

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра інженерії програмного забезпечення**

**Лабораторна робота №3**

з дисципліни

**«**Бази даних і засоби управління**»**

Виконав:

студент ІII курсу

ФПМ групи КП-71

Гришко Ю. І.

Перевірив: Петрашенко А. В.

Київ – 2019

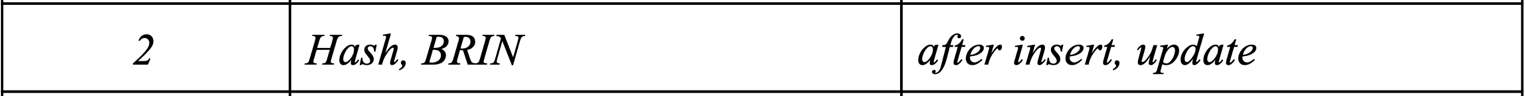
**Ознайомлення з базовими операціями СУБД PostgreSQL**

*Завдання* роботи полягає у наступному:

1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій уPostgreSQL.

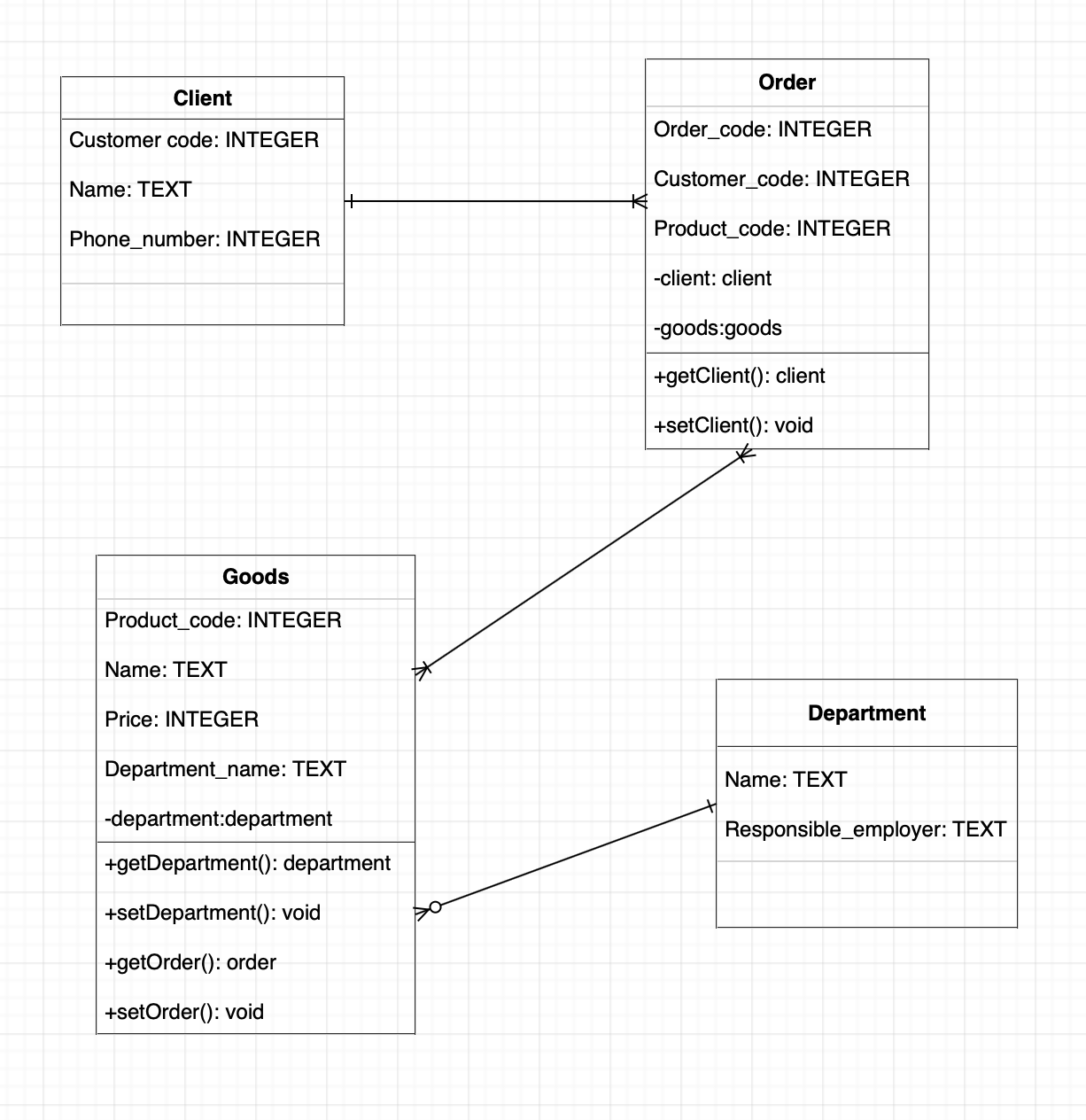
В ході роботи розроблено:

1. Логічну модель БД та Діаграму класів;
2. Функціонал програмного додатку;
3. Об'єктно-орієнтований програмний додаток роботи з БД "Магазин". Для взаємодії з БД використано ORM модуль SQLAlchemy.

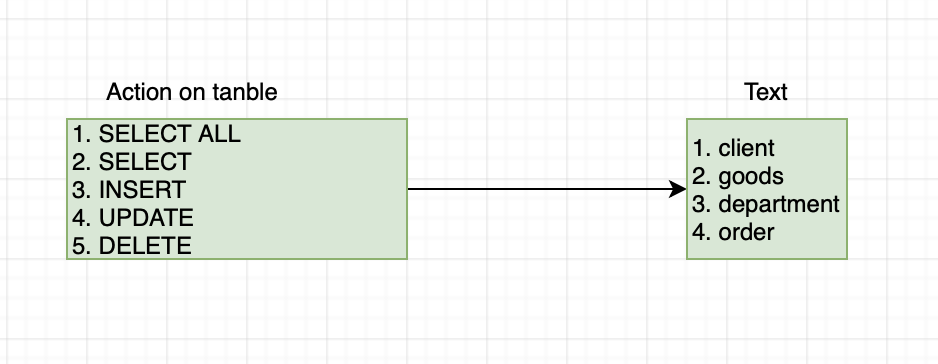
**Варіант 2**

Логічна модель бази даних

****

**Зв’язки між сутнісними класами, сгенеровані за допомогою SqlAlchemy**

**Опис меню програми**

****

**Посилання на основні класи та методи програми**

1. [Launcher](#Launcher)
   1. [Main метод](#main)
2. [DatabaseHelper](#Database_helper)
3. [DbInterface](#Dbinterface)
   1. [Вибрати всі елементи](#select_all)
   2. [Вибрати елементи](#select)
   3. [Оновити дані](#update)
   4. [Записати дані](#insert)
   5. [Видалити дані](#delete)
4. [Model](#Model)
5. [Models(ORM класи)](#Models)
   1. [goods](#promoter)
   2. employee
   3. order
   4. department\_of\_goods
6. [Controller](#Controller)
   1. [Запуск меню програми](#c_start)
   2. [Основне меню програми](#c_show_table_menu)
7. [View](#View)
8. [Utils](#Utils)

**launch****er.py**

from database\_helper import DatabaseHelper  
from controller import Controller  
from view import View  
from model import Model  
import class\_generator  
  
def main():  
 if (is\_need\_generate()):  
 class\_generator.generate\_classes()  
  
 controller = Controller(View(), Model())  
 controller.start()  
 DatabaseHelper.close()  
  
def is\_need\_generate():  
 try:  
 print("Don't need to generate")  
 f = open("models.py")  
 return False  
 except IOError:  
 print("Trying to generate tables objects")  
 return True  
 #finally:  
 # f.close()  
  
if (\_\_name\_\_ == '\_\_main\_\_'):  
 main()

**databas****e\_helper.py**

from sqlalchemy import inspect  
from sqlalchemy import Boolean, BOOLEAN  
from sqlalchemy.exc import NoInspectionAvailable  
import utils  
  
class DatabaseHelper(object):  
 # params  
 \_\_session = None  
 \_\_engine = None  
  
 #Using singleton pattern  
 def \_\_new\_\_(cls, \*args, \*\*kwargs):  
 if not hasattr(cls, '\_\_instance'):  
 cls.\_\_instance = super(DatabaseHelper, cls).\_\_new\_\_(cls)  
 return cls.\_\_instance  
  
 #Using singleton pattern  
 @classmethod  
 def get\_engine(cls):  
 if cls.\_\_engine is None:  
 cls.\_\_engine = engine = db.create\_engine('postgresql://' + credentials.user + ':' + credentials.password + "@" +  
 credentials.host + ":" + credentials.port + "/" + credentials.db\_name)  
 print("Created database engine ", cls.\_\_engine)  
 return cls.\_\_engine  
  
