Demonstration of temporal data management with PostgreSQL

Workshop for Global Geospatial Conference 2013 http://www.gsdi.org

by Dr. David Pullar,
University of Queensland, Australia d.pullar@uq.edu.au

Geoscience Project funded through
International Mining for Development Centre http://im4dc.org/

Introduction
Getting started – Using Portable GIS we create a PostgreSQL geodatabase, load data and view in GIS
Historical permit tracking – Add temporal tracking capability to PostgreSQL and illustrate queries
Viewing history of permits in Google Earth – Developing a web service to serve data

Introduction

A permit is simply what it means; it is consent to do something. Permits may be issued to persons or organisations for any land administrative activity, including mining or undertaking land uses that have potential environmental impacts. They should be subject to licensing controls that list their rights for doing the activity and what restrictions and responsibilities they need to adhere to. Licensing should be applied uniformly for all permits over the landscape, so it requires a land administrative system to manage this. Ideally we would like to build a permit system following well established principles and standards. There are standards and software support in GIS for temporal data management, but there are no widely recognised standards for land permit systems. Looking further afield, a standard (ISO 19152) and software have been developed for Land Administration Domain Model or LADM (Lemmens et al., 2013). While LADM is predominantly focussed on facilitating land ownership registration and cadastral data for developing countries, it also encompasses broad aspects of land rights and interests. In this workshop we will adopt a simplified view of the data models and standards from LADM to implement a land permit system.

The motivation for the workshop is to show that free and open source spatial software is available for setting up a permit system in a database. Permits require a special database to represent the spatial and temporal aspect of a permit and it lifespan. Using PostgreSQL, as a powerful open database server we show how to implement a system for managing permit and queries to support applications. This could be applied to any land resource managed over time; but we give an example based on mining permits. Both the PostgreSQL database and GIS software may be freely downloaded.

A brief explanations of permits

A permit is based upon a simple concept for a permitted activity. To support mineral exploration and mining activities require up-front descriptions of the rights and terms. Figure 1 shows the core information recorded in a LADM data model for permits. At its core it records the rights, responsibilities and restrictions affecting land. For mining rights there are two broad types of rights: i) exploration, and ii) mining permits (*rightType* attribute). Corresponding with rights there are two administrative types for: i) initial application, and ii) granted right (*unitType* attribute), each with its spatial definition (*geometry* attribute) and times for the terms specification (*timeFrom* and *timeDuration* attributes). A permit is subject to changes over its lifetime which are all recorded. A current permit with the up-to-date details is recorded in the permit table. As the type of permit changes through stages of application to granting the historical details of past states are recorded in the permit history table. Types of changes may include codes for apply, grant, change owner, renew, and eventually expire (*adminType* attribute). While not included in this example there would be a reference to document for these changes and times recorded in the permit table for time a change was made to the current state (*timeStamp* attribute) and the time interval of past states (*beginTimeStamp* and *endTimeStamp* attribute).

More information on LADM may be found in (Lemmens et al., 2013) and applying LADM for permits in (Pullar, 2015). The focus of this workshop is on temporal and database management aspects of permits. It is important to keep a complete history of transactions on permits for a number of legal and management reasons. We use concepts for temporal data management (Kulkarni and Jan-Eike, 2012*)* and follow similar methods for implementation in PostgreSQL as described in webpage https://wiki.postgresql.org/wiki/SQL2011Temporal

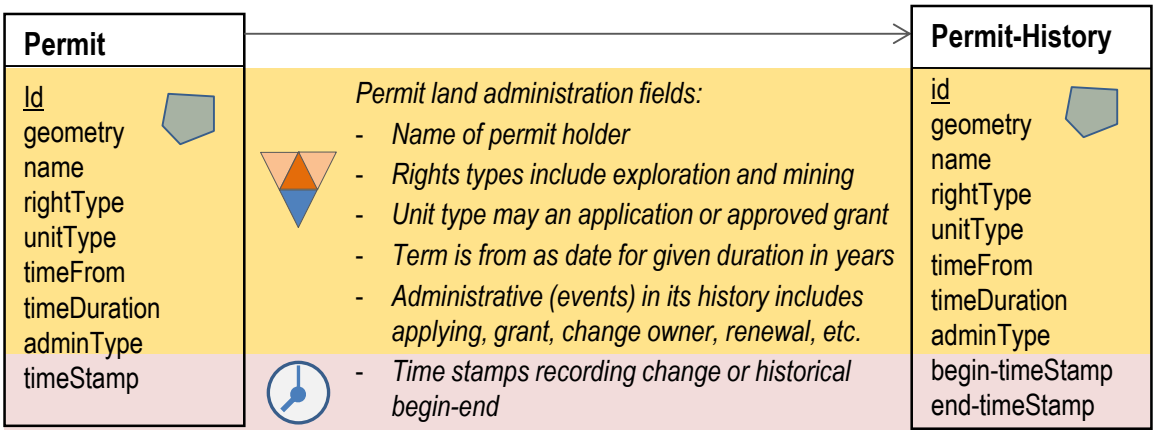


Figure 1. Example land administration fields for a mineral exploration permit table and its historical table. It includes core fields from the LADM for land rights ▽ and transactional time fields 🕐.

Figure 2 illustrates a processing workflow for the lifespan of a permit. It shows the main administrative events for new application, granting, a change in owner, renewal and final expiration. The data in the permit and history tables is shown for a snapshot in time for a renewal. The renewal causes changes in details for the terms (time) specification, the spatial extent of permit as land is relinquished in the process, and the time stamps of these changes in permit and history tables.

