dbwidgets Documentation

Release 0.0.1

Ilker Mustafa Manap

Mar 03, 2020

CONTENTS:

CHAPTER

ONE

INTRODUCTION

Dbwidgets module has two group of classes.

- Database Classes
- GUI Classes

CHAPTER

TWO

DATABASE CLASSES

First group is for database abstraction. We are generating a database model which includes tables and columns, and also primary key and foreign key information from database's internal tables. These information is held in a DB class, which later used by ui components.

By providing same object representation of different databases, ui part becomes database vendor agnostic.

Currently, SQLite and Postgresql databases are implemented.

2.1 DB

class dbwidgets.**DB**(*host=None*, *port=None*, *dbname=None*, *filename=None*)

This class represents the base Database class. New classes for individual databases like sqlite or postgresql inherits this class.

host

Host name of the server that database is running. None for SQLite. Default is None

Type str

port

Port number for the database service. None for SQLite. Default is None

Type int

dbname

Name of the database

Type str

tables

Dictionary of tables in the database. Key value is table name.

Type dict

connection

Connection handle to database.

Type Connection handle

filename

Name of the database file. Valid for SQLite. Default is None

Type str

execute(*query_string*)

Execute a query given by query_string, using the cursor provided.

Parameters **query_string** (*str*) – Query to execute

Returns **List of records** – List of records

Return type list

record(*tablename*, *pkey_column*, *pkey_value*)
Retrieve a record from given tablename, using given primary key column and value.

SELECT * FROM tablename WHERE pkey_column = pkey_value

Parameters

- **tablename** (*str*) – Name of the table
- **pkey_column** (*str*) – Name of the primary key column
- **pkey_value** (*variable*) – Value for the pkey_column

Returns **Record** – Returns a row from table.

Return type list

report()
Print the extracted database schema to standart output

2.1.1 Example

```
from dbwidgets import DBSQLite

db = DBSQLite("test.db")
db.extract()

customer = db.record("customers", "id", 123)
```

2.2 Table

class dbwidgets.**Table**(*tablename*)
This class represents a database table.

These classes are constructed automatically by DB class.

name
Name of the table

Type str

columns
Columns of the table as a dict. Column name as key values.

Type dict

addColumn(*column*)

Parameters **column** (Column) – (Column) Add a column object to table. Parameter must be a Column object This method is not called directly. It is called from DB class when extracting the database schema automatically.

freeform_query(*cursor*, *query_string*)
Execute a freeform query given by query_string, using the cursor provided.

Parameters

- **cursor** (*database cursor*) – Cursor to execute the query
- **query_string** (*str*) – Query to execute

Returns **list** – List of records

Return type list

query(*cur*, *condition=None*)

Execute a query, using the cursor provided. Default query is

SELECT * FROM tablename

If a condition is given, it will be added at the end of the query. Default query with condition must provide a valid sql sentence.

Parameters

- **cursor** (*database cursor*) – Cursor to execute the query
- **condition** (*str or None*) – Conditions to append to the end of the query.

Returns **List of records** – List of records

Return type list

tbprint()

Print a report of the table, lists column descriptions.

2.3 Column

class dbwidgets.**Column**(*name*, *datatype*, *primary_key=False*, *default=None*)

This class represents a column in a table.

name

Name of the column

Type str

datatype

Data type of the column

Type str

default

Default value for the column, default is None

Type Variable

primary_key

True if the column is primary key

Type Boolean

foreign_key_table

Name of the foreign key table

Type str

foreign_key_column

Name of the column that is set as foreign key.

Type str

foreign_key_join_column
Name of the column to use when querying the table.

Type str

join_type
Type of the join to use when querying the table. Either INNER or OUTER.

Type str

addForeignKey(*tablename*, *columnname*, *join_column=None*, *join_type='INNER'*)
Set this column as foreign key. Currently we can't extract the join column from database.

Parameters

- **tablename** (*str*) – Other table's name
- **columnname** (*str*) – Column name in other table
- **join_column** (*str, optional*) – Column name to use for constructing the query to include the column from the other table
- **join_type** (*str, optional*) – Join type for the query, either INNER or OUTER

setJoinColumn(*join_column*, *join_type='INNER'*)
Set join column for foreign key.

Parameters

- **join_column** (*str*) – Column name to use for constructing the query to include the column from the other table
- **join_type** (*str*) – Join type for the query, either INNER or OUTER

setPrimary()
Set primary_key value to True, making this column a primary key.

CHAPTER

THREE

GUI CLASSES

3.1 DBCombobox

class dbwidgets.widgets.**DBComboBox**(*parent*, *db*, *tablename*, *textcolumn*, *idcolumn*, *default_id=None*)

parent
Parent widget for the combobox

Type QWidget

db
Database to connect to.

Type DB object

table
Name of the table

Type str

textcolumn
Name of the column to display

Type str

idcolumn
Name of the column to emit via signal as id.

Type str

default_id
Value of idcolumn to set as selected record.

