

Qt, Python, Postgis, XML, des outils au service de l'exploration de données métier dans QGIS

Sylvain PIERRE
Conseil Départemental du Bas-Rhin

Problématique

Comment introduire des univers de données métier dans QGIS:

- sélectionner, traiter et cartographier des données métier de manière itérative
- interface conviviale

Sur la base d'un plugin dédié à une thématique spécifique,
dégager des méthodes et des outils pouvant être généralisés

Cahier des charges : exploiter les données qualité des cours d'eau

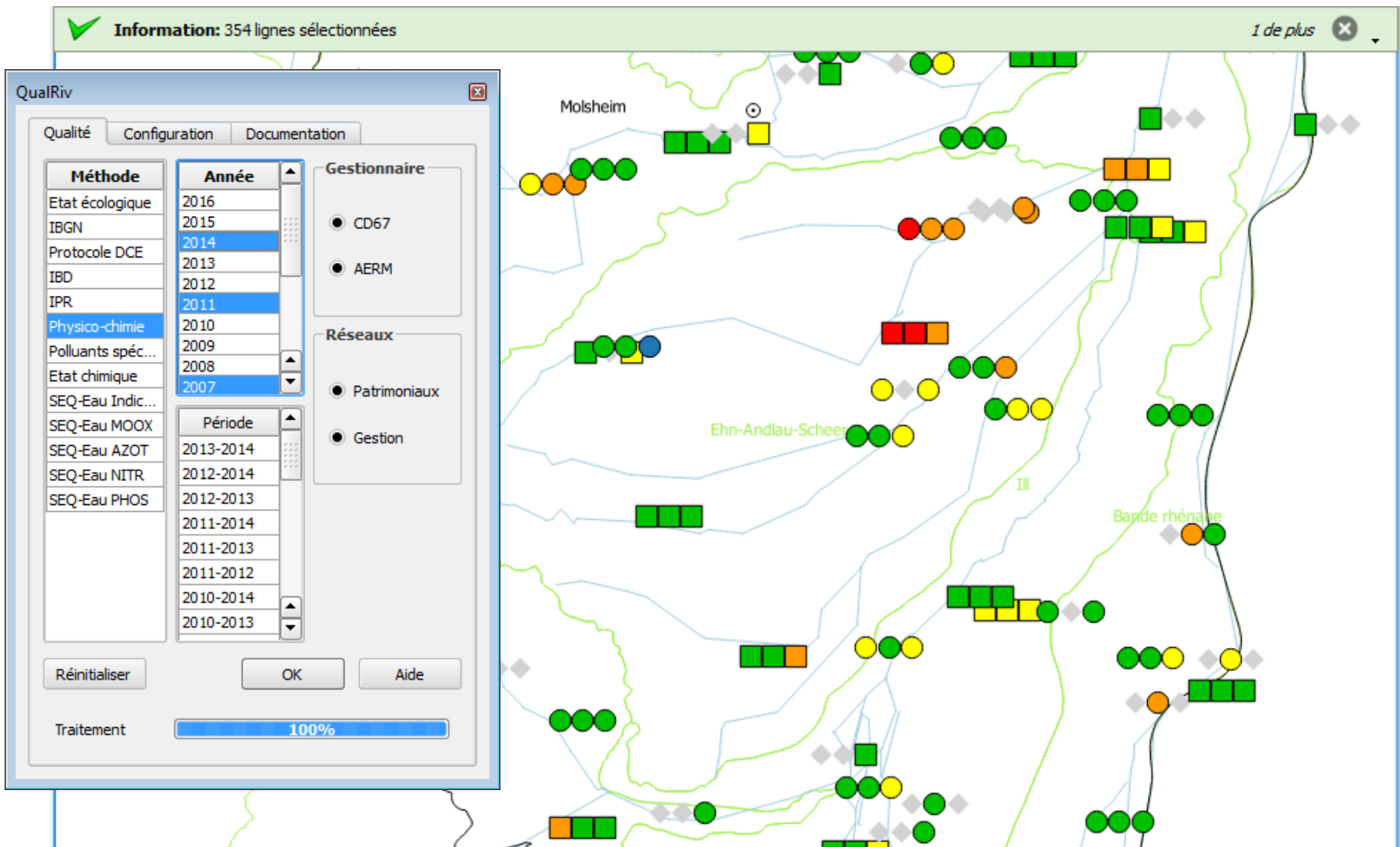
res_nat_bas	lib_methode	indice_qual	lib_periode	lib_gestionnaire	lib_reseau	lib_confiance
2001050	SEQ-Eau Indice macro	65	2001	Agence de l'Eau Rhin-Meuse	Réseau National de Bassin	DCE non compatible
2001500	SEQ-Eau Indice macro	62	2001	Conseil Général du Bas-Rhin	Réseau d'Intérêt Départemental	DCE non compatible
2098800	SEQ-Eau Indice macro	62	2007-2011	Agence de l'Eau Rhin-Meuse	Réseau de contrôle et de surveillance	DCE non compatible
2001046	SEQ-Eau Indice macro	65	2013	Conseil Général du Bas-Rhin	Réseau de connaissance départemental	DCE non compatible
2001050	SEQ-Eau Indice macro	66	2013	Agence de l'Eau Rhin-Meuse	Réseau de contrôle et de surveillance	DCE non compatible
2098300	IBGN	15	2014	Conseil Général du Bas-Rhin	Réseau d'Intérêt Départemental	DCE non compatible
2098450	IBGN	10	2014	Conseil Général du Bas-Rhin	Réseau d'Intérêt Départemental	DCE non compatible
2001050	Physico-chimie	2	2007	Agence de l'Eau Rhin-Meuse	Réseau de contrôle et de surveillance	DCE compatible
2001500	Physico-chimie	2	2007	Conseil Général du Bas-Rhin	Réseau de connaissance départemental	DCE compatible
2001600	Physico-chimie	2	2007	Agence de l'Eau Rhin-Meuse	Réseau de contrôle opérationnel	DCE compatible
2098450	SEQ-Eau MOOX	56	2012	Conseil Général du Bas-Rhin	Réseau de connaissance départemental	DCE non compatible
2098800	SEQ-Eau MOOX	59	2012	Agence de l'Eau Rhin-Meuse	Réseau de contrôle et de surveillance	DCE non compatible
2043000	SEQ-Eau MOOX	69	2012	Conseil Général du Bas-Rhin	Réseau de connaissance départemental	DCE non compatible
2001046	SEQ-Eau NITR	65	2012	Conseil Général du Bas-Rhin	Réseau de connaissance départemental	DCE non compatible
2001050	SEQ-Eau NITR	66	2012	Agence de l'Eau Rhin-Meuse	Réseau de contrôle et de surveillance	DCE non compatible
2001500	SEQ-Eau NITR	49	2012	Conseil Général du Bas-Rhin	Réseau de connaissance départemental	DCE non compatible
2098450	SEQ-Eau AZOT	68	2007-2011	Conseil Général du Bas-Rhin	Réseau de connaissance départemental	DCE non compatible
2098600	SEQ-Eau AZOT	73	2007-2011	Agence de l'Eau Rhin-Meuse	Réseau de contrôle et de surveillance	DCE non compatible
2098800	SEQ-Eau AZOT	70	2007-2011	Agence de l'Eau Rhin-Meuse	Réseau de contrôle et de surveillance	DCE non compatible
2001046	SEQ-Eau MOOX	76	2013	Conseil Général du Bas-Rhin	Réseau de connaissance départemental	DCE non compatible
2001050	SEQ-Eau MOOX	66	2013	Agence de l'Eau Rhin-Meuse	Réseau de contrôle et de surveillance	DCE non compatible
2001500	SEQ-Eau MOOX	34	2013	Conseil Général du Bas-Rhin	Réseau de connaissance départemental	DCE non compatible