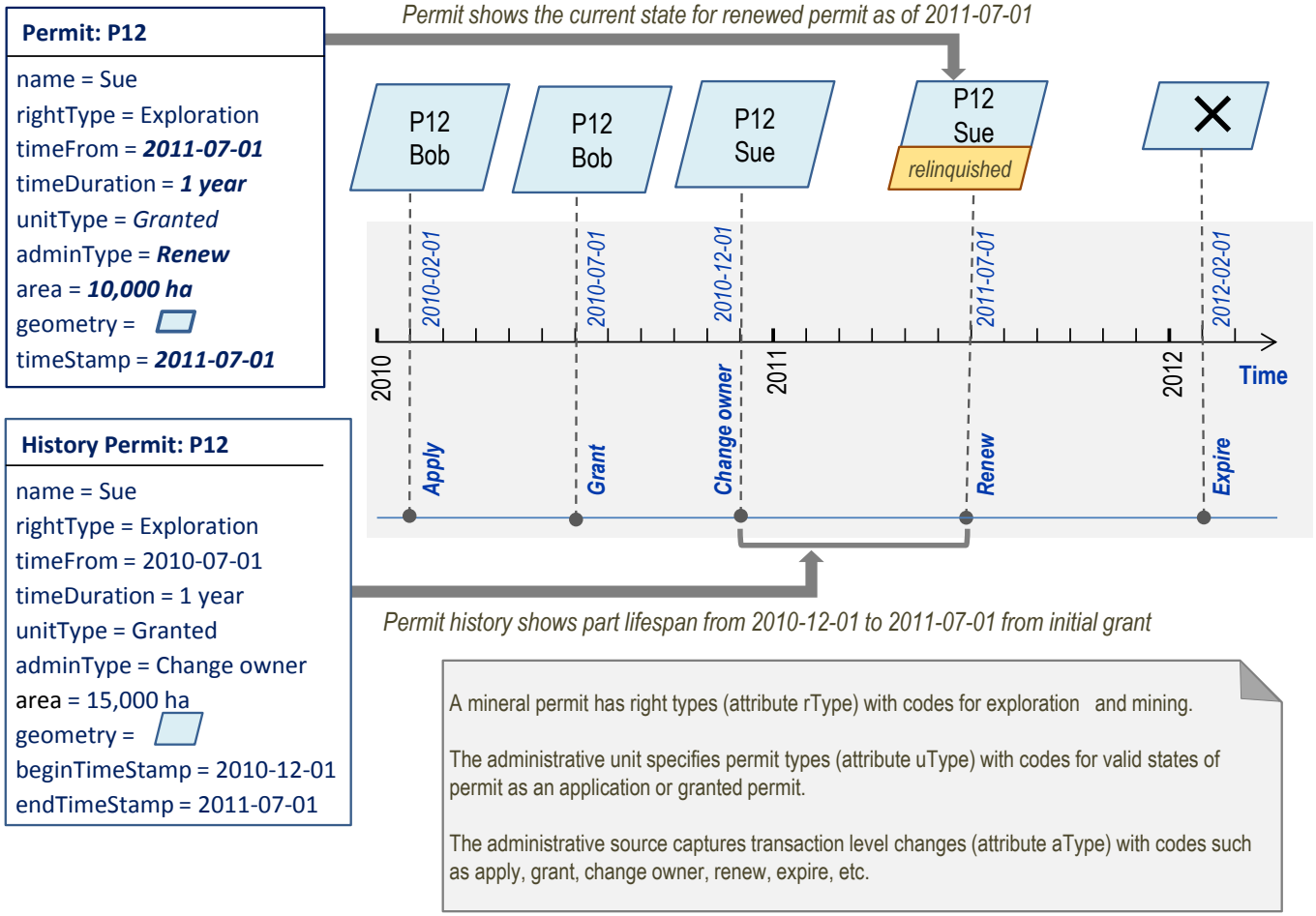


Figure 2. Example timeline for one permit with entries in permit table and history table for one period.

Getting started

1. Using *Portable GIS*

This tutorial uses the open source PostgreSQL relational database (www.postgresql.org) and QGIS software (www.qgis.org) that has been packaged to run in the windows environment without any need for installation. http://archaeogeek.com/blog/2014/09/25/new-portable-gis-releases/

Go to above website and download portablegis_setup_v5.exe; in this workshop it is available on a USB but it may be saved to a local folder.

Start d:*portablegis.exe*. You see a panel to start individual software.

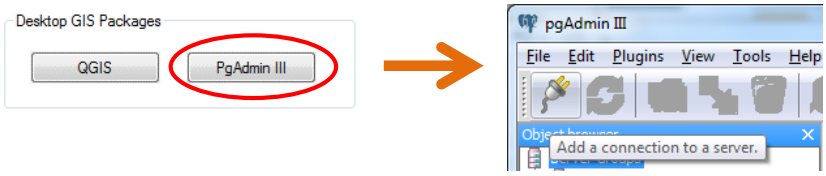
a) Start Desktop Module – QGIS
b) Start Server Module - PostgreSQL Database Server
c) Start Desktop Module – PgAdmin III

We use a small example for mining exploration permits for workshop. GIS datasets are provided for base data. We start by creating a new PostgreSQL database and import the GIS datasets.