 #Using singleton pattern  
 @classmethod  
 def get\_session(cls):  
 if cls.\_\_session is None:  
 cls.\_\_session = Session(cls.get\_engine())  
 print("Database session opened")  
 return cls.\_\_session  
  
 @classmethod  
 def close(cls):  
 cls.get\_session().close()  
 print("Database session closed")  
  
 # Повертає інспектора для виконання запитів (engine - об'єкт для взаємодії з БД)  
 @classmethod  
 def \_\_getInspector(cls):  
 return inspect(cls.get\_engine())  
  
 @classmethod  
 def rollback\_session(cls):  
 cls.get\_session().rollback()  
  
 # Повертає список таблиць у БД (engine - об'єкт для взаємодії з БД)  
 @classmethod  
 def getListOfTables(cls):  
 inspector = cls.\_\_getInspector()  
  
 table\_list = []  
  
 for table\_name in inspector.get\_table\_names():  
 if table\_name is None: continue  
 table\_list.append(table\_name)  
  
 return table\_list  
  
 @classmethod  
 def getTableColumns(cls, table\_name):  
 inspector = cls.\_\_getInspector()  
 return inspector.get\_columns(table\_name)  
  
 @classmethod  
 def get\_primary\_keys(cls, table\_name):  
 inspector = cls.\_\_getInspector()  
 return inspector.get\_primary\_keys(table\_name)  
  
 @classmethod  
 def get\_foreign\_keys(cls, table\_name):  
 inspector = cls.\_\_getInspector()  
 return inspector.get\_foreign\_keys(table\_name)  
  
 @classmethod  
 def fill\_db\_object(cls, model, inflator\_object):  
 if not isinstance(inflator\_object, object):  
 print("Aborting! This is object?! You are WRONG!")  
 return  
  
 db\_object = model.create()  
 for key in inflator\_object.keys():  
 try:  
 if isinstance(key.type, Boolean) or inspect(key.type, BOOLEAN):  
 setattr(db\_object, key.name, utils.get\_boolean\_from(inflator\_object[key]))  
 else:  
 setattr(db\_object, key.name, inflator\_object[key])  
 except NoInspectionAvailable:  
 setattr(db\_object, key.name, inflator\_object[key])  
 return db\_object  
  
 @classmethod  
 def getTableObject(cls, table\_name):   
 if table\_name == 'goods':   
 return goods   
 if table\_name == 'department':  
 return department  
 if table\_name == 'order':   
 return order   
 if table\_name == 'client':  
 return client

**class\_generatory.py**

from database\_helper import DatabaseHelper  
from sqlalchemy import VARCHAR  
  
  
def get\_all\_tables():  
 DatabaseHelper.getTableColumns("Book")  
  
  
def get\_constraint(table\_name, column\_name):  
 primary\_keys = DatabaseHelper.get\_primary\_keys(table\_name)  
 foreign\_keys = DatabaseHelper.get\_foreign\_keys(table\_name)  
  
 if primary\_keys.\_\_contains\_\_(column\_name):  
 return ", primary\_key=True"  
  
 for foreign\_key in foreign\_keys:  
 for referred\_column in foreign\_key['referred\_columns']:  
 if referred\_column == column\_name:  
 return ", ForeignKey('" + str(foreign\_key['referred\_table']) + "." + str(referred\_column) + "')"  
  
 return ""  
  
def generate\_file\_header():  
 class\_text = ""  
  
 class\_text += "from sqlalchemy import \*" \  
 "\nfrom sqlalchemy.ext.declarative import declarative\_base" \  
 "\nfrom sqlalchemy.orm import relationship" \  
 "" \  
 "\n\nBase = declarative\_base()\n\n"  
  
 return class\_text  
  
def generate\_classes():  
 tables = DatabaseHelper.getListOfTables()  
  
 database\_helper\_content = ""  
  
 with open("database\_helper.yandroid", 'r') as fileDb:  
 database\_helper\_content = fileDb.read()  
 fileDb.close()  
  
 with open("database\_helper.py", 'w') as fileDb:  
 fileDb.write("from models import \*\n")  
 fileDb.writelines(database\_helper\_content)  
 fileDb.writelines(generate\_table\_object\_creation())  
  
