# Development of a Metadata Crawler Desktop Application using Python

Project Report for Summer Internship 2015

*Submitted By*

**Aly Akhtar**
**Sachin Sharma**

Under the guidance of

**Mr. Kapil Sharma**

Scientist-D, National Informatics Centre,
Shastri Park, New Delhi

# CERTIFICATE

---

This is to certify that the project report entitled “Development of a Metadata Crawler Desktop Application using Python” is a record of bonafide work carried out by Aly Akhtar & Sachin Sharma of Faculty of Engineering & Technology, Jamia Millia Islamia under my supervision and guidance, as part of their Summer Internship 2015, at the National Informatics Centre, Shastri Park, New Delhi.

**Mr. Kapil Sharma**
Scientist-D, National Informatics Centre
Shastri Park, New Delhi

Date:
Place: New Delhi

# ACKNOWLEDGEMENTS

We take this opportunity to express our deep sense of gratitude to our guide **Mr. Kapil Sharma** for giving us this opportunity to work on this project under his supervision.

Also we would like to specially acknowledge **Mr. Pankaj Khetwal (**Scientist-B, National Informatics Centre, Shastri Park, New Delhi) for his valuable suggestions and guidance during the entire duration of the project.

**Aly Akhtar**

**Sachin Sharma**

# About NIC

National Informatics Centre (NIC) of the Department of Information Technology is providing network backbone and e-Governance support to Central Government, State Governments, UT Administrations, Districts and other Government bodies. It offers a wide range of ICT services including Nationwide Communication Network for decentralized planning, improvement in Government services and wider transparency of national and local Governments.

NIC Headquarters is based in New Delhi. At NIC Headquarters, a large number of Application Divisions exist which provide Informatics Support to the Ministries and Departments of the Central Government. NIC computer cells are located in almost all the Ministries of the Central Government and Apex Offices including the Prime Minister's Office, the Rashtrapati Bhawan and the Parliament House. Apart from this, NIC has various Resource Divisions at the Headquarters which specialize into different areas of IT and facilitate the Application Divisions as well as other NIC Centers in providing state-of-the-art services to the Govt. At the State level, NICs State/UTs Units provide informatics support to their `respective State Government and at the District level lie the NIC District Informatics Offices.

NIC has conceptualized, developed and implemented a very large number of projects for various Central and State Government Ministries, Departments and Organizations. Many of these projects are continuing projects being carried out by various divisions of NIC at New Delhi Headquarters and State/District centers throughout the country. A wide variety of projects are supported by NIC, which offer a glimpse of the multifaceted, diverse activities of this organization, touching upon all spheres of e-governance and thereby influencing the lives of millions of citizens of India.

# About PostgreSQL

PostgreSQL is a powerful, open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. It runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), and Windows. It is fully ACID compliant, has full support for foreign keys, joins, views, triggers, and stored procedures (in multiple languages). It includes most SQL: 2008 data types, including INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, and TIMESTAMP. It also supports storage of binary large objects, including pictures, sounds, or video. It has native programming interfaces for C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, among others, and exceptional documentation.

PostgreSQL prides itself in standards compliance. Its SQL implementation strongly conforms to the ANSI-SQL: 2008 standard. It has full support for subqueries (including sub selects in the FROM clause), read-committed and serializable transaction isolation levels. And while PostgreSQL has a fully relational system catalog which itself supports multiple schemas per database, its catalog is also accessible through the Information Schema as defined in the SQL standard. Data integrity features include (compound) primary keys, foreign keys with restricting and cascading updates/deletes, check constraints, unique constraints, and not null constraints. It also has a host of extensions and advanced features. Among the conveniences are auto-increment columns through sequences, and LIMIT/OFFSET allowing the return of partial result sets. PostgreSQL supports compound, unique, partial, and functional indexes which can use any of its B-tree, R-tree, hash, or GiST storage methods.

Best of all, PostgreSQL's source code is available under a liberal open source license: the PostgreSQL License. This license 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. As such, PostgreSQL is not only a powerful database system capable of running the enterprise, it is a development platform upon which to develop in-house, web, or commercial software products that require a capable RDBMS.

# 1. Project Introduction

The aim of the project was to develop a desktop application to access the metadata information of a PostgreSQL database including tables, columns, functions, procedures, views, indexes etc.

This application, which we have named “MetGeomi”, will allow the user to connect to a remote PostgreSQL database by providing the authentication details like IP, User Name, Password, Database Name etc. The software will then crawl through the database and retrieve metadata information. The information will be displayed in an easy-to-use Graphic User Interface display.

This Project Report is being written to provide a detailed and complete description of the implemented work to the intended readers. The subsequent chapters include all the information one would need to understand this piece of work.

# 2. Tools and Language

---

The whole project is written in python language i.e. both the front end and the back end. The back end is written in basic python language and it uses psycopg2 library for connecting to a PostgreSQL database and extracting the metadata from the database. The front end or the GUI is written in python with the help of pyqt4 library. The windows executable file was created with another python library pyinstaller and the whole installer was created with the inno setup compiler.

Overall the following things were used in realization of the project:

- Python programming language.
- Inno setup compiler.
- Pyinstaller library.
- Psycopg2 library.
- Pyqt4 library.

# 3. IMPLEMENTATION

The basic idea behind this application is to fetch all the metadata from a given remote database and then display it with GUI. The back-end of the application does all the working and then passes it on to the front–end of the application to print it for the user.

The back-end is defined with multiple methods, to perform the needed operations. The GUI takes the user input as the database name, password, host, port and user name which then passes it on to the backend for further operation. The first step is to create a connection with the database, if the connection exists then it proceeds to crawl the database and fetch all the items like table names, views, functions, triggers etc. and create a summary for it, otherwise it gives a message for wrong credentials.