Structuration « lignes » optimisé pour le stockage
 Nombre de combinaison élevé != pas de réponse prédéfinie


Qualité	Gestionnaire	DCE compatible
Très bon	● CD67	● Non
Bon	■ Agence de l'eau	● Oui
Moyen		
Médiocre		
Mauvais		

◆ Absence de données

Résultat: plugin QualRiv



Résultat: plugin QualRiv

 **Information:** 343 lignes sélectionnées

QualRiv

Qualité Configuration Documentation

Méthode	Année
Etat écologique	2016
IBGN	2015
Protocole DCE	2014
IBD	2013
IPR	2012
Physico-chimie	2011
Polluants spéc...	2010
Etat chimique	2009
SEQ-Eau Indic...	2008
SEQ-Eau MOOX	2007
SEQ-Eau AZOT	
SEQ-Eau NITR	
SEQ-Eau PHOS	

Réinitialiser OK Aide

Traitement **100%**

Gestionnaire

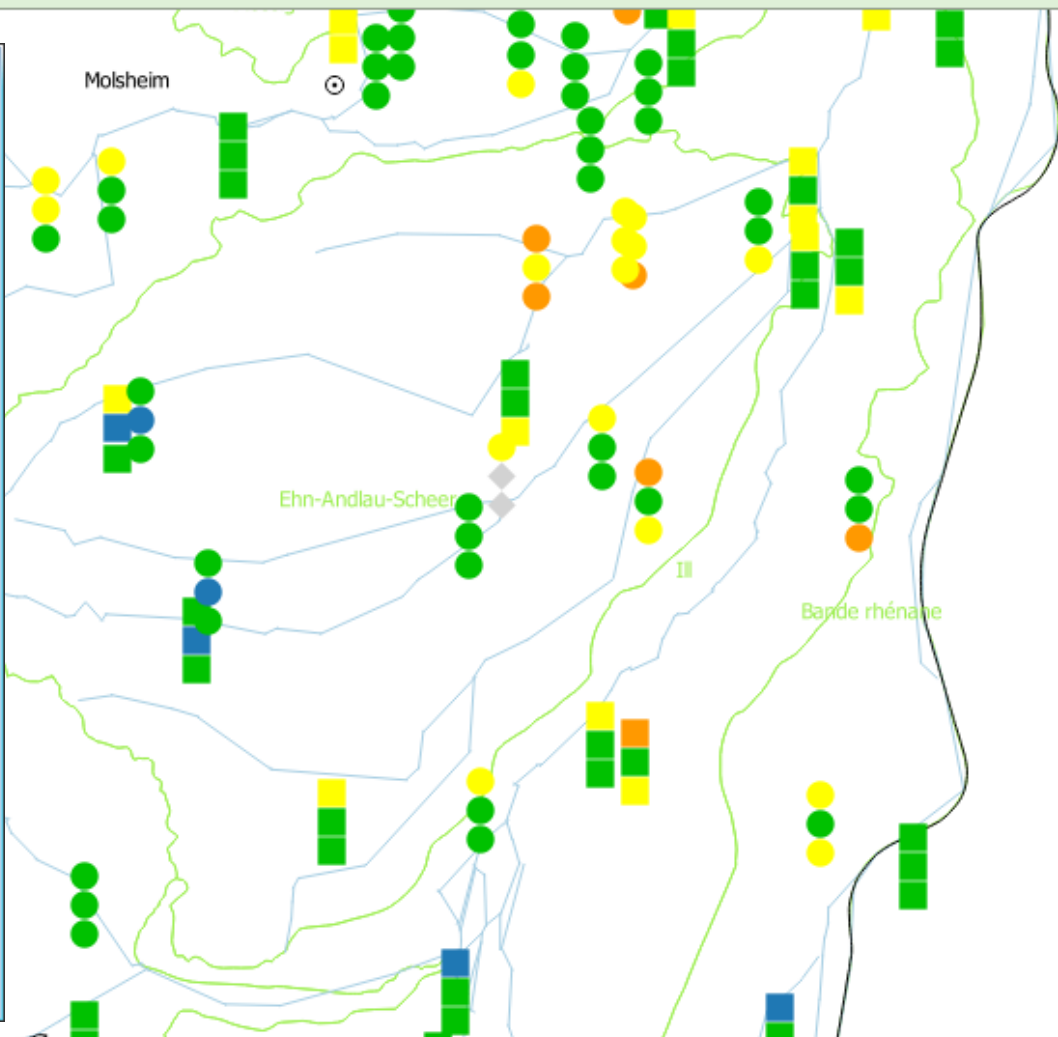
☐ CD67

☐ AERM

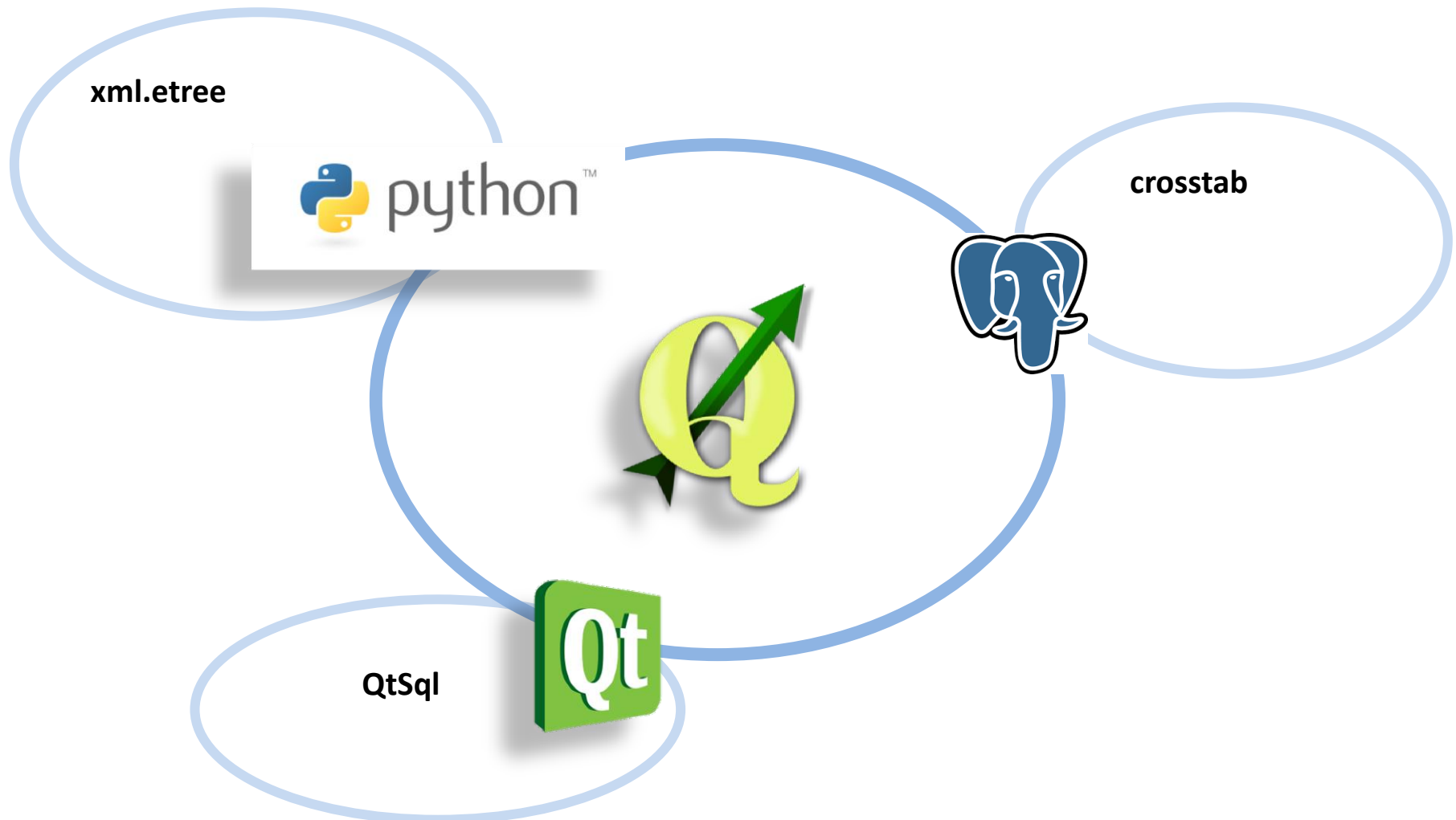
Réseaux

☐ Patrimoniaux

☐ Gestion



Architecture générale d'un plugin « métier »

