Лабораторная работа №8: Создание визуального интерфейса для базы данных

Техническое задание:

Создать оконное приложение позволяющее редактировать базу данных с расписанием Вашей группы.

Минимальные требования к разрабатываемой системе:

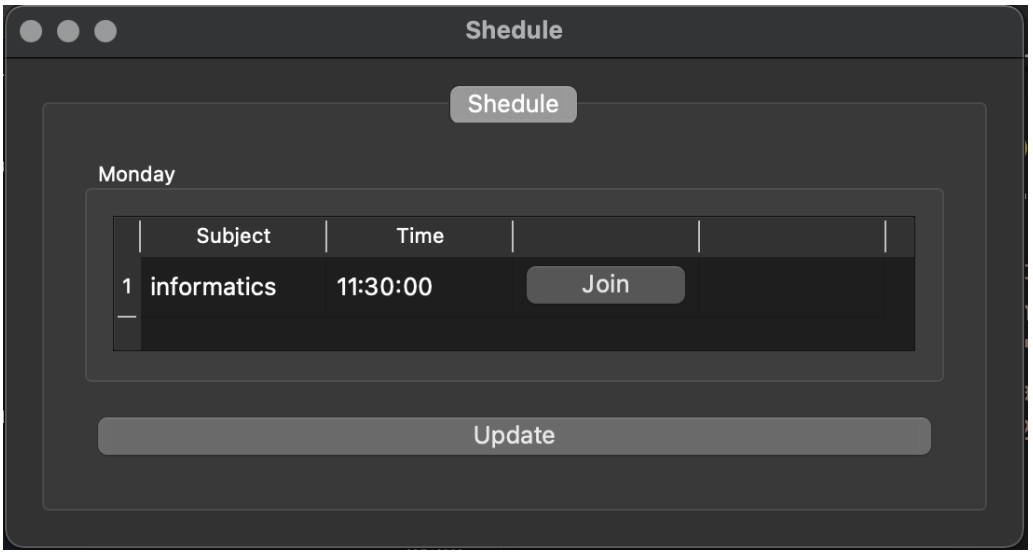
1. Использование библиотеки PyQt5.

2. Использование адаптера psycopg2.

3. Приложение должно иметь при себе функционал позволяющий: просматривать базу данных в удобном для пользователя формате, удалять, добавлять и изменять записи в этой же базе данных.

4. Визуальная часть должна иметь при себе:
    a. Минимум 3 вкладки, в каждой из которых содержится информация из отдельной таблицы в базе данных.
    b. Внутри каждой вкладки информация должна отображаться в виде таблиц.
    c. Внутри каждой вкладки должна отображаться кнопка с обновлением информации.
    d. Внутри каждой таблицы должны отображаться все поля из таблицы в базе данных в виде колонок
    e. Внутри каждой таблицы после каждой строки записи должны быть отображены кнопки изменения и удаления записи
    f. В конце каждой таблицы должна находиться пустая строка с кнопкой для добавления новой записи.
    g. На вкладке с расписанием дни недели должны быть указаны в отдельных таблицах.

Пример частичной реализации:



1. **Импортируем необходимые библиотеки и адаптеры**

```
import psycopg2
import sys

from PyQt5.QtWidgets import (QApplication, QWidget,
                             QTabWidget, QAbstractScrollArea,
                             QVBoxLayout, QHBoxLayout,
                             QTableWidget, QGroupBox,
                            QTableWidgetItem, QPushButton, QMessageBox)
```

2. **Создаем класс MainWindow с конструктором**

```
class MainWindow(QWidget):
  def __init__(self):
    super(MainWindow, self).__init__()

    self._connect_to_db()

    self.setWindowTitle("Shedule")

    self.vbox = QVBoxLayout(self)

    self.tabs = QTabWidget(self)
    self.vbox.addWidget(self.tabs)s

    self._create_shedule_tab()
```

Класс QTabWidget создает структуру, которую можно заполнять вкладками. Вкладки это подстраницы в окне приложения. Аналогом вкладок в оконных приложениях являются вкладки в веб-браузере.