The data that is fetched is shown in tabular form with tabs for every other category and the last tab gives the summary i.e. the total number of each items in the database. The scope of crawling is only limited to the PostgreSQL databases.

# 4. CONCLUSION

The project “Database Metadata Crawling” has been created in python and works to all its requirements. This has will help in easily fetching the metadata from a database from a remote server and displaying it for the user in a user-friendly manner.

This Project helped us in understanding the working of the PostgreSQL database and the crawling of the databases using python and gave us a chance to work on a different thing and learn something new.

Python has rich set of libraries which help us in easily connecting with the databases and even create a windows executable file and graphical user interface for a full-fledged application.

Overall, it has been a good learning experience and very informative in realization of the project.

# 5. CODE

```
#!/usr/bin/python

#Importing the required Modules
import psycopg2
import sys
from PyQt4 import QtCore, QtGui
from GUI import Ui_Dialog
from GUI2 import Ui_Dialog3
from GUI3 import Ui_Dialog4

con = None

#Connecting with the Database
def create_connection(database , user , password , host , port):
    cur_db = database
    cur_usr = user
    cur_pass = password
    cur_host = host
    cur_port = port
    try:
        con = psycopg2.connect(database = cur_db , user = cur_usr , password =
cur_pass , host = cur_host , port = cur_port)
```

```
    except:
        con = -1
    return con

#Extracting the Tables Info from database
def meta_crawl_tables(con):
    cur = con.cursor()

    tables=[]
    cur.execute("SELECT table_name FROM information_schema.tables "
                "WHERE table_schema='public' "
                "AND table_type='BASE TABLE'");
    resp = cur.fetchall()

    for row in resp:
        global tables
        tables.append(row[0])


#Extracting the Columns Info from database
def meta_crawl_columns(con , tables):
    cur = con.cursor()
    to_d_col=[]
    for table in tables:
        cur.execute("SELECT column_name,data_type,character_maximum_length "
```

```
            "FROM information_schema.columns "
            "WHERE table_name = '" + table + "'");
        resp = cur.fetchall()
        for row in resp:
            columns=[]
            global columns
            global to_d_col
            columns.append(table)
            columns.append(row[0])
            columns.append(row[1])
            columns.append(row[2])
            to_d_col.append(columns)


#DataTypeID to DataTypeName Mapping
def typeid_to_typename(con , id):
    cur = con.cursor()

    cur.execute("SELECT typname from pg_type WHERE oid = " + str(id))
    resp = cur.fetchone()
    return resp[0]


#Extracting the Functions Info from database
def meta_crawl_functions(con):
```

```
    cur = con.cursor()
    to_d_func=[]
    user_func=[]
    cur.execute("SELECT routine_name FROM information_schema.routines
WHERE specific_schema = 'public'")
    resp=cur.fetchall()
    for row in resp:
        user_func.append(row[0])



    cur.execute("SELECT proname, prorettype, pronargs, proargnames ,
proargtypes from pg_proc WHERE proname IN " + str(tuple(user_func)))
    resp = cur.fetchall()
    for row in resp:
        functions=[]
        global to_d_func
        ret_type_list=[]
        functions.append(row[0])
        functions.append(typeid_to_typename(con , row[1]))
        functions.append(row[2])
        functions.append(row[3])
        ret_arg = row[4].split()
        for item in ret_arg:
            ret_type_list.append(typeid_to_typename(con , item))
        functions.append(ret_type_list)
        to_d_func.append(functions)
```

```
#Extracting the Triggers Info from database
def meta_crawl_triggers(con):
    cur = con.cursor()
    to_d_trig=[]
    cur.execute("SELECT trigger_name, action_timing, event_manipulation,
event_object_table FROM information_schema.triggers")
    resp=cur.fetchall()
    for row in resp:
        triggers=[]
        global to_d_trig
        triggers.append(row[0])
        triggers.append(row[1])
        triggers.append(row[2])
        triggers.append(row[3])
        to_d_trig.append(triggers)


#Extracting the Views Info from database
def meta_crawl_views(con):
    cur = con.cursor()
    to_d_view=[]
    cur.execute("SELECT table_name FROM information_schema.views WHERE
table_schema = 'public'")
    resp=cur.fetchall()
    for row in resp:
```

```
        view=[]
        global to_d_view
        view.append(row[0])
        tb=[]
        cur.execute("SELECT table_name FROM
information_schema.view_table_usage WHERE view_name = '"+ str(row[0]) +"'")
        sub_resp = cur.fetchall()
        for item in sub_resp:
            tb.append(item[0])
            col=[]
        cur.execute("SELECT column_name FROM
information_schema.view_column_usage WHERE view_name = '"+ str(row[0])
+"'")
        sub_resp = cur.fetchall()
        for item in sub_resp:
            col.append(item[0])


        view.append(tb)
        view.append(col)
        to_d_view.append(view)



#Extracting the Constraints Info from database
def meta_crawl_constraints(con):
    cur = con.cursor()
    cons=[]
```

```
    to_d_cons=[]

    cur.execute("SELECT constraint_name, column_name, table_name "
                "FROM information_schema.constraint_column_usage")
    resp = cur.fetchall()

    for row in resp:
      cons=[]
      global to_d_cons
      cons.append(row[0])
      cons.append(row[1])
      cons.append(row[2])
      to_d_cons.append(cons)

#Extracting the Indexes Info from database
def meta_crawl_indexes(con):
    cur = con.cursor();
    ind=[]
    to_d_ind=[]
    cur.execute("SELECT
i.relname,t.relname,array_to_string(array_agg(a.attname), ', ') "
                "FROM pg_class t, pg_class i, pg_index ix, pg_attribute a "
                "WHERE t.oid = ix.indrelid and i.oid = ix.indexrelid and a.attrelid =
t.oid and a.attnum = ANY(ix.indkey) and t.relkind = 'r' and t.relname like 'test%' "
                "GROUP BY t.relname, i.relname "
                "ORDER BY t.relname, i.relname")
```

```
    resp = cur.fetchall()

    for row in resp:
      ind=[]
      global to_d_ind
      ind.append(row[0])
      ind.append(row[1])
      ind.append(row[2])
      to_d_ind.append(ind)

#Creating GUI
class MyDialog4(QtGui.QWidget):
    def __init__(self, parent=None):
        QtGui.QWidget.__init__(self, parent)
        self.ui = Ui_Dialog4()
        self.ui.setupUi(self)
        self.ui.pushButton_6.clicked.connect(self.NEXT)

    def NEXT(self):
        myapp4.hide()
        myapp.show()

#creating credentials page
class MyDialog(QtGui.QWidget):
    def __init__(self, parent=None):
```

```
        QtGui.QWidget.__init__(self, parent)
        self.ui = Ui_Dialog()
        self.ui.setupUi(self)
        self.ui.pushButton.clicked.connect(self.NEXT)
        self.ui.pushButton_2.clicked.connect(self.CANCEL)

    def retranslateUi(self, Dialog):
        Dialog.setWindowTitle(QtGui.QApplication.translate("Dialog", "Dialog",
None, QtGui.QApplication))
        self.pushButton.setText(QtGui.QApplication.translate("Dialog", "OKAY",
None, QtGui.QApplication))
        self.label.setText(QtGui.QApplication.translate("Dialog", "WRONG
DETAILS GIVEN", None, QtGui.QApplication))



    def NEXT(self):
        # text = self.ui.textEdit.toPlainText()
        database_name = self.ui.lineEdit.text()
        user_name = self.ui.lineEdit_2.text()
        password = self.ui.lineEdit_3.text()
        ip_address = self.ui.lineEdit_4.text()
        port_number = self.ui.lineEdit_5.text()
        var = create_connection(ip_address , user_name , password , database_name ,
port_number)
        global var
        if (var != -1):
            myapp2 = Ui_Dialog2()
```

```
            myapp2.setupUi(self)
            myapp2.pushButton_5.clicked.connect(self.BACK)
        elif(var == -1):
            myapp3.show()
            myapp.hide()

    def BACK(self):
        self.close()

    def CANCEL(self):
        self.close()
#Creating pop up for wrong credentials
class MyDialog2(QtGui.QWidget):
    def __init__(self, parent=None):
        QtGui.QWidget.__init__(self, parent)
        self.ui = Ui_Dialog3()
        self.ui.setupUi(self)
        self.ui.pushButton_6.clicked.connect(self.BACK)

    def BACK(self):
        myapp3.hide()