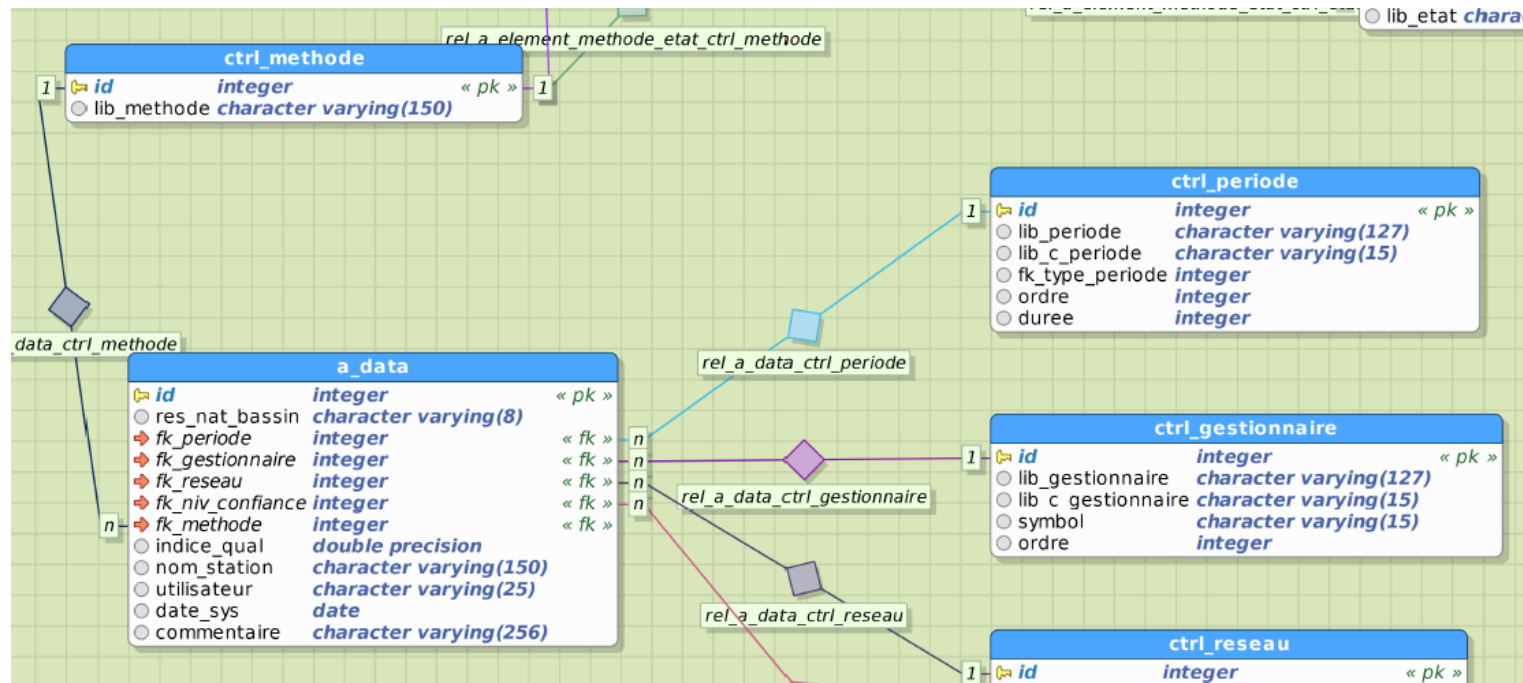


Prérequis: une base de données clairement structurée

- schéma dédié
- rôle de connexion
- modèle optimisé: tables de contrôle, contraintes d'intégrité



pgModeler



QtSql: accès aux données et pilotage des actions sur la base

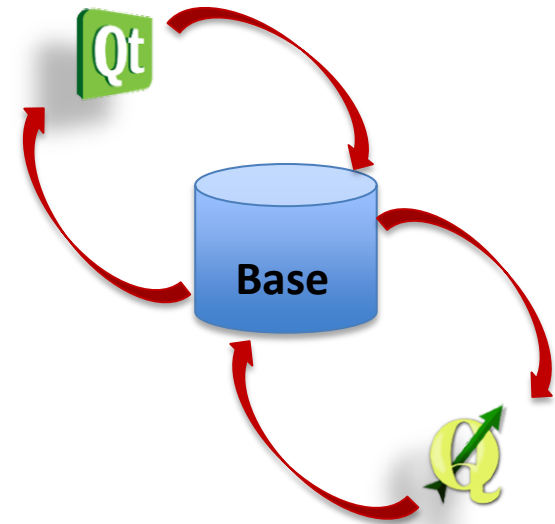


```
from PyQt4.QtSql import QSqlRelationalTableModel, QSqlQuery, QSqlDatabase
```

QSqlDatabase : une classe spécifique pour la base de donnée alpha

```
self.db = QSqlDatabase.addDatabase("QPSQL")
```

```
class QualRivDB(QSqlDatabase):  
    def __init__(self, **kwargs):  
        super(QualRivDB, self).__init__()  
        self.iface = kwargs.get('iface')  
        self.db = QSqlDatabase.addDatabase("QPSQL")  
        self.db.setHostName(kwargs.get('host'))  
        self.db.setPort(kwargs.get('port'))  
        self.db.setDatabaseName(kwargs.get('dbname'))  
        self.db.setUserName(kwargs.get('user'))  
        self.db.setPassword(kwargs.get('password'))
```



