEADER

Where file_path is the full directory path to where the data is stored and filename the name of the CSV file.

The SQL UPDATE command can also be used to load data to a table but for bulk loading is slower.
2.1.3 Converting coordinates to geometry

Although the postal table loaded in the example contains columns of x and y coordinates, to use these spatially these fields would need plotting using the built in functionality of your chosen GIS. However, it is possible to convert these two columns into a single geometry column for use in spatial queries either within the PostgreSQL database or a GIS.

To do this you will need to run further SQL commands on the data, including using the functions that the PostGIS extension adds to the PostgreSQL database. These are described in the following steps:

1. Firstly, add a column to store the geometry; this can be done by entering a query into the SQL window

   SELECT AddGeometryColumn (‘schema_name’, ‘table_name’, ‘geom’, 27700, ‘POINT’, 2)

In the example above ‘geom’ is the name of the column, 27700 is the spatial reference (SRID) the data type is point containing a pair of coordinates (3 here would indicate that the data contained Z (height) values). This command will also add the relevant metadata to the geometry_columns table in PostGIS. This table is created automatically by PostGIS and is used by some GIS to retrieve the details of spatial data stored in the database.

The ‘AddGeometryColumn’ function can also be used to add both point and polygon geometries to a database.

2. Once the geometry column is added to the data it will need to be populated with the spatial information. In the case of the AddressBase data in the given example, this is contained within the columns x_coordinate and y_coordinate. The SQL UPDATE command is used for this by entering the following in the SQL Window.

   UPDATE schema.tablename SET geom = ST_GeomFromText('POINT(' || x_coordinate || ' ' || y_coordinate || ')', 27700 )

In this command the column geom created by AddGeometryColumn is populated from the columns containing the x and y values as points, using the SRID 27700 (British National Grid).

3. The final step is to add a Spatial Index which is used to help improve performance when querying the data. In the SQL window use the following syntax

   CREATE INDEX idx_blpu_geom ON schema.tablename USING gist(geom)

Indexes may also be added to other fields which are queried frequently however adding indexes will increase the size of the database. However adding too many indexes may have an adverse effect on performance.

2.2 Loading Shapefiles into PostGIS

ESRI Shapefiles can be loaded directly into PostGIS either from the PSQL command prompt or using the PostGIS Shapefile loader within pgAdmin III. This example uses the loader as it is the easier method.

1. From Windows Start menu, select pgAdmin III.

2. Navigate using the Object Browser to the database into which the shapefiles are to be loaded. If necessary create a schema in which to store the tables.

3. From the pgAdmin III menu bar select Plugins.
4. On the dropdown menu select PostGIS Shapefile and DBF loader 2.0.

5. Check the database connection by clicking View connection details.

6. This will open a new window displaying the details of the current open connection.

If these are ok click OK, the connection details will appear in the Log window. If successful, a message, 'Connection succeeded' will appear.

7. To load a Shapefile click on Add File.

8. The Select a Shape File dialog will open.
9 In the ‘Places’ column navigate to the location of your Shapefile and select file(s) from the list.
10 Click **Open**, the data will now appear in the Import List.
11 Before importing the data, you will need to check that the details are correct.

<table>
<thead>
<tr>
<th>Shapefile</th>
<th>Schema</th>
<th>Table</th>
<th>Geo Column</th>
<th>SRID</th>
<th>Mode</th>
<th>Rm</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>constine</td>
<td>public</td>
<td>constine</td>
<td>geom</td>
<td>0</td>
<td>Create</td>
</tr>
</tbody>
</table>

12 Highlight the row to edit the details and then double click on the entry to be edited. In the above example the data will be loaded into the public schema and be given the default table name. However the SRID is set to 0 and will need to be changed to 27700 (British National Grid).
13 Click **Import** to load the data. When finished a message will appear in the Log window: **Shapefile import completed**. Click **Cancel** to close the window.
14 Look in the list of tables, the added data will now appear in this list. Under the public schema you will also see the geometry_columns table. If this table is opened, it can be seen that the new table has been added.