3. **Создаем метод для подключения к базе данных**

```
def _connect_to_db(self):
    self.conn = psycopg2.connect(database="<название вашей базы данных>",
                        user="postgres",
                        password="1234",
                        host="localhost",
                        port="5432")

    self.cursor = self.conn.cursor()
```

4. **Создаем метод для отображения вкладки с расписанием**

```
def _create_shedule_tab(self):
    self.shedule_tab = QWidget()
    self.tabs.addTab(self.shedule_tab, "Shedule")

    self.monday_gbox = QGroupBox("Monday")

    self.svbox = QVBoxLayout()
    self.shbox1 = QHBoxLayout()
    self.shbox2 = QHBoxLayout()

    self.svbox.addLayout(self.shbox1)
    self.svbox.addLayout(self.shbox2)

    self.shbox1.addWidget(self.monday_gbox)

    self._create_monday_table()

    self.update_shedule_button = QPushButton("Update")
    self.shbox2.addWidget(self.update_shedule_button)
    self.update_shedule_button.clicked.connect(self._update_shedule)

    self.shedule_tab.setLayout(self.svbox)
```

Класс QWidget() создает виджет, который будет являться вкладкой в нашем приложении. Данный класс может также использоваться для создания отдельных окон, но в нашем случае будет вкладкой.

self.tabs.addTab(self.shedule_tab, "Shedule") добавляет в структуру с вкладками новую вкладку с названием "Shedule".

Класс QGroupBox() может группировать виджеты, он предоставляет рамку, заголовок вверху и может отображать несколько виджетов внутри. В нашем случае он служит исключительно в декоративных целях

5. **Создаем метод для отображения таблицы с расписанием на понедельник**

```
def _create_monday_table(self):
    self.monday_table = QTableWidget()
    self.monday_table.setSizeAdjustPolicy(QAbstractScrollArea.AdjustToContents)

    self.monday_table.setColumnCount(4)
    self.monday_table.setHorizontalHeaderLabels(["Subject", "Time", "", ""])

    self._update_monday_table()

    self.mvbox = QVBoxLayout()
    self.mvbox.addWidget(self.monday_table)
    self.monday_gbox.setLayout(self.mvbox)
```

Класс QTableWidget() создает пустую пользовательскую таблицу аналогичную таблицам Excel.

setSizeAdjustPolicy(QAbstractScrollArea.AdjustToContents)
устанавливает возможность изменения размера под размер данных в ячейке.

Метод setColumnCount() задает таблице количество колонок.

Метод setHorizontalHeaderLabels(["Название", "Название"])
задает колонкам подписи.

6. **Создаем метод для обновления таблицы с расписанием на понедельник**

```
def _update_monday_table(self):
    self.cursor.execute("SELECT * FROM timetable WHERE day='wednesday'")
    records = list(self.cursor.fetchall())

    self.monday_table.setRowCount(len(records) + 1)

    for i, r in enumerate(records):
      r = list(r)
      joinButton = QPushButton("Join")

      self.monday_table.setItem(i, 0,
                  QTableWidgetItem(str(r[0])))
      self.monday_table.setItem(i, 1,
                  QTableWidgetItem(str(r[2])))
      self.monday_table.setItem(i, 2,
                  QTableWidgetItem(str(r[4])))
      self.monday_table.setCellWidget(i, 3, joinButton)

      joinButton.clicked.connect(lambda ch, num=i:
self._change_day_from_table(num))

    self.monday_table.resizeRowsToContents()
```

Заполнение таблицы происходит в цикле for для того, чтобы динамически обрабатывать изменения в количестве записей.

Метод setRowCount() задает таблице количество строк.

Кнопка joinButton не является отдельным свойством класса MainWindow, так как нам не нужно ее "запоминать". Далее интерпретатор запоминает ее с помощью функции-обработчика clicked.connect().

Метод setItem(<Номер строки>, <Номер колонки>, <Строка с данными>) записывает в ячейку с определенным адресом строковые данные.

Метод setCellWidget(<Номер строки>, <Номер колонки>, <Виджет>) помещает в ячейку с определенным адресом виджет.
В нашем случае это кнопка "Join".

Метод resizeRowsToContents() автоматически адаптирует размеры ячеек таблицы под размер данных внутри этой ячейки. Это необходимо использовать для экономии визуального пространства.

7. **Создаем метод изменяющий запись в базе данных по нажатию на кнопку "Join"**

```
def _change_day_from_table(self, rowNum, day):
    row = list()
    for i in range(self.monday_table.columnCount()):
      try:
row.append(self.monday_table.item(rowNum, i).text())
      except:
row.append(None)

    try:
self.cursor.execute("UPDATE SQL запрос на изменение одной строки в базе данных", (row[0],))
self.conn.commit()
    except:
QMessageBox.about(self, "Error", "Enter all fields")
```

Метод columnCount() возвращает количество колонок таблицы.

Конструкция item(<Номер строки>, <Номер столбца>).text() возвращает текст, записанный в определенной ячейке.

8. **Создаем метод обновляющий все таблицы на вкладке**

```
def _update_shedule(self):
    self._update_monday_table()
                ...
        Ваши методы обновления таблиц
```

9. **"Запускаем" наше приложение**

```
app = QApplication(sys.argv)
win = MainWindow()
win.show()
sys.exit(app.exec_())
```

Система может быть дополнена Вашим функционалом, но должна соответствовать минимальным требованиям.