Pour mémoire, nécessité d'une connexion QGIS QgsDataSourceURI

```
self.uri = QgsDataSourceURI()  
self.uri.setConnection(conDB['host'], str(conDB['port']), conDB['dbname'], conDB['user'], conDB['password'])
```


QSqlRelationalTableModel, Qt modèle/vue: alimenter les widget de l'interface



Édition des données - POSTGIS-02 (...)

Fichier Édition Affichage Outils Aide

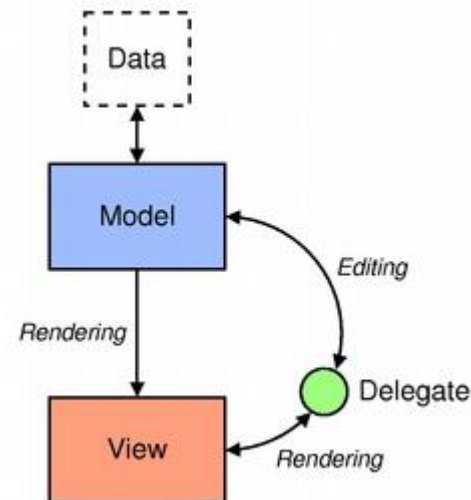
Pas de limite

	id [PK] serial	lib_methode character varying(150)
1	1	Etat écologique
2	2	IBGN
3	3	Protocole DCE
4	4	IBD
5	5	IPR
6	6	Physico-chimie
7	7	Polluants spécifiques
8	8	Etat chimique
9	9	SEQ-Eau Indice macro
10	10	SEQ-Eau MOOX
11	11	SEQ-Eau AZOT
12	12	SEQ-Eau NITR
13	13	SEQ-Eau PHOS
*		

13 lignes.

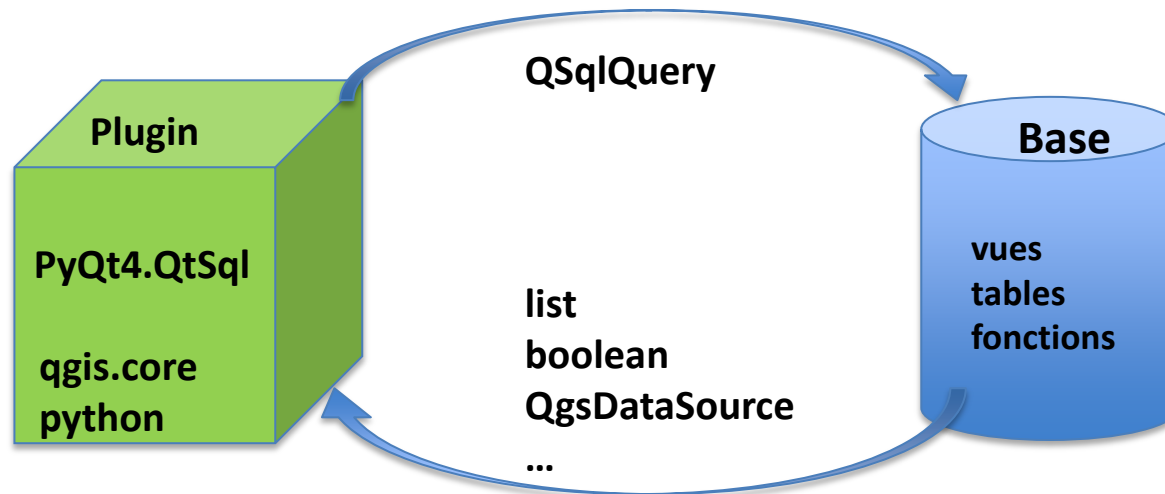
```

self.mdlMethod = QSqlRelationalTableModel(self.tabWidget, self.db)
self.mdlMethod.setTable(self.schemaName+'.ctrl_methode')
self.mdlMethod.sort(0, Qt.AscendingOrder)
self.mdlMethod.setHeaderData(1, Qt.Horizontal, u'Méthode')
self.tblViewMethod.setModel(self.mdlMethod)
  
