МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

**ОТЧЕТ**

по лабораторной работе № 7

на тему: «Разработка приложения, взаимодействующего с базой данных»

по дисциплине: «Программирование на языке Python»

Вариант № 18

Выполнил: Шорин В.Д. Шифр: 171406

Институт приборостроения, автоматизации и информационных технологий

Направление: 09.03.04 «Программная инженерия»

Группа: 71-ПГ

Проверили: Захарова О.В., Раков В.И.

Отметка о зачете:

Дата: «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2019 г.

Орел, 2019 г.

**Задание:**

Разработать приложение (можно web-приложение), взаимодействующее с базой данных. Приложение должно иметь удобный графический интерфейс. Базу данных разработать в соответствии с темой своего варианта (варианты представлены ниже). База данных должна состоять из 2-3 связанных таблиц; одна таблица основная.

Функционал приложения:

- добавление информации в основную таблица;

- удалении информации из основной таблицы;

- отображение информации из основной таблицы.

Добавление и отображение информации должно быть реализовано в читаемой для пользователя форме (внешние ключи не отображать, вместо них отображать пользователю понятную информацию).

18. База данных «Процессоры».

**Код:**

**«main.py»**

from PyQt5 import QtWidgets

from form import MainWindow

import sys

if \_\_name\_\_ == '\_\_main\_\_':

app = QtWidgets.QApplication(sys.argv)

main\_window = MainWindow()

main\_window.show()

sys.exit(app.exec())

**«processor.py»**

class Processor:

data = dict

def \_\_init\_\_(self, \*\*kwargs):

self.data = kwargs

def \_\_str\_\_(self):

return f"\nArticle: {self.data.get('article')}\n" \

f"Producer: {self.data.get('producer')}\n" \

f"Name: {self.data.get('name')}\n" \

f"Cores count: {self.data.get('cores')}\n" \

f"Frequency: {self.data.get('frequency')}\n"\

f"Price: {self.data.get('price')}\n"

@staticmethod

def attributes():

return ['article', 'producer', 'name',

'cores', 'frequency', 'price']

def \_\_dict\_\_(self):

return {

'article': self.data.get('article'),

'producer': self.data.get('producer'),

'name': self.data.get('name'),

'cores': self.data.get('cores'),

'frequency': self.data.get('frequency'),

'price': self.data.get('price'),

}

**«form.py»**

import json

import os

import pymysql

from pymysql.cursors import DictCursor

from typing import List

from PyQt5 import QtWidgets

from PyQt5.QtWidgets import QMessageBox

import lab7 as view

import processor as proc

global processors

processors = list()

global cores\_table

cores\_table = dict

default\_path = "C:/Users/vscho/PycharmProjects/Python\_Lab5"

user\_path\_to\_save = "C:/Users/vscho/PycharmProject/Python\_Lab5"

class MainWindow(QtWidgets.QMainWindow, view.Ui\_MainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setupUi(self)

self.init\_actions()

self.init\_table()

self.init\_values()

self.init\_menu()

self.init\_db()

self.load\_processors()

self.update\_table\_content()

def init\_actions(self):

# set actions on buttons

self.buttonAdd.clicked.connect(self.add\_processor)

self.buttonDelete.clicked.connect(self.delete\_processor)

self.buttonSearch.clicked.connect(self.search\_processor)

self.buttonPrintAll.clicked.connect(self.print\_all)

self.buttonPrintByCheckBox.clicked.connect(self.print\_by\_combo\_box)

def init\_db(self):

self.connection = pymysql.connect(

host='localhost',

user='root',

password='1234',

db='processors',

charset='utf8mb4',

cursorclass=DictCursor

)

self.cursor = self.connection.cursor()

def load\_processors(self):

query = """

SELECT \*

FROM cores

"""

self.cursor.execute(query)

global cores\_table

cores\_table = self.cursor.fetchall()

query = """

SELECT \*

FROM processor

"""

self.cursor.execute(query)

global processors

processors = list()

for row in self.cursor:

for elem in cores\_table:

if elem['idCores'] == row['cores\_idCores']:

cores = elem['value']

processor = proc.Processor(

article=row['article'],

producer=row['producer'],

name=row['name'],

cores=cores,

frequency=row['frequency'],

price=row['price']