PostgreSQL - like many databases - works as a server in a client-server arrangement. The client makes a request to the server and gets back a response. In this workshop the application pgAdmin or QGIS are the client which sends requests to PostgreSQL\PostGIS in the SQL language and the response is usually a table of data. PostGIS is an add-on to PostgreSQL that provides the spatial capabilities. I recommend finding more information on PostreSQL\PostGIS from the OSGeo Live Quickstart Tutorial live.osgeo.org/en/quickstart/postgis_quickstart.html

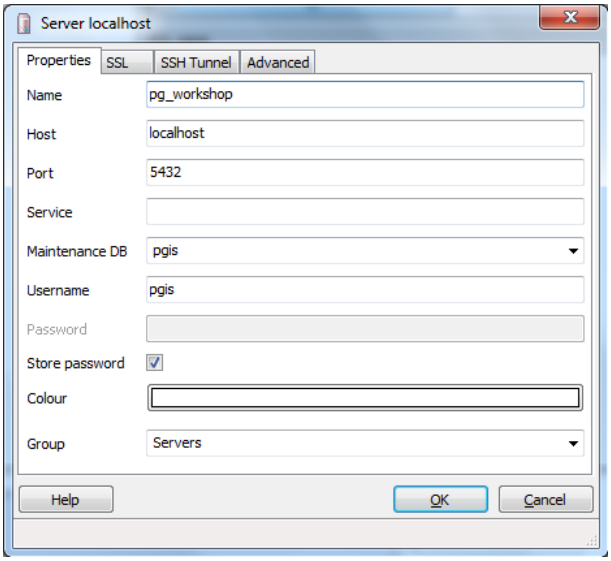
2. Start pgAdmin

   From the *Portable GIS Control Panel* click tab *Desktop Modules*, and click on *PgAdmin III*.



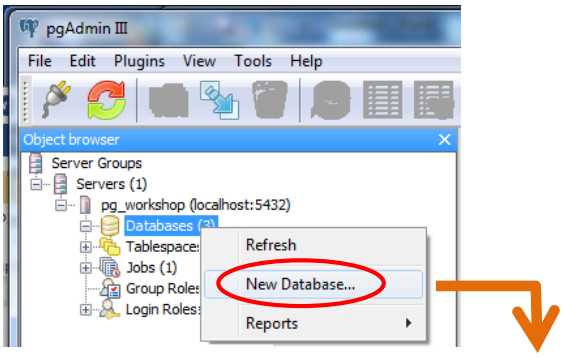
In pgAdmin click menu button to create a connection  to the PostreSQL\PostGIS server. Enter name and properties for the connection as shwn below. Note that after initially creating the connection, you only need to supply the username and password to re-connect  .

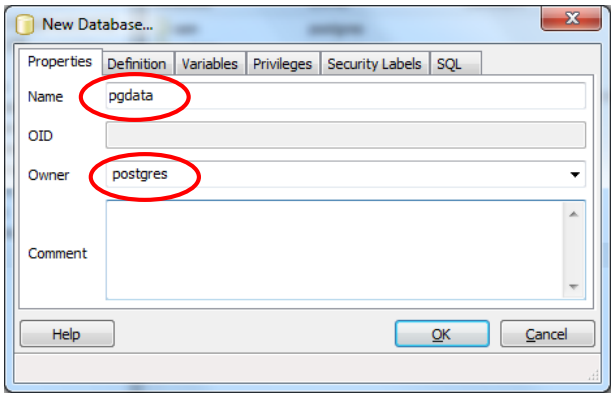
Here are some suggested values.
Name: *pg_workshop*
Host: *localhost*
Port: *5432*
Username: *pgis*
Password: *pgis*

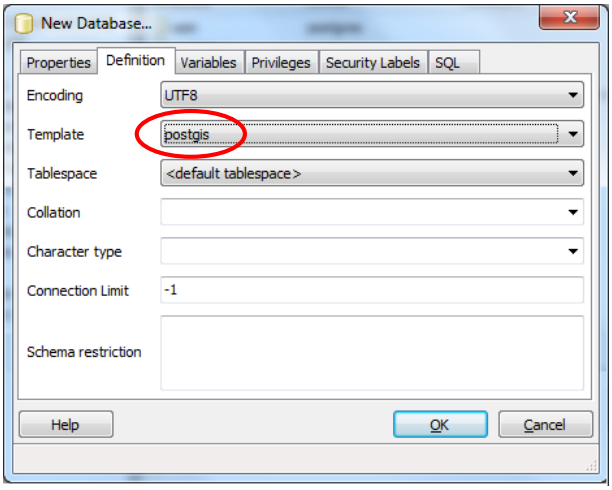


3. *With PgAdmin create a new database*

Right click on Databases, and selection *New Database*. We name it *pgdata* and identify owner as *pgis*. We also need to associate the database with PostGIS; do from the Definition tab by selecting the template *postgis*.







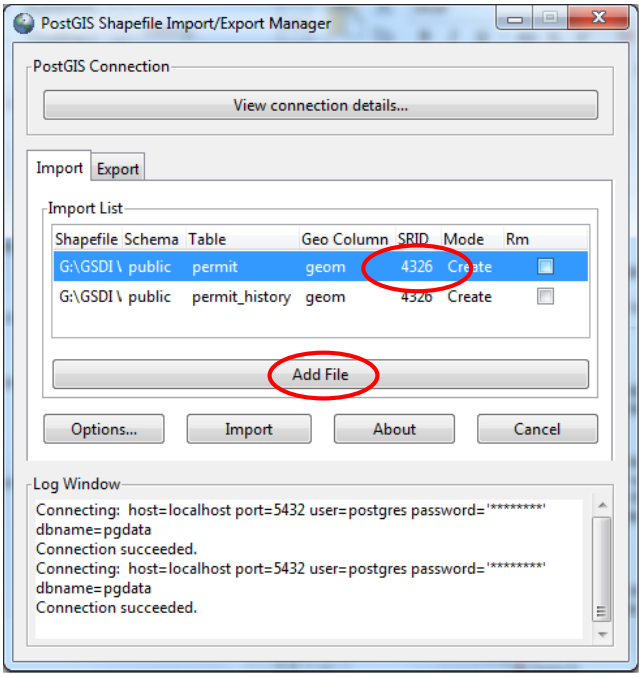
4. *With PgAdmin load GIS datasets*

The connection information is set to be similar parameters as used in previous steps for database *pgdata*.