```



Méthode
Etat écologique
IBGN
Protocole DCE
IBD
IPR
Physico-chimie
Polluants spécifiques
Etat chimique
SEQ-Eau Indice macro
SEQ-Eau MOOX
SEQ-Eau AZOT
SEQ-Eau NITR
SEQ-Eau PHOS

QSqlQuery: actions sur la base



```
lstMethLib = self.db.list_table_values('ctrl_methode',lstMeth )
```

```
def list_table_values(self, table, lstvalues):
    query = QSqlQuery(self.db)
    values = str(lstvalues)[1:-1]
    sql = "SELECT * FROM %s WHERE id IN (%s);" % (table, values)
    lstr = []
    query.exec_(sql)
    while query.next():
        lstr.append(query.value(1))

    return lstr
```

```
if not self.db.has_privilege_create(conDB['user'], self.schemaName):
```

```
def has_privilege_create(self, user, schema):
    query = QSqlQuery(self.db)
    sql = "SELECT has_schema_privilege('%s', '%s', 'CREATE');" % (user,
    schema)
    query.exec_(sql)

    query.next()
    return query.value(0)
```

QSqlQuery: requêtes et création de vue sur la base



```
self.db.create_view_data_filter(lstMeth, lstT , lstGest , lstRes)
```

```
def create_view_data_filter(self, lstMethode, lstTemp , lstGestion , lstReseau):
```

```
    query = QSqlQuery(self.db)
```

```
    data = {'meth': str(lstMethode)[1:-1], 'tmp' : str(lstTemp)[1:-1],  
           'gst' : str(lstGestion)[1:-1], 'res' : str(lstReseau)[1:-1]}
```

```
    sql = """CREATE OR REPLACE VIEW rivqual.v_data_sel AS SELECT  
id, res_nat_bassin, fk_periode, fk_gestionnaire, fk_type_reseau,  
    fk_niv_confiance, fk_methode, ordre, igual, qual, fk_typologie, geom  
FROM v_data  
WHERE fk_methode in %(meth)s AND fk_periode in %(tmp)s  
AND fk_gestionnaire in %(gst)s AND fk_type_reseau in %(res)s);""" % data
```

```
    query.exec_()
```

Nécessité que le rôle de connexion dispose des privilèges nécessaires

Exploitation des données dans Postgres: fonction crosstab

(Oracle , MS-Sql Server = PIVOT)

Exemple trivial



id	rowid	key	value
1	test1	key1	val1
2	test1	key2	val2
3	test1	key3	val3
4	test1	key4	val4
5	test2	key1	val5
6	test2	key2	val6
7	test2	key3	val7
8	test2	key4	val8

rowid	key1	key2	key3	key4
test1	val1	val2	val3	val4
test2	val5	val6	val7	val8

Une caractéristique intéressante des bases de données relationnelles (Postgres dans ce cas) est la possibilité de faire tourner une table autour d'un pivot

Postgres: fonction crosstab, clé de la manipulation de données

Principe appliqué aux données réelles



	rowid	key								value	
id	res_nat_bassin	fk_perio	fk_gestio	fk_type	fk_niv_c	fk_meth	ordre	iqua	double pr	qual	fk_typolo
10276	02001500	10	1	1	1	6	10			2	13
10513	02001500	12	1	1	1	6	12			2	13
10769	02001500	17	1	1	1	6	14			3	13

crosstab()

id	res_nat_bassin	qual_1	fk_conf_1	fk_gest_1	qual_2	fk_conf_2	fk_gest_2	qual_3	fk_conf_3	fk_gest_3
bigint	character varying	integer	integer	integer	integer	integer	integer	integer	integer	integer
3	02001500	2	1	1	2	1	1	3	1	1

Fonction crosstab: en pratique



Activer la fonction crosstab, composante du module tablefunc sous postgres

```
CREATE EXTENSION tablefunc;
```

Une syntaxe à maîtriser;-)

crosstab(text sql)

Renvoie une « table pivot » contenant les noms des lignes ainsi que *N* colonnes de valeur, où *N* est déterminé par le type de ligne spécifié par la requête appelant

crosstab(text source_sql, text category_sql)

Produit une « table pivot » avec les colonnes des valeurs spécifiées par une autre requête

```
SELECT * FROM  
crosstab('SELECT <rowid, key, value > FROM <source> ORDER BY 1,2',  
        'SELECT <key> FROM <source> ORDER by 1') ;
```




Fonction crosstab: points clés

Table pivote sur 1 colonne : utiliser des types ARRAY pour regrouper les données

ARRAY monotype (int, char,...)=> une base bien structurée : fk, tables de contrôle,...

```
SELECT res_nat_bassin, fk_periode, array [qual, fk_niv_confiance, fk_gestionnaire]
```

res_nat_bassin character varying	qual1 integer[]	qual2 integer[]	qual3 integer[]
02001500	{2,1,1}	{2,1,1}	{3,1,1}

Forcer les valeurs pivot

```
SELECT fk_periode from v_data_sel ORDER by 1
```

NoOK

```
SELECT fk_periode from (VALUES (18), (12), (10)) int(fk_periode) ORDER by 1
```