 with open('models.py', 'w') as file:  
 file.write("#Generated by Horba Dima\n")  
 file.writelines(generate\_file\_header())  
 for table in tables:  
 file.writelines(generate\_database\_class(table))  
  
def generate\_table\_object\_creation():  
 tables = DatabaseHelper.getListOfTables()  
  
 class\_text = "\n @classmethod" \  
 "\n def getTableObject(cls, table\_name):"  
  
 for table in tables:  
 class\_text += " \n if table\_name == '" + table + "':" \  
 " \n return " + table  
  
 print(class\_text)  
 return class\_text  
  
def get\_str\_column\_type(type):  
 if isinstance(type, VARCHAR):  
 return "String"  
 else:  
 return str(type)  
  
def generate\_database\_class(table\_name):  
 columns = DatabaseHelper.getTableColumns(table\_name)  
  
 class\_text = ""  
  
 class\_text += "class " + table\_name + "(Base):\n"  
 class\_text += "\t\_\_tablename\_\_ = '" + table\_name + "'\n"  
  
 for column in columns:  
 class\_text += "\n\t" + column['name'] + " = Column('" + column['name'] + "', " + get\_str\_column\_type(column['type']) + get\_constraint(table\_name, column['name']) + ")"  
  
 class\_text += "\n\n\tdef \_\_str\_\_(self):" \  
 "\n\t\treturn '"  
  
 for column in columns:  
 class\_text += column['name'] + "={" + column['name'] + "}, "  
  
 class\_text = class\_text[:-1]  
 class\_text += "'.format("  
  
 for column in columns:  
 class\_text += column['name'] + "=self." + column['name'] + ","  
  
 class\_text = class\_text[:-1]  
 class\_text += ")"  
  
 class\_text += "\n\n\tdef \_\_repr\_\_(self):" \  
 "\n\t\treturn str(self)"  
  
 class\_text += "\n\n\t@classmethod" \  
 "\n\tdef get\_columns(cls):" \  
 "\n\t\treturn ["  
  
 for column in columns:  
 class\_text += table\_name + "." + column['name'] + ","  
  
 class\_text = class\_text[:-1]  
 class\_text += "]\n"  
  
 class\_text += "\n\t@classmethod" \  
 "\n\tdef create(cls):" \  
 "\n\t\treturn " + table\_name + "()\n\n"  
  
 print(class\_text)  
 return class\_text

**model****.py**

from base\_dao import BaseDao  
from database\_helper import DatabaseHelper  
import utils  
  
  
class Model(object):  
  
 def getTablesNames(self):  
 return DatabaseHelper.getListOfTables()  
  
 def getTableObject(self, table\_name):  
 return DatabaseHelper.getTableObject(table\_name)  
  
 def select\_all(self, table\_name):  
 model = self.getTableObject(table\_name)  
 return BaseDao(model).select\_all()  
  
 def select(self, table, object\_to\_find):  
 model = self.getTableObject(table)  
 baseDao = BaseDao(model)  
  
 mapped = utils.map\_keys\_and\_values\_from\_obect(object\_to\_find)  
  
 if mapped is None:  
 print("Aborting! mapped is None")  
 return None  
  
 return baseDao.select(mapped['columns'], mapped['values'])  
  
 def fill\_object(self, table\_name, inflator\_object):  
 model = self.getTableObject(table\_name)  
 return DatabaseHelper.fill\_db\_object(model, inflator\_object)  
  
 def insert(self, object\_to\_insert):  
 baseDao = BaseDao(None)  
 baseDao.insert(object\_to\_insert)  
 print("Inserted \t\t\t\tor Not ;)")  
  
 def update(self, table, object\_to\_update, updated\_object):  
 model = self.getTableObject(table)  
 baseDao = BaseDao(model)  
  
 mappedFind = utils.map\_keys\_and\_values\_from\_obect(object\_to\_update)  
 mappedUpdate = utils.map\_keys\_and\_values\_from\_obect(updated\_object)  
  
 baseDao.update(mappedFind['columns'], mappedFind['values'], mappedUpdate['columns'], mappedUpdate['values'])  
 print("Updated \t\t\t\tor Not ;)")  
  
 def delete(self, table, object\_to\_delete):  
 model = self.getTableObject(table)  
 baseDao = BaseDao(model)  
  
 mapped = utils.map\_keys\_and\_values\_from\_obect(object\_to\_delete)  
  
 baseDao.delete(mapped['columns'], mapped['values'])  
 print("Deleted \t\t\t\tor Not ;)")  
  
 def rollback\_session(self):  
 DatabaseHelper.rollback\_session()