### 3 Working with ArcGIS

Before trying to connect to PostgreSQL from ArcGIS you will need to check that you have compatible versions of the software. Further information on which versions are supported can be found on the ArcGIS Resources website: http://resources.arcgis.com/en/home/

From version 10.0, direct access to PostgreSQL without using ArcSDE is possible. However, to access the database using older versions you will need ArcSDE. These notes use ArcEditor 10.0, and some knowledge of ESRI software is also assumed.

#### 3.1 Adding a Query Layer from PostgreSQL

1 Start ArcMap.
2 From the **File** menu select **Add Data > Add Query Layer**.
3 The **New Query Layer** dialog will open.

![New Query Layer dialog](image)

4 The first step is to set up a connection to the database, click on the **Connection** button.

5 The **Manage Connection(s)** dialog will open; click on **New** to set up a new connection.

![Manage Connection(s) dialog](image)

6 In the next dialog box enter your connection details. These should be similar to those illustrated below. However, if you are unsure they can be found under the server properties in pgAdmin III.

![Database Connection dialog](image)

7 To check they are correct click **Test Connection** and, if correct, click **OK**.

8 Your new connection will appear in **Manage Connection(s)**, as this is the only connection at the moment. It is already selected so click **OK**.

9 The list of tables will now be populated in the **New Query Layer** dialog. Clicking on a layer will select it and display columns.
10 To load a layer you will need to give it a name and create an SQL query. In the example shown, the ‘osdata.strategi.coastline’ has been selected.

![New Query Layer](image)

11 In the **Name** box, enter the name for the layer in ArcGIS and enter a query. In the example shown all of the data is being returned so ‘select * from osdata.strategi.coastline’ is entered. Click **Validate** to test the query and then **Finish**. The data will now appear in ArcMap.

![Map](image)

### 3.2 Adding layers using a database connection

From ArcGIS version 10.1 it is possible to load data directly from a database using a database connection as described in the following steps;

1 In ArcMap open the **Catalog** window

![Catalog](image)
2 In the Catalog window select **Database Connections**.

3 Double click **Add OLE DB Connection**.

4 In the **Data Link Properties** select **Use data source name** and **PostgreSQL30** from the drop-down list.

5 Enter the relevant user name and password and select the database with which you wish to connect from the drop down list.

6 This will add the connection to the list in the Catalog window. Double-click on the connection to see a list of available layers and then drag and drop them into your ArcMap session.

### 4 Working with MapInfo

It has been possible to work with PostgreSQL in MapInfo since version 10.0. However, MapInfo does not currently support versions of PostGIS higher than 1.5. Furthermore, the connection requires a current ODBC driver to have been installed on your local machine. These notes assume that this has been set up. For further information, please contact your computer support or system administrator. It is also recommended that the bytea_output entry in the PostgreSQL.conf file is set to 'escape' (default value is hex). Further details on editing the PostgreSQL.conf file can be found in the guide to installing PostGIS.

#### 4.1 Setting up a MapCatalog

To access spatial data held in PostgreSQL with MapInfo you will need to add a MapCatalog to your database. This is a metadata table similar to the PostGIS geometry columns table containing information about each spatial table such as layer name, projection and style details. A MapCatalog can be created using the **EasyLoader** tool which can be found in MapInfo’s **Tool Manager**.

The first step is to load **EasyLoader**.

1 In MapInfo select the **Tools** menu and **Tool Manager**.

2 Select **EasyLoader** and check **Loaded** and **Autoload** (if you want the tool loaded automatically when you open MapInfo).

3 Click **OK**.
This will add **EasyLoader** to the **Tools** menu. To create the MapCatalog follow these steps:

1. From the Tools menu in MapInfo select **EasyLoader**.

   ![EasyLoader Window](image1)

   2. Click **ODBC** and select your connection

   3. Click the **Map Catalog** button. This will create the catalog. Once completed the message: ‘A Map Catalog was successfully created’ will appear.