Année
2016
2015
2014
2013
2012
2011
2010
2009

Permet de remonter une colonne là ou absence totale de données

id bigint	res_nat_bassin character varying	qual_1 integer	fk_conf_1 integer	fk_gest_1 integer	qual_2 integer	fk_conf_2 integer	fk_gest_2 integer	qual_3 integer	fk_conf_3 integer	fk_gest_3 integer	geom geometry(Point,3948)
1	02001046	2	1	1	2	1	1				01010000206C0F00003
2	02001050	2	1	2	2	1	2				01010000206C0F0000F
3	02001500	2	1	1	2	1	1				01010000206C0F0000C
4	02001700	2	1	2	2	1	2				01010000206C0F0000E

OK

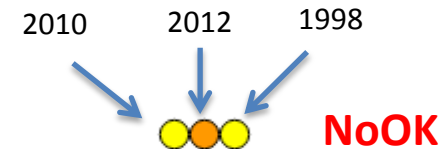
Fonction crosstab: points clés

Clés primaires tables de contrôle, ordre chronologique et affichage



id integer	lib_période character var	fk_type integer	ordre integer
1	2001	1	1
2	2002	1	2
3	2003	1	3
4	2004	1	4
5	2005	1	5
6	2006	1	6
7	2007	1	7
8	2008	1	8
9	2009	1	9
10	2010	1	10
11	2011	1	11
12	2012	1	12
13	2013	1	13
14	2004-2006	2	1
15	2007-2011	2	5
16	2012-2014	2	25
17	2014	1	14
18	2015	1	15

```
SELECT fk_période from (VALUES (19), (12), (10)) int(fk_période) ORDER by 1
```



Travail conséquent sur la base:

DROP contraintes d'intégrité

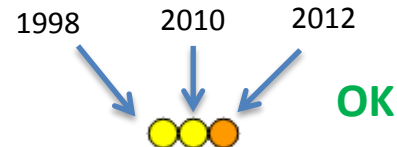
DELETE + INSERT des nouvelles valeurs de la table de contrôle

INSERT nouvelles valeurs table principale

UPDATE fk dans la table principale

Remise en place des contraintes d'intégrité

```
SELECT fk_période from (VALUES (13), (11), (1)) int(fk_période) ORDER by 1
```



Préconisation

Anticiper et prévoir d'emblée toutes les occurrences temporelles

Exploiter la table pivot

Dé-groupement des données et jointure avec le référentiel géographique



```

SELECT row_number() OVER () AS id,
data.res_nat_bassin,
data.qual1[1] AS qual_1, data.qual1[2] AS fk_conf_1, data.qual1[3] AS fk_gest_1,
data.qual2[1] AS qual_2, data.qual2[2] AS fk_conf_2, data.qual2[3] AS fk_gest_2,
data.qual3[1] AS qual_3, data.qual3[2] AS fk_conf_3, data.qual3[3] AS fk_gest_3,
ref.geom::geometry(Point,3948) AS geom
FROM v_data_sel_cross data, ref_station ref
WHERE data.res_nat_bassin = ref.res_nat_bassin;
  
```

Chargement en format memory layer

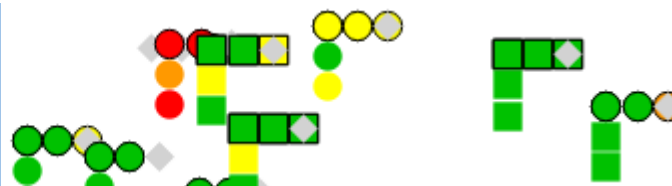
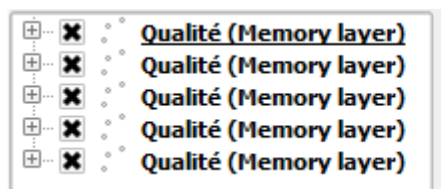
▼ **Infos**

Nom de la couche
 afficher en tant que

Source de la couche

Encodage des données sources ▼

Permet à l'utilisateur de multiplier les représentations



xml.etree – gestion symbologie

Générer un fichier de style (QML) à la volée pour chaque requête utilisateur



QML = XML

```
</categories>
<symbols>
  <symbol alpha="1" type="marker" name="0">
    <layer pass="0" class="EllipseMarker" locked="0">
      <prop k="angle" v="0"/>
      <prop k="color" v="217,217,255"/>
      <prop k="horizontal_anchor_point" v="1"/>
      <prop k="offset" v="4,0"/>
      <prop k="offset_map_unit_scale" v="0,0"/>
      <prop k="offset_unit" v="MM"/>
      <prop k="outline_color" v="0,0,0,255"/>
      <prop k="outline_style" v="no"/>
    </layer>
  </symbol>
</symbols>
```

ElementTree XML API : parser XML python simple

```
from xml.etree import cElementTree as etree
```

xml.etree – gestion symbologie

Identifier la structure XML cible (balises et attributs)




```

<?xml version="1.0" ?>
<qgis>
  <renderer-v2 attr="res_nat_bassin" symbollevels="0" type="categorizedSymbol">
    <categories>
      <category label="02001046" render="true" symbol="0" value="02001046"/>
      <category label="02001500" render="true" symbol="1" value="02001500"/>
      ...
    </categories>
    <symbols>
      <symbol alpha="1" name="0" type="marker">
        <layer class="SimpleMarker" locked="0" pass="0">
          ...
        </layer>
        <layer class="SimpleMarker" locked="0" pass="0">
          ...
        </layer>
        <layer class="SimpleMarker" locked="0" pass="0">
          ...
        </layer>
      </symbol>
      <symbol alpha="1" name="1" type="marker">
        ...
      </symbol>
    </symbols>
  </renderer-v2>
</qgis>
  