        myapp.show()

#creating display for the Database Details
class Ui_Dialog2(object):
```

```
def setupUi(self, Dialog):
    meta_crawl_tables(var)
    meta_crawl_columns(var , tables)
    meta_crawl_functions(var)
    meta_crawl_triggers(var)
    meta_crawl_views(var)
    meta_crawl_constraints(var)
    meta_crawl_indexes(var)
    Dialog.setObjectName("Dialog")
    Dialog.setGeometry(350, 200, 700, 400)
    Dialog.setWindowTitle('MetGeomi')
    Dialog.setWindowIcon(QtGui.QIcon('crawl.ico'))
    self.verticalLayout = QtGui.QVBoxLayout(Dialog)
    self.verticalLayout.setObjectName("verticalLayout")
    self.tabWidget = QtGui.QTabWidget(Dialog)
    self.tabWidget.setObjectName("tabWidget")
    self.tab = QtGui.QWidget()
    self.tab.setObjectName("tab")
    self.horizontalLayout = QtGui.QHBoxLayout(self.tab)
    self.horizontalLayout.setObjectName("horizontalLayout")
    self.tableWidget_7 = QtGui.QTableWidget(self.tab)
    self.tableWidget_7.setGridStyle(QtCore.Qt.NoPen)
    self.tableWidget_7.setObjectName("tableWidget_7s")
    self.tableWidget_7.setColumnCount(1)
    item = QtGui.QTableWidgetItem()
```

```
self.tableWidget_7.setRowCount(len(tables))
self.tableWidget_7.setHorizontalHeaderItem(0, item)
item = QtGui.QTableWidgetItem()
item = self.tableWidget_7.horizontalHeaderItem(0)
item.setText("TABLES")
for i, row in enumerate(tables):
    item = QtGui.QTableWidgetItem("%s"% row)
    self.tableWidget_7.setItem(i, 0, item)
self.horizontalLayout.addWidget(self.tableWidget_7)
self.tableWidget_7.setColumnWidth(0, 160);
self.tableWidget_7.setColumnWidth(1, 160);
self.tableWidget_7.setColumnWidth(2, 180);
self.tableWidget_7.setColumnWidth(3, 120);
self.tabWidget.addTab(self.tab, "")
self.tab_2 = QtGui.QWidget()
self.tab_2.setObjectName("tab_2")
self.horizontalLayout_2 = QtGui.QHBoxLayout(self.tab_2)
self.horizontalLayout_2.setObjectName("horizontalLayout_2")
self.tableWidget = QtGui.QTableWidget(self.tab_2)
# self.tableWidget.setGridStyle(QtCore.Qt.NoPen)
self.tableWidget.setObjectName("tableWidget")
self.tableWidget.setColumnCount(4)
item = QtGui.QTableWidgetItem()
self.tableWidget.setRowCount(len(to_d_col))
self.tableWidget.setHorizontalHeaderItem(0, item)
```

```
item = QtGui.QTableWidgetItem()
self.tableWidget.setHorizontalHeaderItem(1, item)
item = QtGui.QTableWidgetItem()
self.tableWidget.setHorizontalHeaderItem(2, item)
item = QtGui.QTableWidgetItem()
self.tableWidget.setHorizontalHeaderItem(3, item)
item = QtGui.QTableWidgetItem()
item = self.tableWidget.horizontalHeaderItem(0)
item.setText("TABLE")
item = self.tableWidget.horizontalHeaderItem(1)
item.setText("COLUMN")
item = self.tableWidget.horizontalHeaderItem(2)
item.setText("TYPE")
item = self.tableWidget.horizontalHeaderItem(3)
item.setText("LENGTH")
for i, row in enumerate(to_d_col):
  for j, value in enumerate(row):
    item = QtGui.QTableWidgetItem("%s"% value)
    self.tableWidget.setItem(i, j, item)
self.horizontalLayout_2.addWidget(self.tableWidget)
self.tableWidget.setColumnWidth(0, 160);
self.tableWidget.setColumnWidth(1, 160);
self.tableWidget.setColumnWidth(2, 180);
self.tableWidget.setColumnWidth(3, 120);
self.tabWidget.addTab(self.tab_2, "")
```

```
self.tab_3 = QtGui.QWidget()
self.tab_3.setObjectName("tab_3")
self.horizontalLayout_3 = QtGui.QHBoxLayout(self.tab_3)
self.horizontalLayout_3.setObjectName("horizontalLayout_3")
self.tableWidget_2 = QtGui.QTableWidget(self.tab_3)
self.tableWidget_2.setObjectName("tableWidget_2")
self.tableWidget_2.setColumnCount(5)
item = QtGui.QTableWidgetItem()
self.tableWidget_2.setRowCount(len(to_d_func))
self.tableWidget_2.setHorizontalHeaderItem(0, item)
item = QtGui.QTableWidgetItem()
self.tableWidget_2.setHorizontalHeaderItem(1, item)
item = QtGui.QTableWidgetItem()
self.tableWidget_2.setHorizontalHeaderItem(2, item)
item = QtGui.QTableWidgetItem()
self.tableWidget_2.setHorizontalHeaderItem(3, item)
item = QtGui.QTableWidgetItem()
self.tableWidget_2.setHorizontalHeaderItem(4, item)
item = QtGui.QTableWidgetItem()
item = self.tableWidget_2.horizontalHeaderItem(0)
item.setText("FUNCTION")
item = self.tableWidget_2.horizontalHeaderItem(1)
item.setText("TYPE")
item = self.tableWidget_2.horizontalHeaderItem(2)
item.setText("ARGUMENTS")
```

```
item = self.tableWidget_2.horizontalHeaderItem(3)
item.setText("ARGUMENT NAME")
item = self.tableWidget_2.horizontalHeaderItem(4)
item.setText("ARGUMENT TYPE")
for i, row in enumerate(to_d_func):
  for j, value in enumerate(row):
    item = QtGui.QTableWidgetItem("%s"% value)
    self.tableWidget_2.setItem(i, j, item)
self.horizontalLayout_3.addWidget(self.tableWidget_2)
self.tableWidget_2.setColumnWidth(0, 150);
self.tableWidget_2.setColumnWidth(1, 100);
self.tableWidget_2.setColumnWidth(2, 100);
self.tableWidget_2.setColumnWidth(3, 140);
self.tableWidget_2.setColumnWidth(4, 140);
self.tabWidget.addTab(self.tab_3, "")
self.tab_4 = QtGui.QWidget()
self.tab_4.setObjectName("tab_4")
self.horizontalLayout_4 = QtGui.QHBoxLayout(self.tab_4)
self.horizontalLayout_4.setObjectName("horizontalLayout_4")
self.tableWidget_3 = QtGui.QTableWidget(self.tab_4)
self.tableWidget_3.setObjectName("tableWidget_3")
self.tableWidget_3.setColumnCount(4)
item = QtGui.QTableWidgetItem()
self.tableWidget_3.setRowCount(len(to_d_trig))
self.tableWidget_3.setHorizontalHeaderItem(0, item)
```

```
item = QtGui.QTableWidgetItem()
self.tableWidget_3.setHorizontalHeaderItem(1, item)
item = QtGui.QTableWidgetItem()
self.tableWidget_3.setHorizontalHeaderItem(2, item)
item = QtGui.QTableWidgetItem()
self.tableWidget_3.setHorizontalHeaderItem(3, item)
item = QtGui.QTableWidgetItem()
item = self.tableWidget_3.horizontalHeaderItem(0)
item.setText("TRIGGER")
item = self.tableWidget_3.horizontalHeaderItem(1)
item.setText("ACTION")
item = self.tableWidget_3.horizontalHeaderItem(2)
item.setText("EVENT")
item = self.tableWidget_3.horizontalHeaderItem(3)
item.setText("TABLE")
for i, row in enumerate(to_d_trig):
  for j, value in enumerate(row):
    item = QtGui.QTableWidgetItem("%s"% value)
    self.tableWidget_3.setItem(i, j, item)
self.horizontalLayout_4.addWidget(self.tableWidget_3)
self.tableWidget_3.setColumnWidth(0, 170);
self.tableWidget_3.setColumnWidth(1, 160);
self.tableWidget_3.setColumnWidth(2, 180);
self.tableWidget_3.setColumnWidth(3, 120);
self.tabWidget.addTab(self.tab_4, "")
```

```
self.tab_5 = QtGui.QWidget()
self.tab_5.setObjectName("tab_5")
self.horizontalLayout_5 = QtGui.QHBoxLayout(self.tab_5)
self.horizontalLayout_5.setObjectName("horizontalLayout_5")
self.tableWidget_4 = QtGui.QTableWidget(self.tab_5)
self.tableWidget_4.setObjectName("tableWidget_4")
self.tableWidget_4.setColumnCount(3)
item = QtGui.QTableWidgetItem()
self.tableWidget_4.setRowCount(len(to_d_view))
self.tableWidget_4.setHorizontalHeaderItem(0, item)
item = QtGui.QTableWidgetItem()
self.tableWidget_4.setHorizontalHeaderItem(1, item)
item = QtGui.QTableWidgetItem()
self.tableWidget_4.setHorizontalHeaderItem(2, item)
item = QtGui.QTableWidgetItem()
item = self.tableWidget_4.horizontalHeaderItem(0)
item.setText("VIEW")
item = self.tableWidget_4.horizontalHeaderItem(1)
item.setText("TABLES")
item = self.tableWidget_4.horizontalHeaderItem(2)
item.setText("COLUMNS")
for i, row in enumerate(to_d_view):
  for j, value in enumerate(row):
    item = QtGui.QTableWidgetItem("%s"% value)
    self.tableWidget_4.setItem(i, j, item)
```

```
self.horizontalLayout_5.addWidget(self.tableWidget_4)
self.tableWidget_4.setColumnWidth(0, 200);
self.tableWidget_4.setColumnWidth(1, 220);
self.tableWidget_4.setColumnWidth(2, 210);
self.tabWidget.addTab(self.tab_5, "")
self.tab_7 = QtGui.QWidget()
self.tab_7.setObjectName("tab_7")
self.horizontalLayout_7 = QtGui.QHBoxLayout(self.tab_7)
self.horizontalLayout_7.setObjectName("horizontalLayout_7")
self.tableWidget_5 = QtGui.QTableWidget(self.tab_7)
self.tableWidget_5.setObjectName("tableWidget_5")
self.tableWidget_5.setColumnCount(3)
item = QtGui.QTableWidgetItem()
self.tableWidget_5.setRowCount(len(to_d_ind))
self.tableWidget_5.setHorizontalHeaderItem(0, item)
item = QtGui.QTableWidgetItem()
self.tableWidget_5.setHorizontalHeaderItem(1, item)
item = QtGui.QTableWidgetItem()
self.tableWidget_5.setHorizontalHeaderItem(2, item)
item = QtGui.QTableWidgetItem()
item = self.tableWidget_5.horizontalHeaderItem(0)
item.setText("INDEX")
item = self.tableWidget_5.horizontalHeaderItem(1)
item.setText("TABLES")
item = self.tableWidget_5.horizontalHeaderItem(2)
```