   ![Map Catalog](image2)

   In your database you will now see a new schema MapInfo which contains one table, `mapinfo_mapcatalog`. If this table is opened, it will be seen that it will not contain any records. This will be updated as data is added from PostgreSQL into MapInfo.
4.2 Connecting to PostgreSQL

4.2.1 Opening a connection

The first step in connecting from MapInfo to PostgreSQL is to open a database connection.

1. From the File menu select Open DBMS Connection.
2. In the dialog that opens select ODBC and click New.

3. In the Select Data Source dialog click on the Machine Data Source tab and select the PostgreSQL data source. Then click OK.

You have now opened a connection to the database which can be managed using the DBMS toolbar.

To check you have an open connection the last icon in the toolbar will appear, clicking on this icon will close the database connection.

4.2.2 Making a table mappable

The next step is to make the table in PostgreSQL mappable; this will add the details of the table created earlier into the Map Catalog.

1. To do this click on the Make Table Mappable icon on the DBMS toolbar.

2. The Select DBMS Table dialog box will now open.
3. In the dropdown menu next to Schema chose the database schema where the spatial tables are stored.

![Image of Select DBMS Table dialog]

4. Below this a list of tables will be seen. Click on one and select Open.

5. A new dialog opens: Make Table Mappable.

![Image of Make Table Mappable dialog]

In the Index Type drop-down, select ‘PostGIS’ (this will only work if your table has an entry in the geometry_columns table). In the Index Column drop-down, select the column containing the geometry, in this case geom, this will be the column to which a spatial index will be applied.

6. The next step is to set the object type and the styling; the style definition will be stored in the map catalog and can be edited later if required.

7. Select the appropriate Object Type for your layer (Symbol, Line, Region).

8. Next select the Object Style by clicking on the appropriate button. The style window will open. Select your chosen style and colour and click OK. Uncheck Per Row Style (leaving this checked will style the objects in the layer using a column in the data).

9. Click OK. Once the table has been made mappable the following dialog will appear, click OK to close this box.
4.2.3 Opening the table

The table is now ready to be opened in MapInfo using the following procedure;

1. Click on **Open DBMS Table** on the DBMS toolbar.
2. In the dialog click on the table which has made mappable (if the table is not mappable it will open as a table browser) and click **Open**.
3. Select **Standard Mode** and **Download Data (Linked Table)**. For details of the other options please use the MapInfo help documentation.
4. Under MapInfo TAB file location, click on the icon and select a location on your local drive to store the TAB file and click **OK**. The layer will now open in a Map window.

Once a table is mappable, a TAB file is created to load on subsequent occasions, providing the database connection is open. Simply open the **TAB** from the **File > Open** menu.

4.2.4 Changing the style of a table.

Once a table has been made mappable it is possible to change the style using the DBMS toolbar.

1. On the DBMS toolbar click on the **Change the Symbol for a mappable DBMS table** icon.
2. In the next window select the table and click **OK**.
3. The **Change Table Object Style** dialog will open.
4. Click on the object type (Symbol, Line or Region).
5. A style dialog will open. Select the desired style and click **OK** on both dialogs to close them.
6. If the table is already open you will need to click on the **Refresh DBMS Table** icon on the DBMS toolbar to refresh the Map window.

4.3 Saving tab files into PostgreSQL

A TAB file opened in MapInfo can be saved into a PostgreSQL database by using **Save Copy As** on the **File** menu within MapInfo.

1. From the **File** menu select **Save Copy As**.
2. In the dialog select the table to be saved to PostgreSQL.
3 Click **Save As**...

4 In the next dialog change Save as type: to PostgreSQL:database_name (where database name is the name of the required spatial database).

5 Select **Schema** from the drop-down menu and enter a name for your table and click **Save**.

6 The **Save As** dialog will then ask for a file name for the TAB file which will be stored locally. Enter a name and click **Save**.

If pgAdmin III is now opened, it will be seen that your table is now contained in the chosen database and entries for the table added in both the geometry_columns and mapinfo_mapcataolog tables.

## 5 Working with CadCorp SIS

The ability to connect to a PostGIS spatial database is part of the core functionality of CadCorp SIS. These notes give an overview of how to open a spatial table already stored with PostGIS and have been created using CadCorp SIS 7.1 Map Modeller.