Type Variable, optional

Returns **DBComboBox** – DBComboBox widget

Return type dbwidgets.DBComboBox

fill()
Fills the combobox from table rows.

idxChanged()
If another item is selected, the id of that column emitted via self.signalMasterId.

refill(*obj*)
Reset the contents of the combobox filtering with obj, which send by another widget.

Parameters **obj** – Emitted via another widget

setMaster(*otherwidget*, *mycolumn_name*)
Sets the master widget. The items in the combobox will be filtered with the value comes from the master widget's signalMasterId signal.

Raises exception if

- There is no foreign key in the current table,
- Foreign key table name is not the same as master table name
- Foreign key column name is not present inside master table.

Parameters

- **otherwidget** – Master widget that holds the master table.
- **mycolumn_name** – Column name in the detail table.

3.2 DBNavigatorWidget

class dbwidgets.widgets.**DBNavigatorWidget**(*parent*, *db*, *tablename*)
DBNavigatorWidget will provide a compound widget to display/edit/delete individual rows from tables.

3.3 DBTableWidget

class dbwidgets.widgets.**DBTableWidget**(*parent*, *db*, *tablename*, *default_id=None*)
DBTableWidget provides a QTableWidget with a table.

parent
The parent widget on user interface to put the DBTableWidget on.

Type QWidget

db
Database to connect to

Type DB object

table
Name of the table to display on DBTableWidget

Type str

dataquery
Default SQL query to fill the DBTableWidget

Type str

current_id
The selected row's primary key value

Type Variable

check_row(*x*, *y*)
Check if the selected row is changed. Set selected_id attribute to that row's primary key value

This method is invoked via cellClicked signal. It is not expected to call it from application.

refill (*obj*)
If a master widget is defined for this widget, change the contents of the DBTableWidget using the foreign key from master widget.

Parameters **obj** (*Variable*) – Foreign key value emitted from master widget.

setMaster (*otherwidget*, *mycolumn_name*, *other_table_column_to_display=None*)
Sets the master widget. The values in the table widget will be filtered with the value comes from the master widget's signalMasterId signal.

Raises exception if

- There is no foreign key in the current table,
- Foreign key table name is not the same as master table name
- Foreign key column name is not present inside master table.

Parameters

- **otherwidget** (*One of the DBWidgets*) – Master widget that holds the master table.
- **mycolumn_name** (*str*) – Column name in the detail table.

3.4 Example

In the example below, the database has two tables: city and district:

```
CREATE TABLE city (
   id INTEGER NOT NULL,
   name VARCHAR,
   PRIMARY KEY (id)
);

CREATE TABLE district (
   id INTEGER NOT NULL,
   name VARCHAR,
   city_id INTEGER,
   PRIMARY KEY (id),
   FOREIGN KEY(city_id) REFERENCES city (id)
);
```

City and district tables are connected with one to many relationship. One city has many districts.

```
import  sys
from dbwidgets import DBSQLite
from dbwidgets.widgets import DBComboBox, DBTableWidget, DBNavigatorWidget
from PySide2.QtWidgets import QApplication, QDialog
from ui_test import Ui_Dialog

class MainWindow(QDialog, Ui_Dialog):
    def __init__(self, app=None):
        super(MainWindow, self).__init__()
        self.app = app
        self.db = DBSQLite("test.db")
        self.db.extract()
        self.setupUi(self)
```

(continues on next page)

(continued from previous page)

```
14          self.city = DBComboBox(self.widget1, self.db, "city", "name", "id", 34)
15          self.district = DBComboBox(self.widget2, self.db, "district", "name", "id")
16          self.district.setMaster(self.city, "city_id")
17          self.districtlist = DBTableWidget(self.widget3, self.db, "district")
18          self.districtlist.setMaster(self.city, "city_id")
19          self.citylist = DBTableWidget(self.widget4, self.db, "city")
20          self.districtlist.setMaster(self.citylist, "city_id")
21          self.show()
22
23  if __name__ == "__main__":
24      app = QApplication(sys.argv)
25      mainWin = MainWindow(app)
26      ret = app.exec_()
27      app.exit()
28      sys.exit(ret)
```

- Line 11, we create the database object by giving connection parameters. For sqlite, file name is sufficient.
- Line 12, call extract() to create the table objects inside database object.
- Line 14, place combobox widget for city table. Here, self.widget1 is the placeholder on ui definition. self.db is the database, “city” is the name of the table, “name” is the column for displaying inside combobox. 34 is the optional default value of the primary key for the combobox to display first.
- Line 15, place the combobox for “district” table. Similar parameters as above.
- Line 16, now we will create a master detail connection between these two comboboxes. We will call detail table’s setMaster method. self.city is the widget to use as master, “city_id” is the column name on the detail table, in our example, “district” table’s “city_id” column.

With three lines, we have two comboboxes connected with master-detail relationship.

- Line 17 for creating a DBTableWidget for “district” table.
- Line 18 to connect “district” dbtablewidget to city combobox
- Line 19 to create a dbtablewidget for “city” table.
- Line 20 to connect “district” dbtablewidget to “city” dbtablewidget

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

T