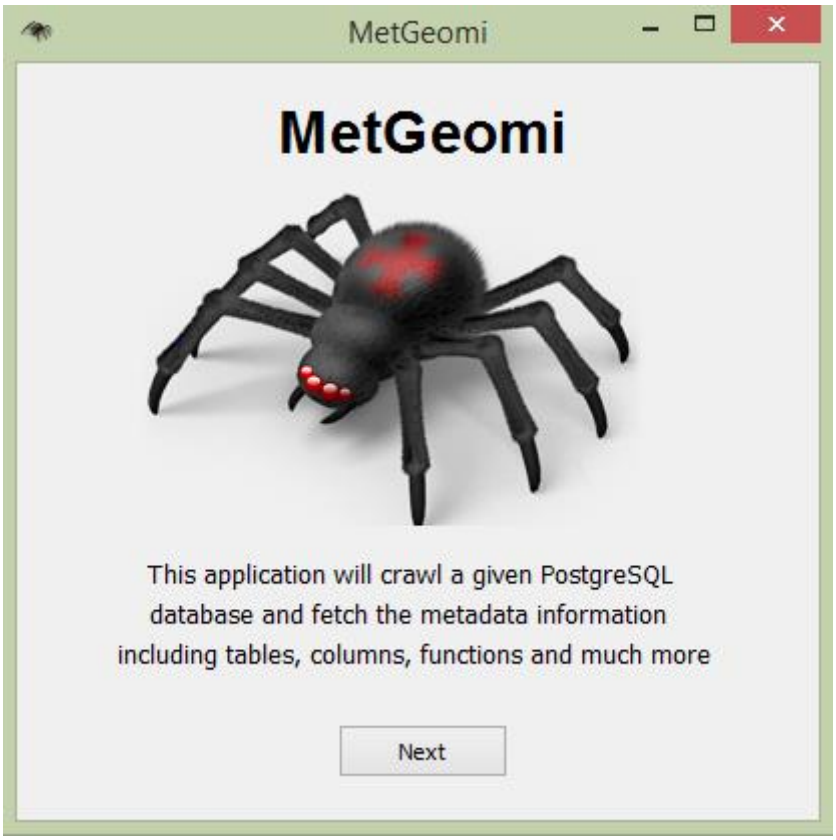
```
item.setText("COLUMNS")
for i, row in enumerate(to_d_ind):
  for j, value in enumerate(row):
    item = QtGui.QTableWidgetItem("%s"% value)
    self.tableWidget_5.setItem(i, j, item)
self.horizontalLayout_7.addWidget(self.tableWidget_5)
self.tableWidget_5.setColumnWidth(0, 200);
self.tableWidget_5.setColumnWidth(1, 220);
self.tableWidget_5.setColumnWidth(2, 210);
self.tabWidget.addTab(self.tab_7, "")
self.tab_8 = QtGui.QWidget()
self.tab_8.setObjectName("tab_8")
self.horizontalLayout_8 = QtGui.QHBoxLayout(self.tab_8)
self.horizontalLayout_8.setObjectName("horizontalLayout_8")
self.tableWidget_6 = QtGui.QTableWidget(self.tab_8)
self.tableWidget_6.setObjectName("tableWidget_6")
self.tableWidget_6.setColumnCount(3)
item = QtGui.QTableWidgetItem()
self.tableWidget_6.setRowCount(len(to_d_cons))
self.tableWidget_6.setHorizontalHeaderItem(0, item)
item = QtGui.QTableWidgetItem()
self.tableWidget_6.setHorizontalHeaderItem(1, item)
item = QtGui.QTableWidgetItem()
self.tableWidget_6.setHorizontalHeaderItem(2, item)
item = QtGui.QTableWidgetItem()
```

```
        item = self.tableWidget_6.horizontalHeaderItem(0)
        item.setText("VIEW")
        item = self.tableWidget_6.horizontalHeaderItem(1)
        item.setText("TABLES")
        item = self.tableWidget_6.horizontalHeaderItem(2)
        item.setText("COLUMNS")
        for i, row in enumerate(to_d_cons):
          for j, value in enumerate(row):
            item = QtGui.QTableWidgetItem("%s"% value)
            self.tableWidget_6.setItem(i, j, item)
        self.horizontalLayout_8.addWidget(self.tableWidget_6)
        self.tableWidget_6.setColumnWidth(0, 200);
        self.tableWidget_6.setColumnWidth(1, 220);
        self.tableWidget_6.setColumnWidth(2, 210);
        self.tabWidget.addTab(self.tab_8, "")
        self.tab_6 = QtGui.QWidget()
        self.tab_6.setObjectName("tab_6")
        self.horizontalLayout_6 = QtGui.QHBoxLayout(self.tab_6)
        self.horizontalLayout_6.setObjectName("horizontalLayout_6")
        self.listWidget_6 = QtGui.QListWidget(self.tab_6)
        self.listWidget_6.setObjectName("listWidget_6")
        item = QtGui.QListWidgetItem("\nTotal Number of Tables : %i"%
len(tables))
        self.listWidget_6.addItem(item)
        item = QtGui.QListWidgetItem("\nTotal Number of Columns : %i"%
len(to_d_col))
```