1 From the Home tab click on **Add Overlay**.

2 Select **Database Overlay** from the drop-down menu.
3. In the **Spatial Database** dialog select the **PostGIS** icon and click **Next**.

![Spatial Database Dialog](image1)

4. In the next dialog you will need to enter the connection details for the PostgreSQL database which contains the spatial data you wish to load and then click **Next**.

![PostgreSQL Connection Dialog](image2)

5. In the next window select the layer you wish to load and click **Next**.

![PostGIS Layer Selection Dialog](image3)
6 Finally select the connection type as either Dynamic or Local (this will load information to your local drive) and enter details of any feature coding or text attribute display. After this, click Next.

7 Finally either the entire layer can be loaded or a SQL query can be used to bring back specific features. Once this is done click Finish. The layer will be loaded and can be used alongside the other layers in a map using standard functionality.

6 Working with QGIS

QGIS is an open source GIS which includes tools for connecting, managing and querying a PostGIS database. These notes are based on QGIS version 1.8.0-Lisboa, with the DB Manager plug-in loaded.

NOTE: In QGIS version 1.8 a file browser has been introduced for managing spatial data connections, however this will only works with PostGIS 2.1.

6.1 Connecting to PostGIS

To connect to a PostGIS database follow these steps:

1 In QGIS click on Add PostGIS Layers on the Layers toolbar (alternatively this can be accessed from the Layer menu).

2 In the Add PostGIS layers dialog click on New to set up a connection.

3 In the next dialog you will need to enter connection details, if you are unsure these can be found in the Object Browser of pgAdmin III.

   Name – a user defined name which will be used to identify the connection
   Host – either ‘localhost’ or the name of the server where your database is located
   Port – the default is 5432
   Database – the name of the database to be connected
   Username – the user name with which you connect to PostGIS
   Password – your password

4 This completed dialog will look something like the one below, clicking on Test Connect will verify the details are correct.
5 Once the details are completed click **OK** to close the dialog.

6 To access the data held in the database click on **Connect**. A list of available spatial layers will now appear similar to the example shown below. If you have data stored in multiple schemas you will need to click the plus next to your data to review the available layers.

7 To load a layer in to QGIS click on the layer and then click on **Add**, the layer will now appear in the QGIS **Layer** window and can be styled using the **Style Manager**.

### 6.2 Adding a subset of a layer

It is also possible to query a layer before loading to add a sub set of the data. The following steps describe how to do this:

1 In **QGIS** click on **Add PostGIS Layers** on the **Layers** toolbar.

2 From the **Connections** drop-down select the connection to the required database and click **Connect**.

3 The available layers will now appear listed in the blank area; select the schema and layer you wish to load.

4 Next click on the **Build query** button to open the **Query Builder**.

5 In the example only the M6 Motorway is to be loaded.

6 The fields containing information about the feature are listed on the left hand side of the dialog, in this example, the road number is contained in the field Number0.

7 To see the values contained in the field click on the **field name** and then click on **All** on the right-hand side beneath the **Values** area. (If your data contains a lot of values you may wish to click ‘Sample’ to bring back a shorter list).

8 The next step is to build your SQL where’ statement. To do this, double click on the field containing your values, for example, Number0.

9 Next select your operator, in this example we wish to use: `. Double-click the value from the **Values list**. The dialog should now be populated similar to that below.
10 Click on **Test** to check that the query is correct. If it does not return an error you can then click **OK** to add the query to the Layer.

11 The data has not loaded at this point. However if one scrolls across to the right in the **Add PostGIS Layers** dialog the query details can be seen under the SQL column.

12 Click on **Add to load** the layer into QGIS.

13 The image below shows the result of the example query with the extent of the M6 Motorway taken from Strategi data.
6.3 Using DB Manager

If the DB Manager plug-in has not been loaded, it can be added to QGIS from the Plugins menu and Fetch Python Plugins (see the guide to QGIS for more details). This tool can be used to manage your connection to PostGIS from QGIS, load layers into QGIS and perform spatial queries. The advantage of using this over the previous method is that it allows more complex spatial queries to be built. For example, you may have a layer of habitat data and wish to identify all the nesting sites that are within a specified distance from a specific type habitat within the layer.