Click Add File button to add provided workshop shapefiles *permit.shp* and *permit_history.shp* in folder *d:/usbgis/gsdidata*.

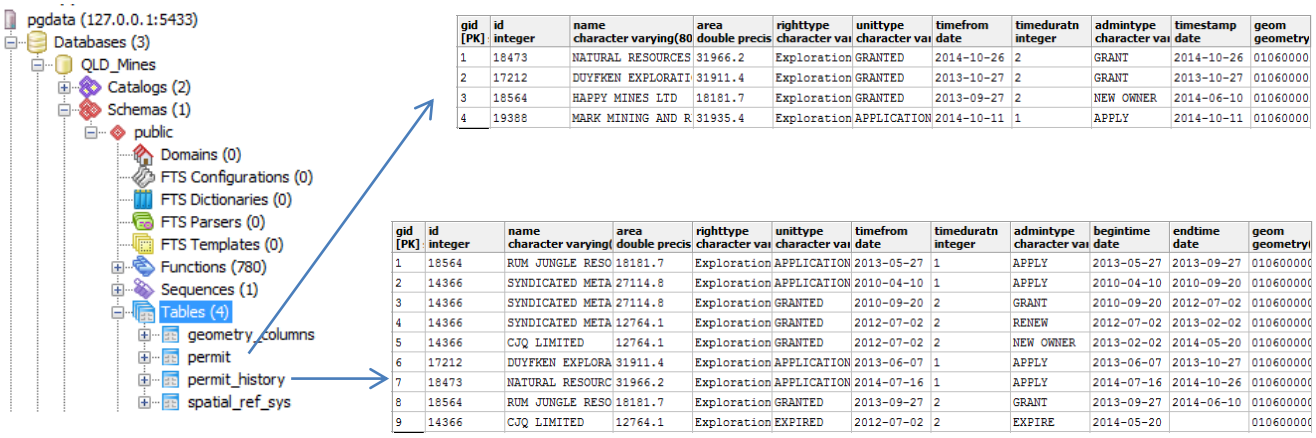
For each set the spatial reference to WGS84 (SRID: 4326), and the outout geomtry column is named *geom*.

Once both *permit.shp* and *permit_history.shp* are added to import list, click *Import*.



5. *View imported data in PGAdmin browser*

   Right click on Databases again and select *Refresh*. Now you should be able to see the tables in the contents tree for pgAdmin, if you right click on the tables you can *View Data*.



gid [PK]	id integer	name character varying(80	area double precis	righttype character var	unittype character var	timefrom date	timeduratn integer	admintype character var	timestamp date	geom geometry
1	18473	NATURAL RESOURCES	31966.2	Exploration	GRANTED	2014-10-26	2	GRANT	2014-10-26	01060000
2	17212	DUYFKEN EXPLORATI	31911.4	Exploration	GRANTED	2013-10-27	2	GRANT	2013-10-27	01060000
3	18564	HAPPY MINES LTD	18181.7	Exploration	GRANTED	2013-09-27	2	NEW OWNER	2014-06-10	01060000
4	19388	MARK MINING AND R	31935.4	Exploration	APPLICATION	2014-10-11	1	APPLY	2014-10-11	01060000

gid [PK]	id integer	name character varying(	area double precis	righttype character var	unittype character var	timefrom date	timeduratn integer	admintype character var	begintime date	endtime date	geom geometry
1	18564	RUM JUNGLE RESO	18181.7	Exploration	APPLICATION	2013-05-27	1	APPLY	2013-05-27	2013-09-27	010600000
2	14366	SYNDICATED META	27114.8	Exploration	APPLICATION	2010-04-10	1	APPLY	2010-04-10	2010-09-20	010600000
3	14366	SYNDICATED META	27114.8	Exploration	GRANTED	2010-09-20	2	GRANT	2010-09-20	2012-07-02	010600000
4	14366	SYNDICATED META	12764.1	Exploration	GRANTED	2012-07-02	2	RENEW	2012-07-02	2013-02-02	010600000
5	14366	CJQ LIMITED	12764.1	Exploration	GRANTED	2012-07-02	2	NEW OWNER	2013-02-02	2014-05-20	010600000
6	17212	DUYFKEN EXPLORA	31911.4	Exploration	APPLICATION	2013-06-07	1	APPLY	2013-06-07	2013-10-27	010600000
7	18473	NATURAL RESOURC	31966.2	Exploration	APPLICATION	2014-07-16	1	APPLY	2014-07-16	2014-10-26	010600000
8	18564	RUM JUNGLE RESO	18181.7	Exploration	GRANTED	2013-09-27	2	GRANT	2013-09-27	2014-06-10	010600000
9	14366	CJQ LIMITED	12764.1	Exploration	EXPIRED	2012-07-02	2	EXPIRE	2014-05-20		010600000

6. *View data in PostgreSQL with a SQL*

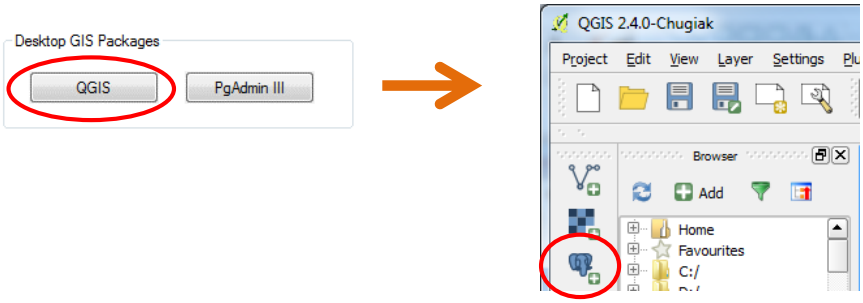
   I assume you are familiar with SQL. You may run SQL in PgAdmin by clicking on the SQL button 

. Try it out by entering a simple query in the editor and view output pane, i.e. enter *SELECT * FROM permit* and click execute  .