**model****s.py**

from sqlalchemy import \*  
from sqlalchemy.ext.declarative import declarative\_base  
from sqlalchemy.orm import relationship  
  
Base = declarative\_base()  
  
class goods(Base):  
 \_\_tablename\_\_ = 'goods'  
  
 goods\_code = Column('goods\_code', INTEGER, primary\_key=True)  
 goods\_name = Column('goods\_name', String)  
 goods\_price = Column('goods\_price', INTEGER)  
 goods\_id = Column('goods\_id', INTEGER)  
 department\_name = Column('department\_name', String, ForeignKey('department.department\_name'))  
  
 def \_\_str\_\_(self):  
 return 'goods\_code={goods\_code}, goods\_name={goods\_name}, goods\_price={goods\_price}, goods\_id={goods\_id},' \  
 ' department\_name={department\_name},'.format(goods\_code=self.goods\_code,goods\_name=self.goods\_name,goods\_price=self.goods\_price,goods\_id=self.goods\_id,department\_name=self.department\_name)  
  
 def \_\_repr\_\_(self):  
 return str(self)  
  
 @classmethod  
 def get\_columns(cls):  
 return [goods.goods\_code,goods.goods\_name,goods.goods\_price,goods.goods\_id,goods.department\_name]  
  
 @classmethod  
 def create(cls):  
 return goods()  
  
class department(Base):  
 \_\_tablename\_\_ = 'department'  
  
 department\_name = Column('department\_name', String, primary\_key=True)  
 department\_id = Column('department\_id', INTEGER)  
 responsible\_employer = Column('responsible\_employer', String)  
  
 def \_\_str\_\_(self):  
 return 'department\_name={department\_name}, department\_id={department\_id}, responsible\_employer={responsible\_employer},'.format(department\_name=self.department\_name,department\_id=self.department\_id,responsible\_employer=self.responsible\_employer)  
  
 def \_\_repr\_\_(self):  
 return str(self)  
  
 @classmethod  
 def get\_columns(cls):  
 return [department.department\_name,department.department\_id,department.responsible\_employer]  
  
 @classmethod  
 def create(cls):  
 return department()  
  
class order(Base):  
 \_\_tablename\_\_ = 'order'  
  
 order\_code = Column('order\_code', INTEGER, primary\_key=True)  
 order\_data = Column('order\_data', DATE)  
 customer\_code = Column('customer\_code', INTEGER, ForeignKey('client.customer\_code'))  
 product\_code = Column('product\_code', INTEGER, ForeignKey('goods.goods\_code'))  
 order\_id = Column('order\_id', INTEGER)  
  
 def \_\_str\_\_(self):  
 return 'order\_code={order\_code}, order\_data={order\_data}, customer\_code={customer\_code}, product\_code={product\_code},order\_id={order\_id},'.format(order\_code=self.order\_code,order\_data=self.order\_data,customer\_code=self.customer\_code,product\_code=self.product\_code,order\_id=self.order\_id)  
  
 def \_\_repr\_\_(self):  
 return str(self)  
  
 @classmethod  
 def get\_columns(cls):  
 return [order.order\_code,order.order\_data,order.customer\_code,order.product\_code,order.order\_id]  
  
 @classmethod  
 def create(cls):  
 return order()  
  
class client(Base):  
 \_\_tablename\_\_ = 'client'  
  
 customer\_code = Column('customer\_code', INTEGER, primary\_key=True)  
 name = Column('name', String)  
 phone\_number = Column('phone\_number', INTEGER)  
 client\_id = Column('client\_id', INTEGER)  
  
 def \_\_str\_\_(self):  
 return 'customer\_code={customer\_code}, name={name}, phone\_number={phone\_number}, client\_id={client\_id},'.format(customer\_code=self.customer\_code,name=self.name,phone\_number=self.phone\_number,client\_id=self.client\_id)  
  
 def \_\_repr\_\_(self):  
 return str(self)  
  
 @classmethod  
 def get\_columns(cls):  
 return [client.customer\_code,client.name,client.phone\_number,client.client\_id]  
  
 @classmethod  
 def create(cls):  
 return client()