Before using the tool you must have set up a connection to your PostGIS database using the connection tool described previously. DB Manager is accessed from the Database menu by selecting DB Manager. The tool will open with a list of available connections in the right hand side. Clicking on the +/- next to each entry will reveal the details of available schemas and tables for each connection.

1. From the Schema menu a schema within the connected database can be added or deleted. Tables can also be created, edited or deleted from the Table menu.

2. To load a layer into QGIS, simply click on the layer name and drag and drop it in to the list of layers within QGIS. Unlike the connection tool, you can then work with the layer in QGIS with DB Manager still open.

3. A layer can also be previewed within the tool. To preview a layers table click on the layer and then select the Table tab in the left hand window, as shown below.
4 To view the layer’s geometry, click the **Preview** tab.

![Image](image.png)

### 6.3.1 Performing SQL queries with DB manager

The DB Manager can also be used to perform complex spatial queries on data in the same way as can be carried out using pgAdmin III. The advantage using the DB manager is that the results can be directly ported into QGIS. In the following example we wish to define all Great Britain railway stations that are within 1 km of a motorway.

1 In DB manager click on the **SQL Window** icon to open the SQL window.

![SQL Window](image.png)

This window is comprised of two panes. These are the SQL Query pane where the query is written and the Results pane where a tabular view of the results, obtained by running the query, are displayed.

2 Next, enter the query details into the SQL **Query** pane using an SQL select statement.

3 First we want all the railway stations obtained from the Strategi® railway points layer. These can be identified using the legend field. The initial SQL statement will look like this:

   ```sql
   SELECT * FROM strategi.railway_point WHERE station.legend = 'Railway Station';
   ```

4 However by using the wildcard (*) this will return all the data in the table. Furthermore by adding an alias for the table name we do not have to enter the full schema and table name details. All we need to return is the name of the station, its unique identifier and its geometry so it can be added to the map later. The statement can be amended to this:

   ```sql
   SELECT station.geom, station.gid, station.name FROM strategi.railway_point as station
   WHERE station.legend = 'Railway Station';
   ```
5 Next we need to add the Strategi motorway layer and use the ST_DWithin function to identify our stations within 1 km of the motorways. This is the equivalent to creating a buffer around each feature and identifying the stations which fall within it.

The SQL statement will now look like this:

```sql
SELECT DISTINCT ON (station.gid) station.geom, station.gid, station.name
FROM strategi.railway_point AS station, strategi.motorway AS mway
WHERE ST_DWithin (mway.geom, station.geom, 1000.00)
AND (station.legend = 'Railway Station') ;
```

The DISTINCT key word has also been added on the station id to remove any duplicate records and an alias used for the motorways layer. In the SQL Window it will look like this:

6 To run the query click **Execute**.

7 The results will appear in the **Results** pane.

8 To load the result into QGIS check the box **Load as new layer**. The following details will appear:

9 Enter the details for the column of unique values and the layer geometry, in the example these are ‘gid’ and ‘geom’.

10 Give the layer a name which can be used to identify it in QGIS and click on **Load now**!
11 The results of the query are now displayed in QGIS.

7 User Examples of using PostGIS to perform spatial queries

7.1 NHS Bradford and Airedale

Task – to identify which postcodes fall within each GP Practice boundary and establish gaps in service.

Benefit – ensuring continuity and quality of patient care across the Bradford and Airedale area.

Within the Bradford and Airedale area there are approximately 12,000 postcodes which had to be tested against 87 practice boundaries. By using PostGIS each postcode code is tested against each GP area creating a comma separated list for each practice boundary. In addition postcode areas with a low number of GP practices were identified to ensure suitable GP coverage across the area.