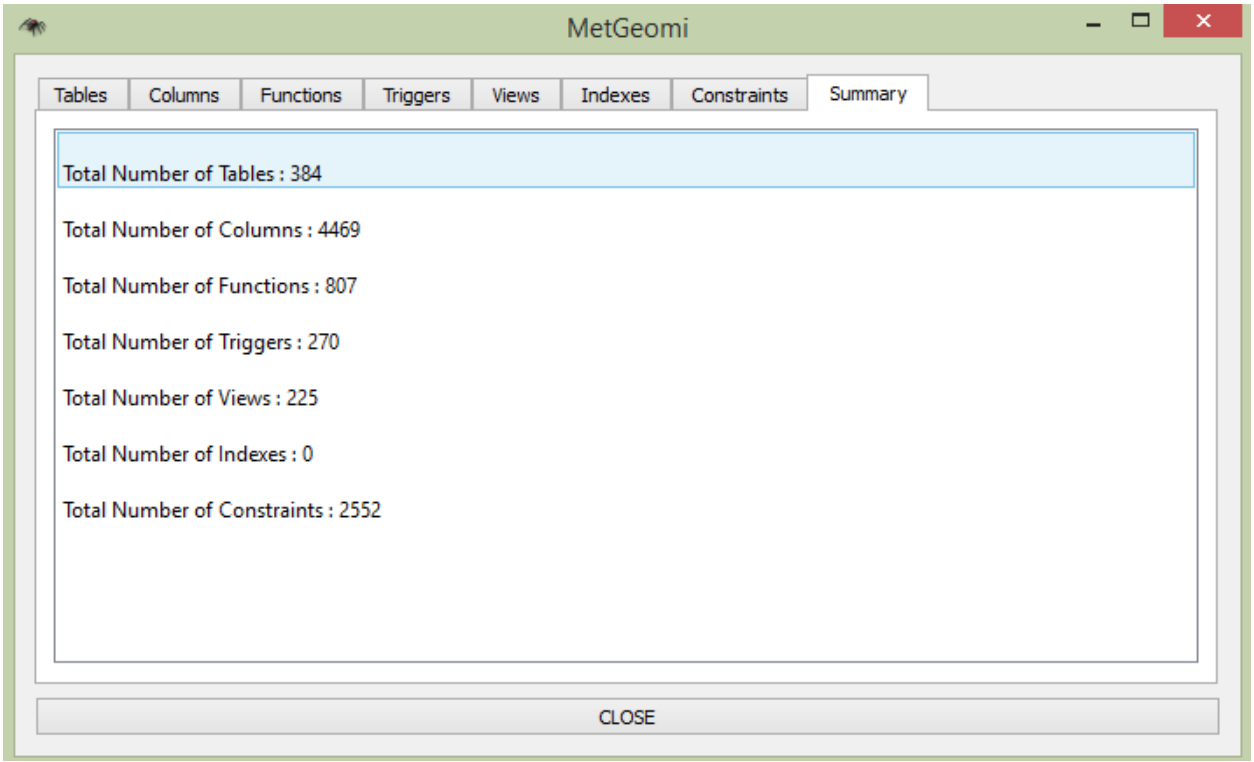
```
        self.listWidget_6.addItem(item)
        item = QtGui.QListWidgetItem("\nTotal Number of Functions : %i"%
len(to_d_func))
        self.listWidget_6.addItem(item)
        item = QtGui.QListWidgetItem("\nTotal Number of Triggers : %i"%
len(to_d_trig))
        self.listWidget_6.addItem(item)
        item = QtGui.QListWidgetItem("\nTotal Number of Views : %i"%
len(to_d_view))
        self.listWidget_6.addItem(item)
        item = QtGui.QListWidgetItem("\nTotal Number of Indexes : %i"%
len(to_d_ind))
        self.listWidget_6.addItem(item)
        item = QtGui.QListWidgetItem("\nTotal Number of Constraints : %i"%
len(to_d_cons))
        self.listWidget_6.addItem(item)
        self.horizontalLayout_6.addWidget(self.listWidget_6)
        self.tabWidget.addTab(self.tab_6, "")
        self.verticalLayout.addWidget(self.tabWidget)
        self.pushButton_5 = QtGui.QPushButton(Dialog)
        self.pushButton_5.setObjectName("pushButton_5")
        self.verticalLayout.addWidget(self.pushButton_5)


        self.retranslateUi(Dialog)
        self.tabWidget.setCurrentIndex(7)
        QtCore.QMetaObject.connectSlotsByName(Dialog)
```