We will return to PgAdmin, but next we look at the GIS datasets in QGIS. QGIS is a free and open source GIS; we use it in this workshop for viewing and analysing permit data. The workshop assumes familiarity with GIS. Further instructions on using QGIS are available on website: www.qgis.org/en/site/forusers/trainingmaterial

7. *Start QGIS*

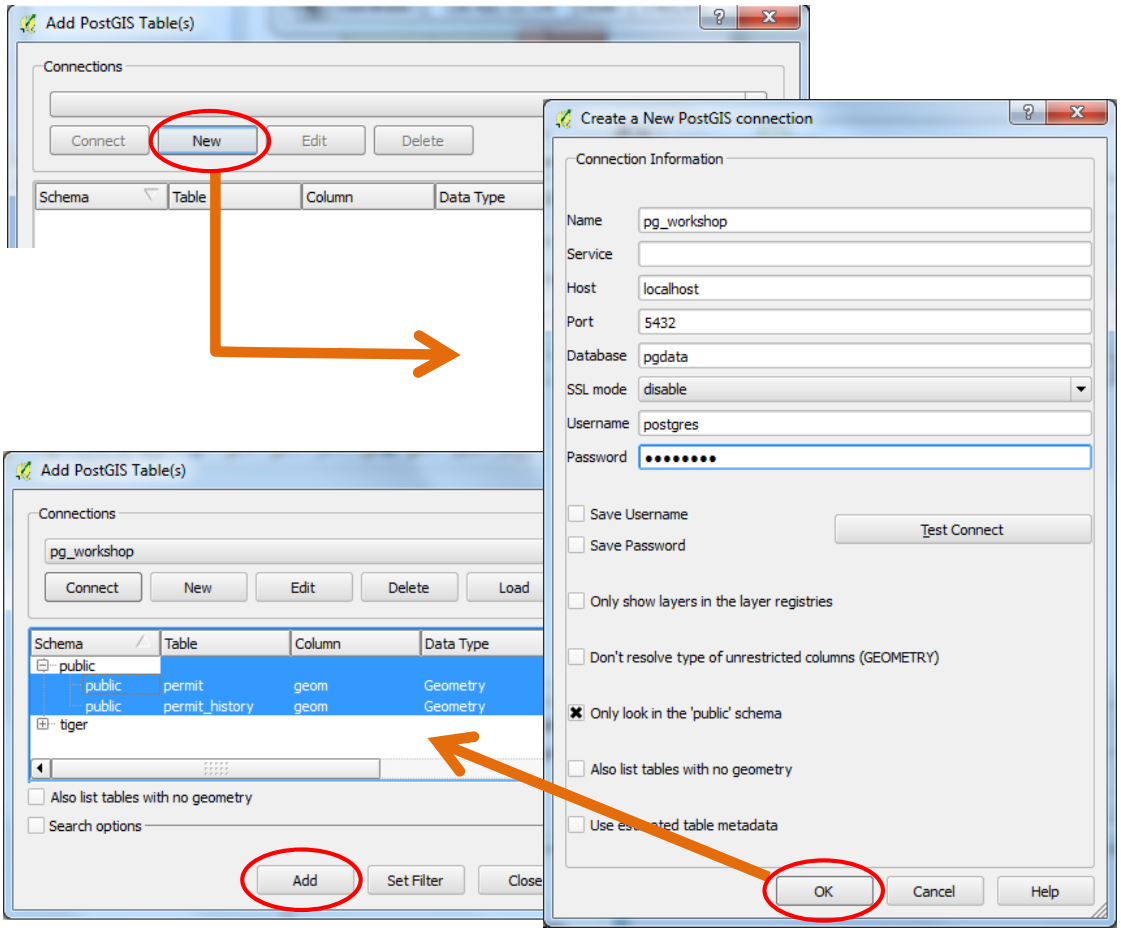
   From the *Portable GIS Control Panel* click tab *Desktop Modules*, and click on *QGIS*.

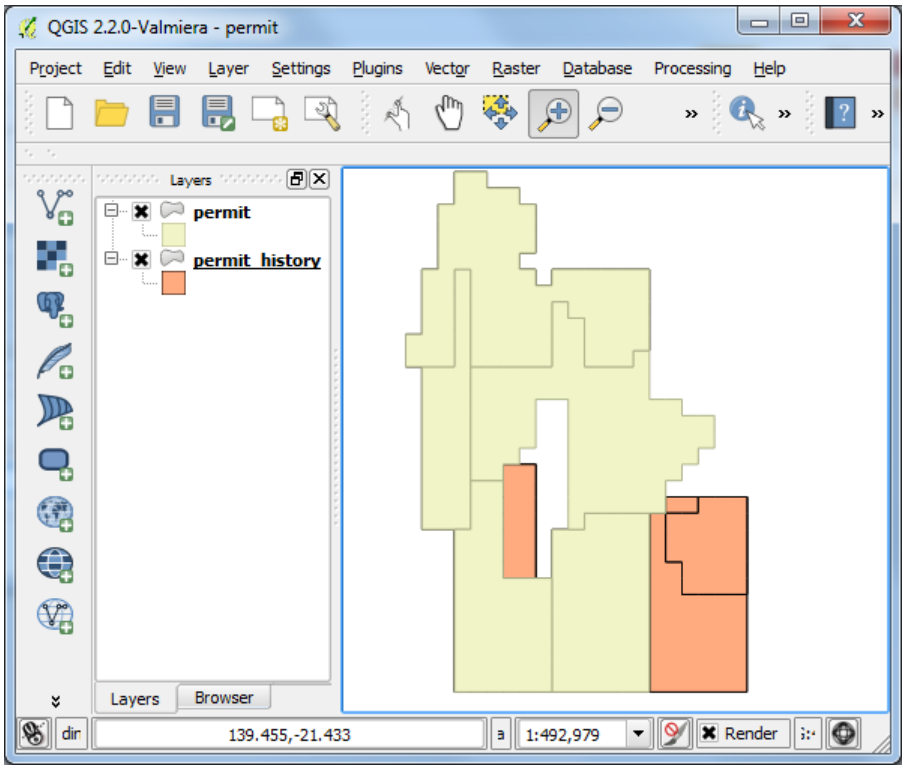


8. *Add GIS layers to QGIS*

   In QGIS we create a connection to a PostGIS database  . Click New and complete form with the connection information as above. You can test the connection in this form, click OK.