The following SQL query was used in PostGIS for the analysis:

```sql
SELECT postcode_1, array_to_string (ARRAY(SELECT trim(both from practice_code)
FROM "NHS_Data".vw_practice_qof_gte_900
WHERE st_within("OS_Data".cp_nhs_postcode_0612.the_geom ,
"NHS_Data".vw_practice_qof_gte_900.wkb_geometry)), ',') as Q900_practice_codes,
(SELECT count(practice_code)
FROM "NHS_Data".vw_practice_qof_gte_900
WHERE st_within("OS_Data".cp_nhs_postcode_0612.the_geom ,
"NHS_Data".vw_practice_qof_gte_900.wkb_geometry)) as Num_practices_with_Qof_GTE900,
the_geom
FROM "OS_Data".cp_nhs_postcode_0612
WHERE cp_nhs_postcode_0612.district_name::text ~~ 'Bradford%':text
```

The 'array to string' function converts an array into a separated list in this case "," so this gets inserted as a column of data, the end product is a view with the following format:
Postcode, Practice_code(s), number of practices boundaries that the postcode point is within area.

```sql
SELECT *
INTO "NHS_Data".nhs_postcodes_with_0_practices_Q900
FROM "NHS_Data".vw_Postcodes_with_practices_qof_gte_900
WHERE Num_practices_with_Qof_GTE900 = 0

SELECT *
INTO "NHS_Data".nhs_postcodes_with_1_practices_Q900
FROM "NHS_Data".vw_Postcodes_with_practices_qof_gte_900
WHERE Num_practices_with_Qof_GTE900 <=1

SELECT *
INTO "NHS_Data".nhs_postcodes_with_2_practices_Q900
FROM "NHS_Data".vw_Postcodes_with_practices_qof_gte_900
WHERE Num_practices_with_Qof_GTE900 <=2
```

The results were displayed using MapInfo via an ODBC link to the PostGIS database as shown below.
7.2 Quality Observatory South East Coast

Task – the identification of three other nearest GP practices to each GP Practice within the South East Coast Strategic Health Authority.

Benefit – ensuring continuity of practice provision across the area. The following example uses PostGIS for a nearest neighbour search taking advantage of PostgreSQL unique array offerings. The data held in the table ‘public.Q37_GPP_Immform’ which is stored in PostGIS has been geocoded using Ordnance Survey’s Code-Point® dataset. This dataset lists GP practices within South East Coast SHA with the following fields:

Gid, GPP_CODE, GPP_NAME, POSTCODE, SHA11CD, SHA11CDO, SHA_NAME, PCO11CD, PCO11CDO, PCO_NAME, CCG_CODE, CCG_NAME, PCT_CLUSTE, the_geom

The query finds the nearest neighbouring practices (max set to 3) given a geometry and n geometries of data. Contained in the query is a WHERE clause which uses a PostGIS function ST_DWithin() search. In this particular example the ST_DWithin function utilises indexes to limit the search list to distance of one mile (approximately 1609 metres) from a given geometry.

```
SELECT a."GPP_CODE", a."GPP_NAME", a.nn_gid[2] As nn_1,
       a.nn_gid[3] As nn_2,
FROM
  (SELECT b."GPP_CODE", b."GPP_NAME",b.the_geom,
   ARRAY(SELECT c."GPP_CODE"
   FROM public."Q37_GPP_Immform" c
   ORDER BY ST_Distance(b.the_geom, c.the_geom)
   LIMIT 4) As nn_gid
   FROM public."Q37_GPP_Immform" b ) a
ORDER BY a."GPP_CODE"
```

ST_Distance – for geometry type returns the two-dimensional cartesian minimum distance (based on spatial ref) between two geometries in projected units. For geography type defaults to return spheroidal minimum distance between two geographies in metres.

ST_DWithin – returns true if the geometries are within the specified distance of one another.

The result is a cross tab like query that lists the three nearest neighbouring GP practice codes for each GP practice in one row with the neighbouring practices listed as nn_1, nn_2, nn_3.
The map below shows the three nearest neighbouring practices to G81001 as returned in row one of the query above.

Quantum GIS has ability to view PostGIS data directly and do simple filters on it. In the example below, the LAT (Local Area Team) field of CCGBoundaries table is used to filter for only CCG boundaries within either LAT Q67 (Kent and Medway) and Q68 (Surrey and Sussex).