**con****troller.py**

class Controller(object):  
  
 def \_\_init\_\_(self, view, model):  
 self.view = view  
 self.model = model  
  
 def start(self):  
 tables\_names = self.model.getTablesNames()  
 length = int(len(tables\_names))  
 self.view.print\_main\_menu(tables\_names)  
 input\_number = self.view.request\_input()  
 try:  
 num = int(input\_number) - 1  
 if num > length + 1 or num < 0:  
 print("Wrong choice, make another...")  
 print()  
 self.start()  
 else:  
 if num == length:  
 return  
 self.show\_table\_menu(tables\_names[num])  
 except TypeError as error:  
 print("Wrong choice, make another...")  
 print()  
 self.start()  
  
  
 def show\_table\_menu(self, table\_name):  
 self.view.print\_table\_menu(table\_name)  
 table\_object = self.model.getTableObject(table\_name)  
 input\_number = self.view.request\_input()  
 try:  
 num = int(input\_number)  
  
 if num == 1:  
 result = self.model.select\_all(table\_name)  
 print(result)  
 print()  
 self.show\_table\_menu(table\_name)  
 return  
  
 if num == 2:  
 filled\_object = self.view.request\_input\_object(table\_object.get\_columns())  
 result = self.model.select(table\_name, filled\_object)  
 print(result)  
 print()  
 self.show\_table\_menu(table\_name)  
 return  
  
 if num == 3:  
 filled\_object = self.view.request\_input\_object(table\_object.get\_columns())  
 db\_object = self.model.fill\_object(table\_name, filled\_object)  
 self.model.insert(db\_object)  
 print()  
 self.show\_table\_menu(table\_name)  
 return  
  
 if num == 4:  
 filled\_object = self.view.request\_input\_object(table\_object.get\_columns())  
 updated\_object = self.view.request\_input\_object(table\_object.get\_columns())  
 self.model.update(table\_name, filled\_object, updated\_object)  
 print()  
 self.show\_table\_menu(table\_name)  
 return  
  
 if num == 5:  
 filled\_object = self.view.request\_input\_object(table\_object.get\_columns())  
 self.model.delete(table\_name, filled\_object)  
 print()  
 self.show\_table\_menu(table\_name)  
 return  
  
 if num == 6:  
 print()  
 self.start()  
 return  
  
 except TypeError as error:  
 self.model.rollback\_session()  
 print("Wrong choice, make another...", error)  
 print()  
 self.show\_table\_menu(table\_name)

**view****.py**

import pandas as pd  
  
  
class View(object):  
  
 def request\_input(self):  
 print("It's time to make your choice: ")  
 return input()  
  
 def request\_input\_object(self, columns):  
 inputed\_object = {}  
 print("Let's fill some objects: ")  
 for column in columns:  
 print(column.name + " = ")  
 value = input()  
 if str(value) == "": continue  
 inputed\_object[column] = str(value)  
 return inputed\_object  
  
 def print\_main\_menu(self, tables\_names):  
 index = 1  
 for table\_name in tables\_names:  
 print(str(index) + ". " + str(table\_name))  
 index += 1  
  
 print()  
 print(str(index) + ". Exit")  
  
 def print\_table\_menu(self, table\_name):  
 print("-------------------------------- " + str(table\_name) + " --------------------------------")  
 print("1. SELECT ALL")  
 print("2. SELECT")  
 print("3. INSERT")  
 print("4. UPDATE")  
 print("5. DELETE")  
 print()  
 print("6. Back")  
  
  
 def show\_fetched\_result(self, db\_object, result):  
 df = pd.DataFrame(result)  
 df.columns = db\_object.columns  
 print(df)

**utils.p****y**

def map\_keys\_and\_values\_from\_obect(obj):  
 if not isinstance(obj, object):  
 print("Aborting! U pass not objects, WTF?!")  
 return None  
  
 columns = []  
 values = []  
  
 for key in obj.keys():  
 columns.append(key)  
 values.append(obj[key])  
  
 return {"columns": columns, "values": values}  
  
def get\_boolean\_from(stuff):  
 try:  
 maybe\_nmber = int(str(stuff))  
 return maybe\_nmber == 1  
 except TypeError:  
 # definitely not number ;)  
 try:  
 maybe\_string = str(stuff)  
 return maybe\_nmber.lower() == "true"  
 except TypeError:  
 #FUCK IT  
 return False

**Досліди з індексацією**

**Створення тригера**