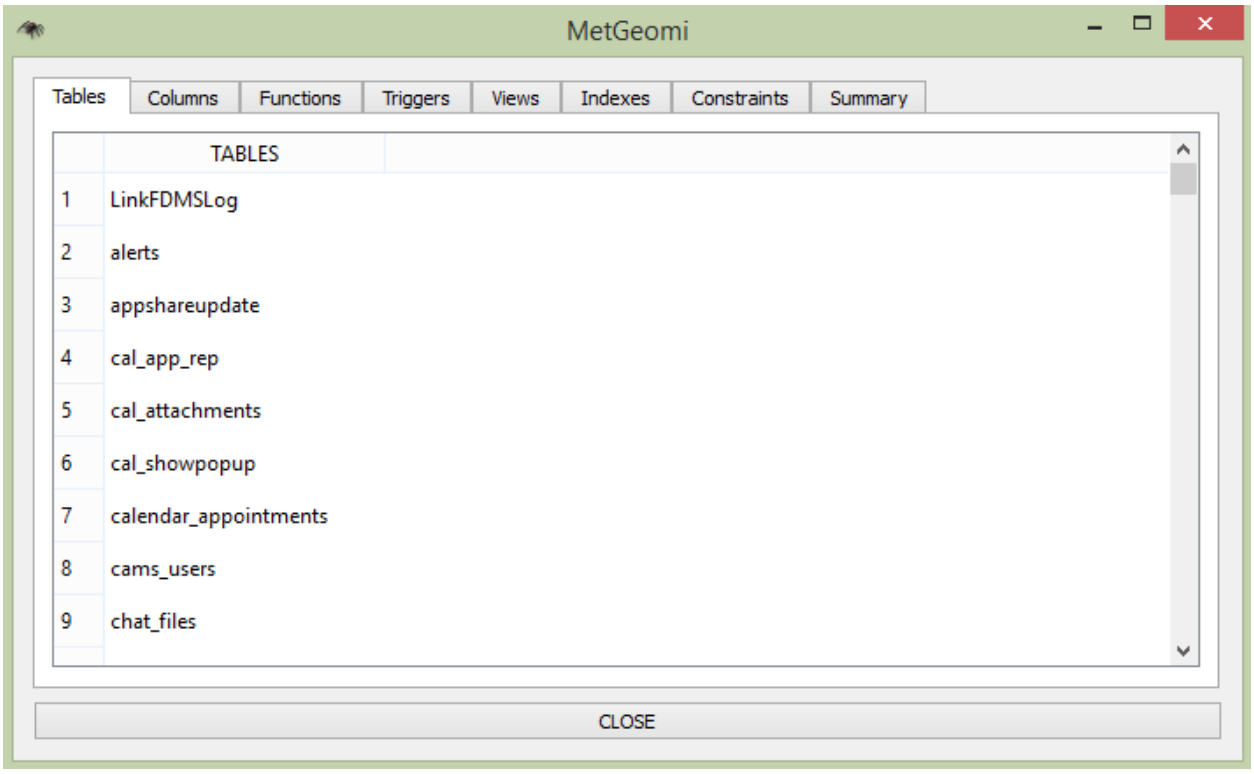
```
    def retranslateUi(self, Dialog):
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab),
QtGui.QApplication.translate("Dialog", "Tables", None,
QtGui.QApplication.UnicodeUTF8))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_2),
QtGui.QApplication.translate("Dialog", "Columns", None,
QtGui.QApplication.UnicodeUTF8))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_3),
QtGui.QApplication.translate("Dialog", "Functions", None,
QtGui.QApplication.UnicodeUTF8))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_4),
QtGui.QApplication.translate("Dialog", "Triggers", None,
QtGui.QApplication.UnicodeUTF8))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_5),
QtGui.QApplication.translate("Dialog", "Views", None,
QtGui.QApplication.UnicodeUTF8))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_6),
QtGui.QApplication.translate("Dialog", "Summary", None,
QtGui.QApplication.UnicodeUTF8))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_7),
QtGui.QApplication.translate("Dialog", "Indexes", None,
QtGui.QApplication.UnicodeUTF8))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_8),
QtGui.QApplication.translate("Dialog", "Constraints", None,
QtGui.QApplication.UnicodeUTF8))
        self.pushButton_5.setText(QtGui.QApplication.translate("Dialog", "CLOSE",
None, QtGui.QApplication.UnicodeUTF8))


#Main
if __name__ == "__main__":
    app = QtGui.QApplication(sys.argv)
```