   Select both *permit* and *permit_history* and click Add to add as layers to QGIS.


Add PostGIS Table(s)
Connections
Connect
New
Edit
Delete
Schema
Table
Column
Data Type
Create a New PostGIS connection
Connection Information
Name
pg_workshop
Service
Host
localhost
Port
5432
Database
pgdata
SSL mode
disable
Username
postgres
Password
Save Username
Save Password
Test Connect
Only show layers in the layer registries
Don't resolve type of unrestricted columns (GEOMETRY)
Only look in the 'public' schema
Also list tables with no geometry
OK
Cancel
Help
Add PostGIS Table(s)
Connections
pg_workshop
Connect
New
Edit
Delete
Load
Schema
Table
Column
Data Type
public
public
permit
geom
Geometry
public
permit_history
geom
Geometry
tiger
Also list tables with no geometry
Search options
Add
Set Filter
Close


QGIS 2.2.0-Valmiera - permit
Project
Edit
View
Layer
Settings
Plugins
Vector
Raster
Database
Processing
Help
Layers
permit
permit_history
Layers
Browser
139.455,-21.433
1:492,979
Render

Support for historical tracking

Database management systems provide a special mechanism to control tables when rows are modified. The mechanism is specified by SQL statements that are executed when SQL insert, update, or delete operations are performed (either just before or after). The control actions are specified in a user function on rows; it refers to new rows being added to database with the keyword NEW, and to rows being deleted with the keyword OLD. The functions specify what happens, but require a trigger to be set in the database to fire it off.

SQL triggers provide a means to assure valid processing occurs within the database. They happen invisibly to the client application so it is important to have a well-defined logical purpose. We use triggers to keep the history of permits, with only minor modification to the current permit table. Namely it notes the time (or date) a change occurs, but then records all changes in a permit history table keeping track of the start and end dates when these changes takes effect. The table below lists the triggers and functions used.

We return to pgAdmin to setup temporal data management of the historical dataset.

9. *In pgAdmin add triggers to PostgreSQL*

Use the SQL tool  to enter the triggers and functions.
First enter the functions functions: *insertPermit(), deletePermit()*, and *updatePermit()* by copying text from table below to the SQL editor and execute it  . Do each separately using the Clear the window command  between each.

Next enter the triggers: *on_insertPermit()*, *on_deletePermit()*, and *on_updatePermit()* by selecting and copy the text below to the SQL editor and execute it.

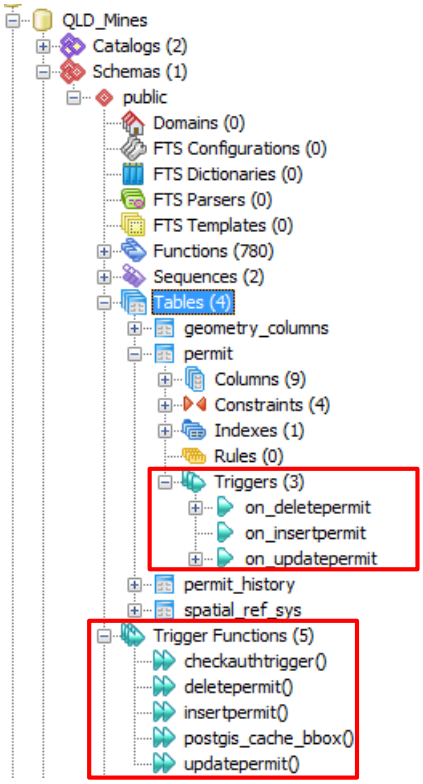
Insert Trigger	Delete Trigger	Update Trigger

CREATE FUNCTION insertPermit()
RETURNS TRIGGER AS $$
BEGIN
NEW.timeStamp = current_date;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
	CREATE FUNCTION deletePermit()
	RETURNS TRIGGER AS $$
	BEGIN
	INSERT INTO permit_history
	VALUES(
	DEFAULT,
	OLD.id,
	OLD.name,
	OLD.area,
	OLD.rightType,
	OLD.unitType,
	OLD.timeFrom,
	OLD.timeDuratn,
	OLD.adminType,
	OLD.timeStamp,
	current_date,
	OLD.geom
	);
	RETURN null;
	END;
	$$ LANGUAGE plpgsql;
		CREATE FUNCTION updatePermit()
		RETURNS TRIGGER AS $$
		BEGIN
		NEW.timeStamp = current_date;
		INSERT INTO permit_history
		VALUES (
		DEFAULT,
		OLD.id,
		OLD.name,
		OLD.area,
		OLD.rightType,
		OLD.unitType,
		OLD.timeFrom,
		OLD.timeDuratn,
		OLD.adminType,
		OLD.timeStamp,
		current_date,
		OLD.geom
		);
		RETURN NEW;
		END;
		$$ LANGUAGE plpgsql;

CREATE TRIGGER on_insertPermit BEFORE INSERT ON permit FOR EACH ROW EXECUTE PROCEDURE insertPermit();	CREATE TRIGGER on_deletePermit AFTER DELETE ON permit FOR EACH ROW EXECUTE PROCEDURE deletePermit();	CREATE TRIGGER on_updatePermit BEFORE UPDATE ON permit FOR EACH ROW EXECUTE PROCEDURE updatePermit();

As you enter functions and triggers in SQL editor and execute them you see a message to say it succeeded (if not go back, delete and re-enter following instructions carefully).