```

Catégorisé
 Colonne: res_nat_bassin

Symbole	Valeur	Légende
	02001046	02001046
	02001500	02001500



Marker

- Symbole simple
- Symbole simple
- Symbole simple

xml.etree – gestion symbologie



Identifier la structure XML cible – symbole, élément de base






```



<layer class="SimpleMarker" locked="0" pass="0">
  <prop k="angle" v="0"/>
  <prop k="color" v="0,193,0,255"/>
  <prop k="horizontal_anchor_point" v="0"/>
  <prop k="name" v="circle"/>
  <prop k="offset" v="0,0"/>
  <prop k="offset_map_unit_scale" v="0,0"/>
  <prop k="offset_unit" v="MM"/>
  <prop k="outline_color" v="0,0,0,255"/>
  <prop k="outline_style" v="solid"/>
  <prop k="outline_width" v="0"/>
  <prop k="outline_width_map_unit_scale" v="0,0"/>
  <prop k="outline_width_unit" v="MM"/>
  <prop k="scale_method" v="area"/>
  <prop k="size" v="3.8"/>
  <prop k="size_map_unit_scale" v="0,0"/>
  <prop k="size_unit" v="MM"/>
  <prop k="vertical_anchor_point" v="1"/>
</layer>
  
```



Type de symbole: **Symbole simple**




Couleurs: Remplissage  Bordure 



Taille: 3.800000   Millimètre














Style de bordure externe: **Ligne continue** 

Largeur de bordure externe: 0.000000   Millimètre

Angle: 0.00 °  

Décalage X,Y: 0.000000  0.000000   Millimètre

Point d'ancrage: **Gauche**  **Centre vertical** 

xml.etree – gestion symbologie

Une classe python pour chaque nœud de la hiérarchie XML



Principe de construction

Racine de la hiérarchie

```
etreeElt = etree.Element("qgis")
```

```
subElt = etree.SubElement(etreeElt, "balise")
```

Niveaux hiérarchiques gérés par la relation
"parent/enfant" entre classe

```
self.renderer = RendererCat(etreeElt) #n+1  
self.cat = Category(self.renderer.categories) #n+2  
self.symbols = Symbol(self.renderer.symbols) #n+2  
layerSymb = LayerSymbol(self.symbols.symbol) #n+3
```

```
class RendererCat():  
    def __init__(self, root):  
        self.renderer = etree.SubElement(root, "renderer-v2")  
        self.categories = etree.SubElement(self.renderer, "categories")  
        self.symbols = etree.SubElement(self.renderer, "symbols")  
  
    def create(self):  
        self.renderer.set("attr", "res_nat_bassin")  
        self.renderer.set("symbollevels", "0")  
        self.renderer.set("type", "categorizedSymbol")  
  
class Category():  
    def __init__(self, parent):  
        self.category = etree.SubElement(parent, "category")  
  
    def create(self, index, value):  
        self.category.set("render", "true")  
        self.category.set("symbol", index)  
        self.category.set("value", value)  
        self.category.set("label", value)  
  
class Symbol():  
    def __init__(self, parent):  
        self.symbol = etree.SubElement(parent, "symbol")  
  
    def create(self, value):  
        self.symbol.set("alpha", "1")  
        self.symbol.set("type", "marker")  
        self.symbol.set("name", value)
```

xml.etree – gestion symbologie

QgsFeatureIterator Itération



```
def init_render(self):
    etreeElt = etree.Element("ggis")
    self.renderer = RendererCat(etreeElt)
    self.renderer.create()

    field_names = [field.name() for field in self.layer.pendingFields()]
    idx = 0
    iterL = self.layer.getFeatures()
    self.nbfeatures = self.layer.featureCount()
    for f in iterL:
        data = dict(zip(field_names, f.attributes()))

        self.cat = Category(self.renderer.categories)
        self.cat.create(str(idx), data["res_nat_bassin"])

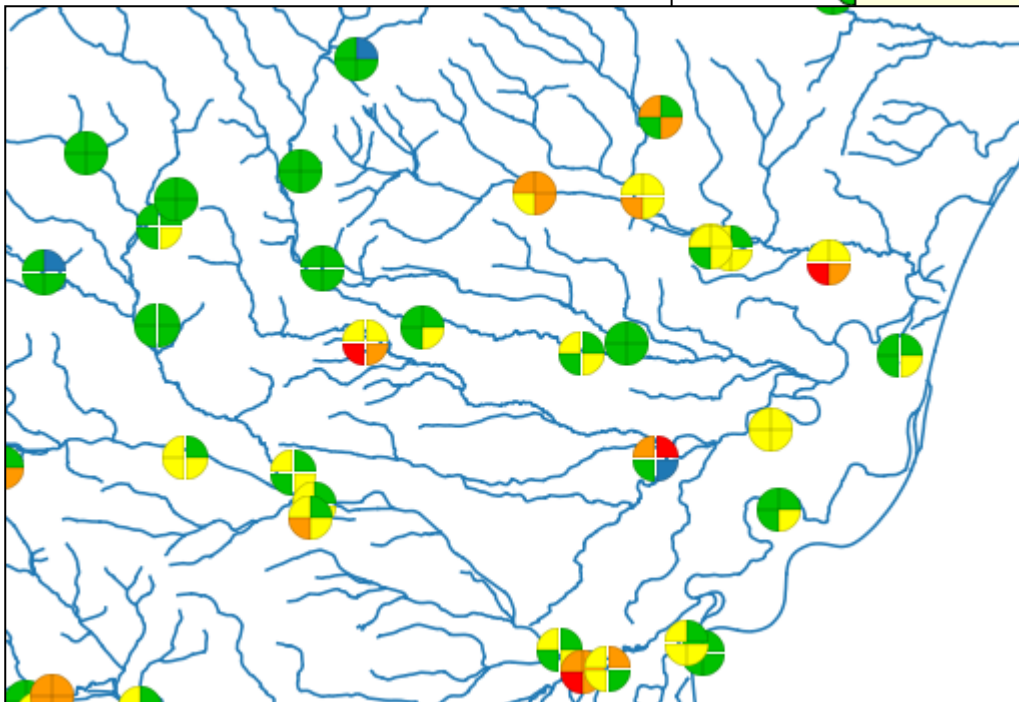
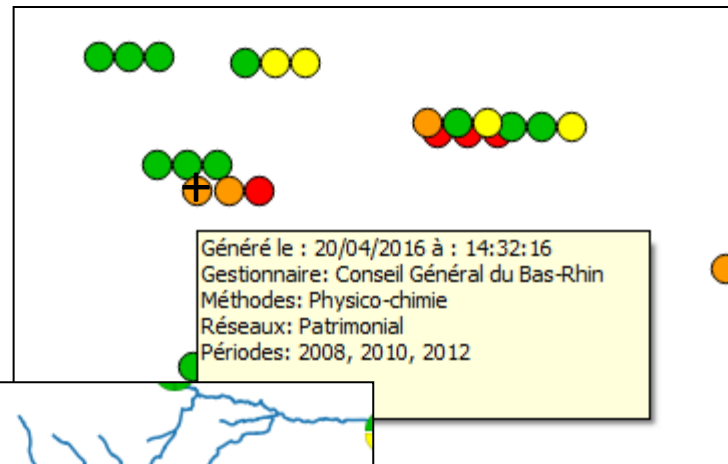
        self.symbols = Symbol(self.renderer.symbols)
        self.symbols.create(str(idx))

        data1 = {k: v for k, v in data.items() if k.endswith('_1')}
        data2 = {k: v for k, v in data.items() if k.endswith('_2')}
        data3 = {k: v for k, v in data.items() if k.endswith('_3')}

        if data1:
            layerSymb = LayerSymbol(self.symbols.symbol)
            layerSymb.create(data1["fk_gest_1"], data1["qual_1"], data1["fk_conf_1"], 1, self.sens)
        if data2:
            layerSymb = LayerSymbol(self.symbols.symbol)
            layerSymb.create(data2["fk_gest_2"], data2["qual_2"], data2["fk_conf_2"], 2, self.sens)
        if data3:
            layerSymb = LayerSymbol(self.symbols.symbol)
            layerSymb.create(data3["fk_gest_3"], data3["qual_3"], data3["fk_conf_3"], 3, self.sens)
        idx += 1
```