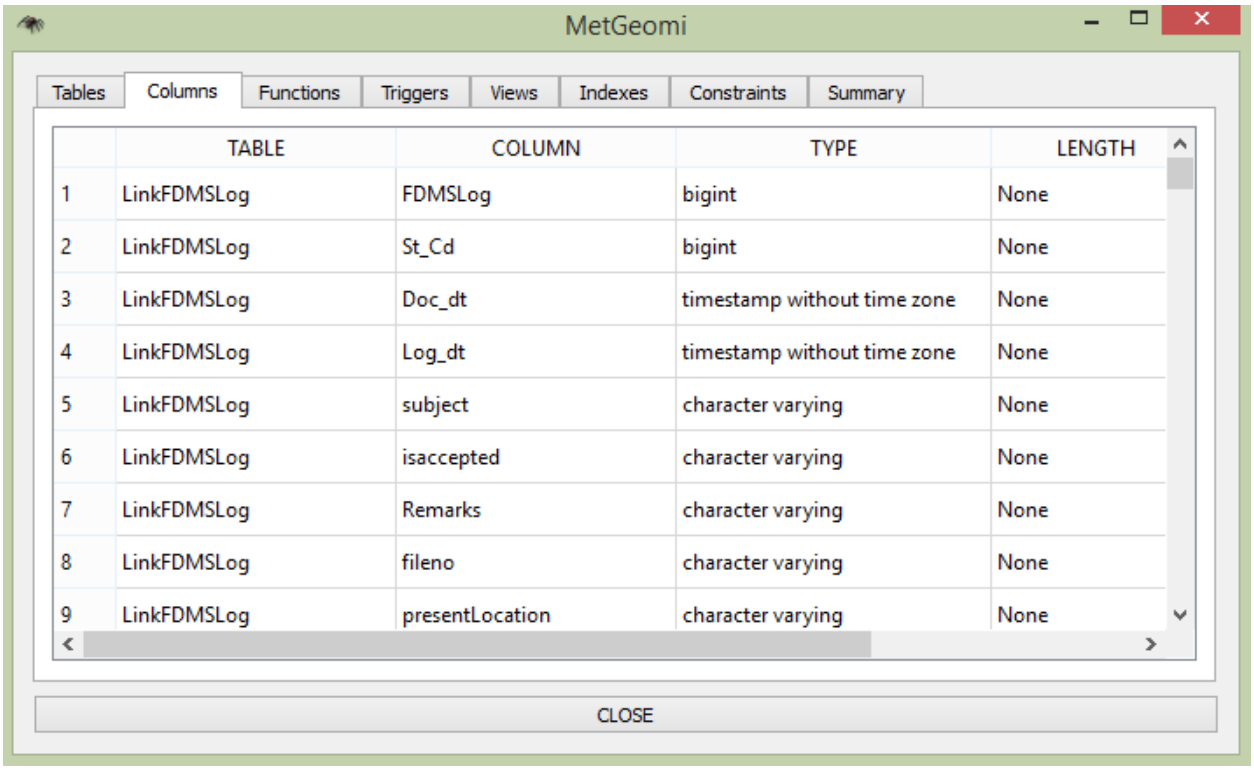
```
myapp = MyDialog()
myapp3 = MyDialog2()
myapp4 = MyDialog4()
myapp4.show()
sys.exit(app.exec_())
```