Refresh the contents tree in pgAdmin to see the functions and triggers are listed.



10. Insert a new permit inn pgAdmin

We will run an SQL statement to insert a new permit application. The SQL is shown below. Note we convert a text definition of the geometry coordinates to the geometry type in PostgreSQL; this has a multi-polygon geometry as one permit may cover arbitrary extents. Copy the SQL below and execute it in pgAdmin.

INSERT INTO permit VALUES (DEFAULT, 19899, 'HAPPY MINES LTD', 6393.2, 'Exploration', 'APPLICATION', '2015-8-1', 1, 'APPLY', NULL, ST_GeomFromText('MULTIPOLYGON(((139.467839 -21.248515, 139.501171 -21.248515, 139.501174 -21.41518, 139.467839 -21.41518, 139.467839 -21.248515)))', 4326))

View data for the permit table in QGIS. Note that the change date is automatically set with the *insertPermit()* function.

11. Modify a permit in QGIS

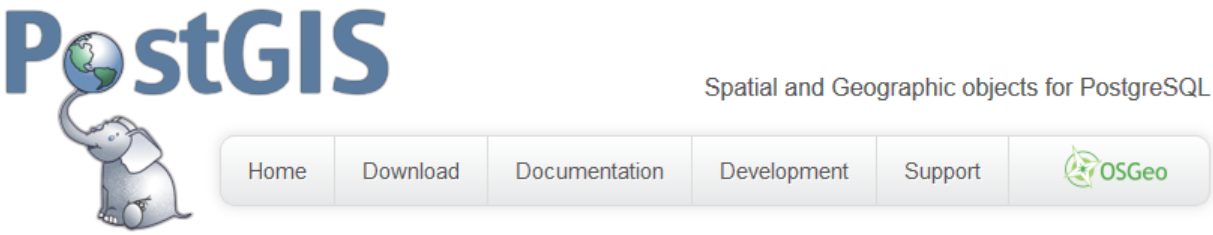
Edit  the data for permits in QGIS. Change the name of the owner for the newly added permit from 'HAPPY MINES LTD' to 'UNHAPPY MINES LTD'. Stop editing and save the changes to layer when prompted by QGIS.

Now open a view of data for the permit_history table. Notice and entry with the old row data is stored with dates used to track changes, this was done by *updatePermit()* function.

12. Delete a permit in QGIS

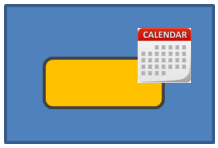
Again go back and edit  the data for permit table in QGIS. On left margin select the whole row for the last row and delete it  . Stop editing and save the changes to layer when prompted by QGIS.

Now open a view of data for the permit_history table. Notice and entry with the old row data is stored with dates used to track changes, this was done by *deletePermit()* function.



We will use PostGIS (http://postgis.net) for many of the queries. PostGIS is an add-on to PostgreSQL to extend its functionality for spatial database processing. It is not absolutely necessary but is definitely handy, and used by many people!

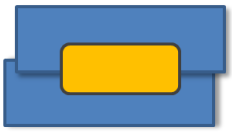
13.Queries on historical permit database in pgAdmin



Query permits that expire in the next month

This case arises frequently for notifications that a permit will expire soon. In the SQL query you need to do a little date algebra. See documentation http://www.postgresql.org/docs/9.2/static/functions-datetime.html. A permit expires after the granted date plus the duration (assumed to be in years in this example). In SQL use the time specification to get the future date, i.e. *date_grant + duration * '1 YEAR'::interval*. We assume the current date is 1st September 2015 (or use PotgreSQL keyword for *current_date* in real world application) and add 1 month to this. This should return result that "HAPPY MINES LTD" had a 2 year permit starting 27th September 2013, so it will expire within in the next month.

```
SELECT id, name, timeFrom, timeDuratn FROM permit WHERE (timeFrom + timeDuratn * interval '1 YEAR') <
(date '2015-9-1' + interval '1 MONTH')
```



Report on the complete record of a permit

This query needs to combine the current state permit table and the permit history table. This is implemented by a SQL union operator which combines two queries. The query statements are formatted to have the same number of attributes and types. Finally the output is sorted on the change date in the lifespan for permits.

```
SELECT id, name, unitType, timeFrom, timeDuratn, adminType, timeStamp FROM permit WHERE id=14366
UNION
SELECT id, name, unitType, timeFrom, timeDuratn, adminType, beginTime as timeStamp FROM
permit_history WHERE id=14366
ORDER BY timeStamp;
```

Finding permits over a certain area

Spatial query is a common requirement in many applications. People frequently want to know current or historical permits in an area. We show this in SQL and how to output the geometry in KML (the format used by Google).

```
SELECT id, name, unitType, timeFrom, timeDuratn, ST_AsKML(geom) as kml FROM permit WHERE
ST_Intersects(geom,ST_MakeEnvelope (139.4 -21.4,139.8,139.8,-21.2,4326))
```

ST_AsKML() is a PostGIS function to output the geometry as text which may be copied into a kml file. Below is a simple kml file which you can copy/paste to a text editor. Click on one cell from kml output for above query, copy its

contents for multi-geometry and paste into KML template below at spot '…paste here'. You may then open the file in Google Earth and you will see a permit displayed. The significance of doing this is that we can run queries on spatial databases with Google-compatible output; we automate this later as a web service linked directly to Google Earth.