)

processors.append(processor)

self.update\_table\_content()

def init\_values(self):

self.comboBoxCores.clear()

self.comboBoxCores.addItems(['None', '2', '4', '6'])

processors.append(proc.Processor(

article=1,

producer=1,

name=1,

cores=1,

frequency=1,

price=1

))

processors.append(proc.Processor(

article=2,

producer=2,

name=2,

cores=2,

frequency=2,

price=1

))

processors.append(proc.Processor(

article=3,

producer=3,

name=3,

cores=3,

frequency=3,

price=333

))

processors.append(proc.Processor(

article=4,

producer=4,

name=4,

cores=4,

frequency=4,

price=444

))

processors.append(proc.Processor(

article=5,

producer=5,

name=5,

cores=5,

frequency=5,

price=555

))

processors.append(proc.Processor(

article=6,

producer=6,

name=6,

cores=6,

frequency=6,

price=6

))

def add\_processor(self):

processor = proc.Processor(

article=self.editArticle.text(),

producer=self.editProducer.text(),

name=self.editName.text(),

cores=self.editCores.text(),

frequency=self.editFrequency.text(),

price=self.editPrice.text()

)

processors.append(processor)

cores = self.editCores.text()

self.cursor.execute(

f"select idCores from cores where idCores = {cores}"

)

cores = self.cursor.fetchall()

query = f"insert into processors.processor("\

f"article, producer, name, "\

f"cores\_idCores, frequency, price)"\

f"value ('{self.editArticle.text()}', '{self.editProducer.text()}',"\

f"{self.editName.text()}, {cores[0]['idCores']},"\

f"{self.editFrequency.text()}, {self.editPrice.text()})"

self.cursor.execute(query)

self.connection.commit()

self.clear\_add\_edits()

self.update\_table\_content()

def delete\_processor(self):

article = self.editDeleteArticle.text()

for processor in processors:

if processor.data.get('article') == article:

processors.remove(processor)

self.cursor.execute(f'delete from processors.processor where article = {article}')

self.connection.commit()

self.editDeleteArticle.clear()

self.update\_table\_content()

def search\_processor(self):

producer = self.editSearch.text()

data = list()

for processor in processors:

if str(processor.data.get('producer')) == producer:

data.append(processor)

self.editSearch.clear()

if len(data) == 0:

msg\_box\_no = QtWidgets.QMessageBox.question(self, 'Empty', "There are no such processors", QMessageBox.Ok)

if msg\_box\_no == QMessageBox.Ok:

pass

self.update\_table\_content()

pass

else:

self.update\_table\_content(data)

def print\_all(self):

self.update\_table\_content()

def init\_table(self):

self.tableResult.setColumnCount(6)

self.tableResult.setRowCount(1)

for i, attr in enumerate(proc.Processor.attributes()):

self.tableResult.setItem(

0, i, QtWidgets.QTableWidgetItem(attr)

)

def update\_table\_content(self, data: List[proc.Processor] = None):

if data is None:

data = processors.copy()

global msg\_box

if len(data) == 0:

msg\_box = QtWidgets.QMessageBox.question(self, 'Empty', "There are no processors", QMessageBox.Ok)

if msg\_box == QMessageBox.Ok:

pass

self.tableResult.clear()

rows = len(data)

self.tableResult.setRowCount(rows + 1)

for i, attr in enumerate(proc.Processor.attributes()):

self.tableResult.setItem(

0, i, QtWidgets.QTableWidgetItem(attr)

)

if self.checkBoxPrice.isChecked():

check\_list = list()

for row, processor in enumerate(data):

if float(data[row].data['price'] > 100):

check\_list.append(processor)

if len(check\_list) == 0:

msg\_box = QtWidgets.QMessageBox.question(self, 'Empty', "There are no processors", QMessageBox.Ok)

if msg\_box == QMessageBox.Ok:

pass

else:

self.tableResult.clear()

rows = len(check\_list)

self.tableResult.setRowCount(rows + 1)

for i, attr in enumerate(proc.Processor.attributes()):

self.tableResult.setItem(

0, i, QtWidgets.QTableWidgetItem(attr)