Éléments complémentaires

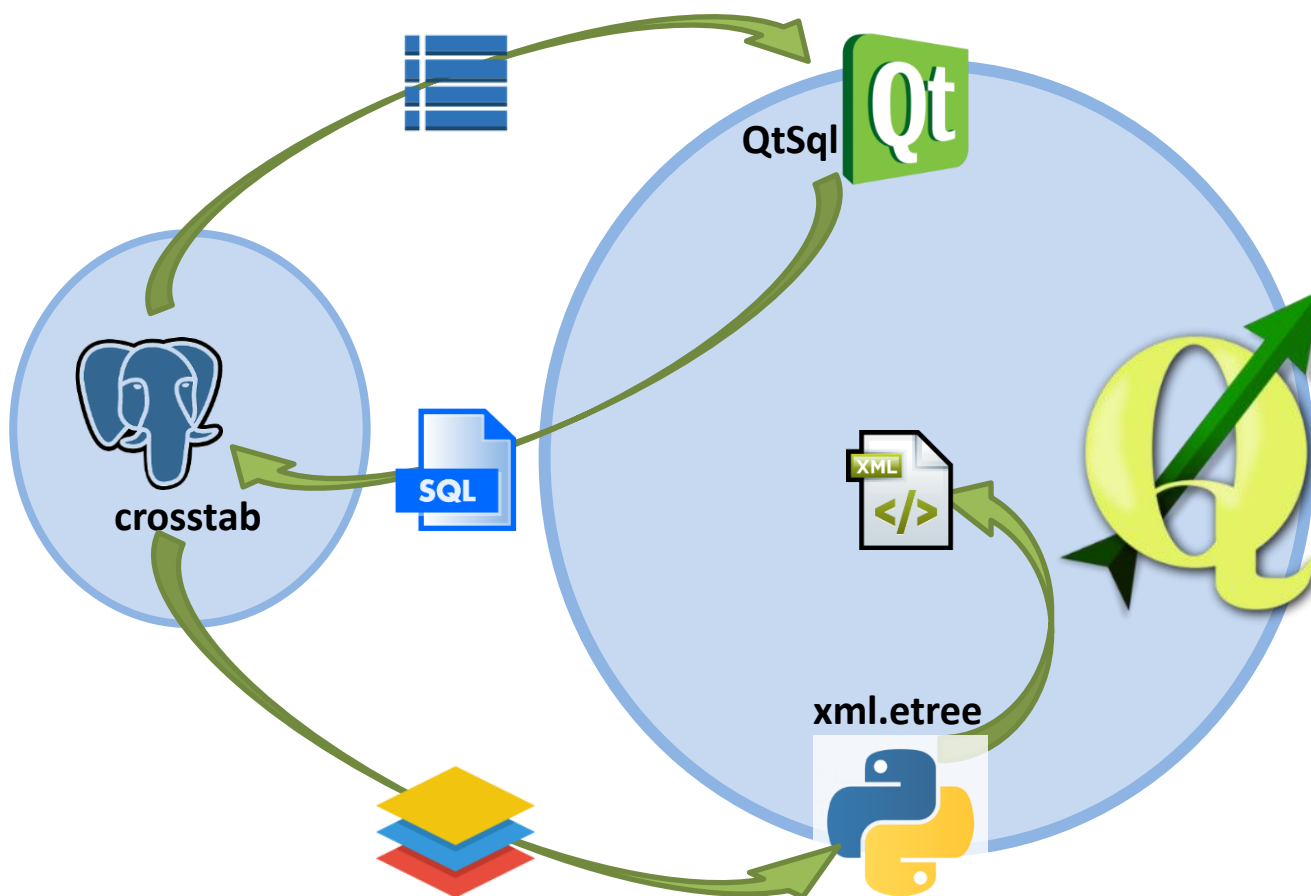
Renseignement de métadonnées



Des possibilités illimitées...



Architecture détaillée d'un plugin « métier »



Répartition des « tâches » entre SGBD et QGIS

Un véritable écosystème

En maîtriser les différentes briques

SGBD

Modèles simples et robustes

Modularité

Utilisation totale ou partielle

Références

<http://www.postgresql.org/docs/9.1/static/tablefunc.html>

<https://docs.python.org/2/library/xml.etree.elementtree.html>

<http://www.portalsig.org/content/python-cree-automatiquement-des-fichiers-styles-qml-de-qgis-partir-de-fichiers-ou-de-shapef>

<http://doc.qt.io/qt-5/qtsql-index.html>