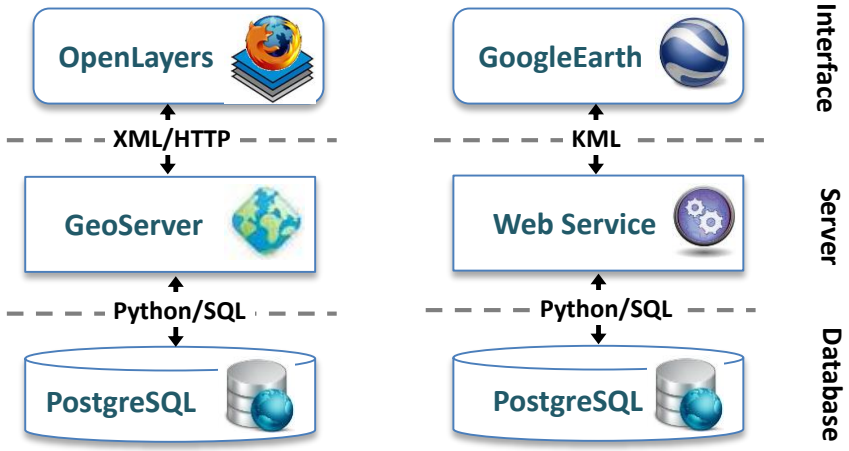
```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Placemark>
    <name>Polygon placemark</name>
    <description>Example output </description>
    ...  paste here  ...
  </Placemark>
</kml>
```

These queries demonstrate information we wish to explore for permits, and just some of the potent SQL. We refer to online documentation for PostgreSQL http://www.postgresql.org/ and PostGIS http://postgis.net to learn more with queries and reporting.

14. Visualising permit data over the Internet

The next logical stage is to view the data online along with other base-mapping layers.

There are a couple of software options to view the data stored in PostgreSQL over the Internet: i) as an open source server application, and ii) as a scriptable web service (see figure below). OpenGeo provide a tutorial on setting up a server application (http://workshops.opengeo.org/postgis-spatialdbtips), but it still requires significant technical expertise. We use the latter option because it was the easiest to implement and illustrates time-based permit queries.



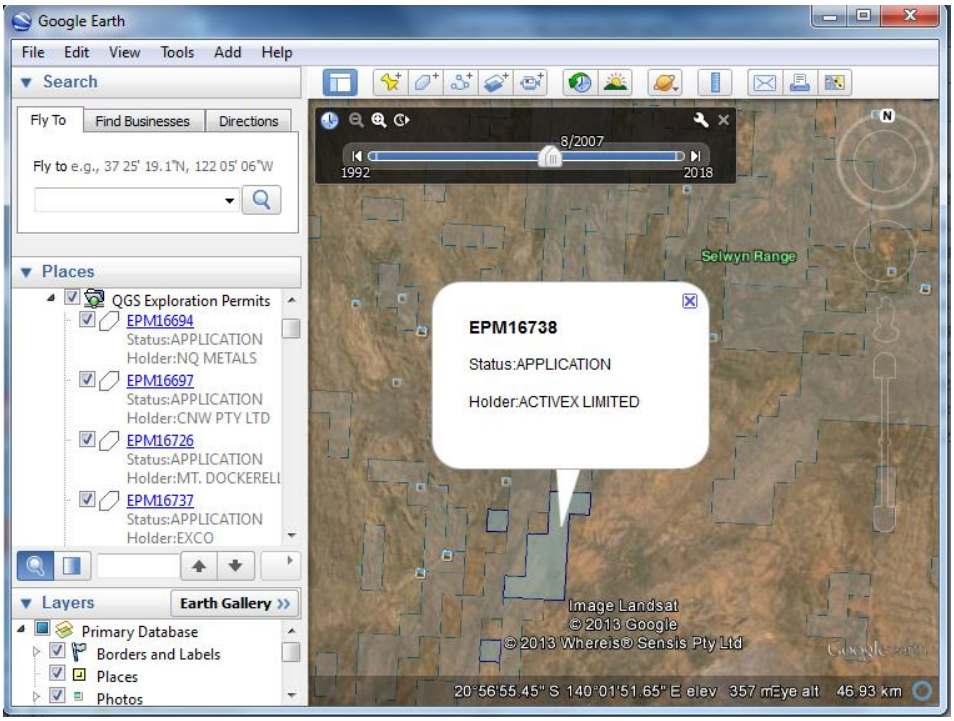
i) Web client with server application ii) Web client link to scriptable web service

Figure . Options for client-server computing architecture to access permit database.

The **client part** uses Google Earth as it provides satellite base-mapping and timespan display. External data may be added to Google Earth by opening a KML as well as other server layers. All we do is add KML data using the *Network Link* in KML which accesses a web service return KML spatial data.

The **server part** is implemented as a web service written using a python script. It processes parameters from Google Earth, the main one being a bounding box for query. The scripts then performs a PostGIS query similar to the one described in the last section returning KML.

It is beyond this workshop to explain the methods fully, we refer to article '*Using PHP and MySQL to create KML*' which describes the approach using PHP. https://developers.google.com/kml/articles/phpmysqlkml . The figure below shows example of web service running in Google Earth.



References

Kulkarni, K., and Jan-Eike M (2012) Temporal features in SQL: 2011. ACM SIGMOD Record 41(3), 34-43.

Christiaan Lemmen C., van Oostrom P. and van der Molen, P. (2013) Land Administration Domain Model. GIM International 27(6), 15-15. [URL: www.gim-international.com/issues/articles/id1984-Land_Administration_Domain_Model.html]

Lemmen C., van Oosterom P. and Bennett, R. (2015 in press) The Land Administration Domain Model. Land Use Policy.

Pullar, D. (2015 submitted) An implementation model for managing mining interests with permits using standards for land administration, submitted to Computers & Geosciences.