MetGeomi
Enter IP Addres
localhost
Enter user
postgres
Enter Password
Enter Database
nic_coal
Enter Port
5432
Cancel
NEXT


MetGeomi
Tables
Columns
Functions
Triggers
Views
Indexes
Constraints
Summary
Total Number of Tables : 384
Total Number of Columns : 4469
Total Number of Functions : 807
Total Number of Triggers : 270
Total Number of Views : 225
Total Number of Indexes : 0
Total Number of Constraints : 2552
CLOSE


MetGeomi
Tables
Columns
Functions
Triggers
Views
Indexes
Constraints
Summary
TABLES
1 LinkFDMSLog
2 alerts
3 appshareupdate
4 cal_app_rep
5 cal_attachments
6 cal_showpopup
7 calendar_appointments
8 cams_users
9 chat_files
CLOSE


MetGeomi
Tables
Columns
Functions
Triggers
Views
Indexes
Constraints
Summary
TABLE COLUMN TYPE LENGTH
1 LinkFDMSLog FDMSLog bigint None
2 LinkFDMSLog St_Cd bigint None
3 LinkFDMSLog Doc_dt timestamp without time zone None
4 LinkFDMSLog Log_dt timestamp without time zone None
5 LinkFDMSLog subject character varying None
6 LinkFDMSLog isaccepted character varying None
7 LinkFDMSLog Remarks character varying None
8 LinkFDMSLog fileno character varying None
9 LinkFDMSLog presentLocation character varying None
CLOSE

# 6.FUTURE SCOPE

Though this application is completed according to the requirements, it can be further developed and very interesting features can be added ti it, like:

After the crawled data is displayed, an option can be given to the user to save this data into a new database, so that when changes are made to the database then it can be compared with the saved version of it and the differences can be found out- basically it will be the comparison between the different version of a database.

In the graphical input the saved database can be given in a dropdown list for faster access, then it can ask for the option of comparison or just the data viewing from the user.

# 7. REFERENCES


1. **http://zetcode.com/gui/pyqt4/**
2. **https://mborgerson.com/creating-an-executable-from-a-python-script**
3. **https://pythonspot.com/building-an-application-gui-with-pyqt-beginners-tutorial/**
4. **https://wiki.postgresql.org/wiki/Psycopg2_Tutorial**
5. **http://www.py2exe.org/index.cgi/Py2exeAndPyQt**
6. **http://www.py2exe.org/index.cgi/ListOfOptions**