)

for row, processor in enumerate(check\_list):

for column, attribute in enumerate(proc.Processor.attributes()):

self.tableResult.setItem(

row + 1,

column,

QtWidgets.QTableWidgetItem(

str(check\_list[row].data[attribute])

)

)

else:

for row, processor in enumerate(data):

for column, attribute in enumerate(proc.Processor.attributes()):

self.tableResult.setItem(

row + 1,

column,

QtWidgets.QTableWidgetItem(

str(data[row].data[attribute])

)

)

def print\_by\_combo\_box(self):

index = self.comboBoxCores.currentIndex()

value = self.comboBoxCores.currentText()

if index == 0:

self.update\_table\_content()

elif index == 1:

self.print\_by\_cb\_index(value)

elif index == 2:

self.print\_by\_cb\_index(value)

elif index == 3:

self.print\_by\_cb\_index(value)

def print\_by\_cb\_index(self, value):

self.tableResult.clear()

cores\_list = list()

for row, processor in enumerate(processors):

if str(processors[row].data['cores']) == value:

if self.checkBoxPrice.isChecked():

if float(processors[row].data['price']) > 100:

cores\_list.append(processor)

else:

cores\_list.append(processor)

if len(cores\_list) == 0:

msg\_box\_cores = QtWidgets.QMessageBox.question(self, 'Empty', "There are no processors", QMessageBox.Ok)

if msg\_box\_cores == QMessageBox.Ok:

pass

rows = len(cores\_list)

self.tableResult.setRowCount(rows + 1)

for i, attr in enumerate(proc.Processor.attributes()):

self.tableResult.setItem(

0, i, QtWidgets.QTableWidgetItem(attr)

)

for row, processor in enumerate(cores\_list):

for column, attribute in enumerate(proc.Processor.attributes()):

self.tableResult.setItem(

row + 1,

column,

QtWidgets.QTableWidgetItem(

str(cores\_list[row].data[attribute])

)

)

def clear\_add\_edits(self):

self.editArticle.clear()

self.editProducer.clear()

self.editName.clear()

self.editCores.clear()

self.editFrequency.clear()

self.editPrice.clear()

def init\_menu(self):

menu = self.menuBar()

menu.setNativeMenuBar(False)

self.actionCreate.triggered.connect(self.on\_create)

self.actionOpen.triggered.connect(self.on\_open)

self.actionSave.triggered.connect(self.on\_save)

self.actionSave\_as.triggered.connect(self.on\_save\_as)

self.actionExit.triggered.connect(QtWidgets.qApp.exit)

self.actionAbout.triggered.connect(self.on\_about)

def on\_create(self):

self.clear\_add\_edits()

self.editDeleteArticle.clear()

self.editSearch.clear()

processors.clear()

self.update\_table\_content()

def on\_open(self):

path = QtWidgets.QFileDialog.getOpenFileName(

self,

'Open file',

default\_path,

"\*.json"

)[0]

if os.path.isfile(path):

with open(path, 'r') as f:

data = json.load(f)

global processors

processors = list()

for elem in data:

processor = proc.Processor(

article=elem['article'],

producer=elem['producer'],

name=elem['name'],

cores=elem['cores'],

frequency=elem['frequency'],

price=elem['price']

)

processors.append(processor)

self.update\_table\_content()

def on\_save(self):

if os.path.exists(f'{user\_path\_to\_save}/db.json'):

data = []

for p in processors:

data.append(p.\_\_dict\_\_())

with open(f'{user\_path\_to\_save}/db.json', "w", encoding='utf-8') as fp:

json.dump(data, fp)

else:

self.on\_save\_as()

def on\_save\_as(self):

path = QtWidgets.QFileDialog.getSaveFileName(

self,

'Save file',

default\_path,

"\*.json"

)[0]

if path != '':

global user\_path\_to\_save

user\_path\_to\_save = path

data = []

for p in processors:

data.append(p.\_\_dict\_\_())

with open(user\_path\_to\_save, "w", encoding='utf-8') as fp:

json.dump(data, fp)

def on\_about(self):

about = QtWidgets.QMessageBox.question(self, 'Info', "Vladislav Shorin", QMessageBox.Ok)

if about == QMessageBox.Ok:

pass