МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ,
СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА» (СПбГУТ)
Кафедра информационных управляющих систем

Б1.В.24 «Программирование критических сервисов»
для специальностей по направлению
09.03.02 «Информационные системы и технологии»

Отчет по лабораторным работам № 1, 2, 3

Студент гр. ИБ-11з _____ Фамилия И.О.
(подпись)
Проверил _____ Параничев А.В.
(оценка и подпись)

Санкт-Петербург
2022 год

Лабораторная работа № 1: Создание базы данных MySQL с персональной информацией и настройка окружения на языке Python

[Желтым выделен текст, который нужно поменять на свой; зеленым выделен текст, который следует прочитать, выполнить действия по соответствующим рекомендациям, а перед отправкой отчета удалить и сохранить изменения!)

Отчет следует оформить, изменяя данный шаблон, переименовав его, исходя из своего номера группы, а также фамилии и имени.

Исходные данные:

Следует добавить 2 поля с персональной информацией в скрипт таблицы (пример представлен ниже), в зависимости от последней цифры зачетки:

0 - вид документа и номер документа;

1 - номер паспорта и место жительства;

2 - номер телефона и пароль;

3 - номер кредитной карточки и секретный код

4 - комер кредитной карточки и годность: месяц/год

5 - вид кредитной карты и номер карты

6 - почтовый логин и пароль

7 - вид логина и логин

8 - имя держателя карты и секретный код

9 - имя держателя карты и годность: месяц/год

Сервер-логин-пароль задаются на занятии, иначе/либо выбираются самостоятельно (в этом случае следует привести свои значения, объяснить подключение в отчете: СУБД MySQL).

Перейти в phpMyAdmin (http://www.sutct.org/relRatEnDedEnMELingLEAK/index.php), войти с предоставленными логином и паролем, добавить скрипт для последующего сохранения идентификаторов стикеров и соответствующей персональной информации].

Скрипт для создания базы данных по логину 9_02 и паролю abcdsdfsdf на удаленном сервере http://www.sutct.org/relRatEnDedEnMELingLEAK/index.php, включающих персональную информацию (номер карты и пин-код) и соответствующие стикеры:

```
USE 9_02;
DROP TABLE IF EXISTS t_stickers;
CREATE TABLE t_stickers(
  t_id int(5) AUTO_INCREMENT UNIQUE,
  sticker_unique_id varchar(100),
  sticker_id varchar(100),
  sticker_name varchar(50),
  PRIMARY KEY (sticker_unique_id)
```

);

Объяснение полей о персональной информации: 3-4 строки!

[В месседжере Telegram создать бот (поиск по фразе "@BotFather"), получив на него *токен*, сам бот запустить: команда "/start" в мессенджере. https://tlgrm.ru/docs/bots

составить недлинное имя, ассоциированное со своим заданием; запомнить токен вида botNNN:ABC и нигде его не публиковать, т.к. токен позволяет выполнять любые настрой-ки бота], первоначальная информация о боте в формате JSON (по адресу https://api.telegram.org/botNNN:ABC/getme, где botNNN:ABC - секретный токен бота).]

В мессенджере Telegram создан бот … , информацию о котором можно получить по REST API запросу https://api.telegram.org/botNNN:ABC/getme (где botNNN:ABC - секретный токен бота):

{JSON-ответ}

Разработка Telegram-бота выполняется с помощью PyCharm на языке Python, после создания проекта устанавливается библиотека "чистого" API Telegram: pyTelegramBot, для обращения к функциям СУБД MySQL подключена библиотека pymsql.

Установить/настроить PyCharm, создать новый проект (File->New Project), установить библиотеки в консоли (окно Terminal) для работы с мессенджером Telegram и СУБД MySQL:

pip install pyhton-telegram-bot

pip install pymysql

Рабочие примеры, с помощью которых можно дополнить/изменить проект: https://docs-python.ru/packages/biblioteka-python-telegram-bot-python/

Лабораторная работа № 2: Обработка персональных данных из базы данных MySQL в мессенджере Telegram на языке Python

Для программной реализации Telegram-бота следует добавить файлы в проект:

[название и состав файлов менять необязательно; кодом можно воспользоваться, либо написать свой!)]

- config.py – для сохранения данных ***токена*** и логин/пароля от БД;
- db_mysql.py – для управления базой данных MySQL;
- tg_mysql_bot.py – для запуска работы бота по управлению стикерами.

Содержимое тестовых файлов приведено ниже:

config.py

```
API_TOKEN = '1111111:AAAAAAAAAAAAAAA'
PHP_PASSWORD = 'adfasdfasdfasd'
PHP_DB_LOGIN = '9_02'
```

db_mysql.py

```
import time
import json
import pymysql
from decimal import *
import config
# для распознавания типа Decimal из формата JSON
def decimal_default(obj):
    if isinstance(obj, Decimal):
        return float(obj)
    raise TypeError("Ошибка сериализации JSON: '%s'" % type(obj).__name__)

class dbase_mysql:
    def __init__(self):
        self._db = self._mysql_connect()
        if self._db is not None:
            print("Проверка чтения")
            str_test = self.select_data(2) # проверка доступа по индексу 2
            print("str_test = {}".format(str_test))
            str_test = self.select_data('AllStickerNames') # проверка вывода всех
стикеров
            print("str_test = {}".format(str_test))

    def _mysql_connect(self):
        print("_mysql_connect()")
        _db = None
        try:
            _db = pymysql.connect(
                    host='sutct.org',
                    user=config.PHP_DB_LOGIN,
                    password=config.PHP_PASSWORD,
                    database=config.PHP_DB_LOGIN)
            print("Подключение к БД успешно установлено!) (время:
{})".format(time.localtime()))
        except Exception as e:
            print("Подключение к БД не установлено!( \n{}\n{}\n(время: {})".
                  e.__repr__(), e.args, format(time.localtime()))
        return _db

    # для обработки запросов SELECT: всех переменных столбца (flag_int = 0), одной
```

```
переменной (flag_int = 1),
    #                   либо insert/delete/update (flag_int = 2)
    def _run_sql(self, sql_string, flag_int):
        print("_run_sql({},{})".format(sql_string, flag_int))
        db = self._mysql_connect()
        print("Подключение к базе данных выполнено!")
        ret = None
        try:
            if db is not None:
                cur = db.cursor()
                print("Запрос {} успешно подготовлен!)".format(sql_string))
            else:
                print("Не удалось подключиться к БД!(")
                return ret
        except:
            print("Запрос {} не удалось подготовить!(".format(sql_string))
            return ret
        try:
            cur.execute(sql_string)
            print("Выполнено: cur.execute(sql)")
            print("flag_int = {}".format(flag_int)) # 0 и 1 - SELECT, 2 - INSERT
            if flag_int == 0 or flag_int == 1: # SELECT: ячейка (1) (add_param -
имя извлекаемой переменной) или все (0)
                rows = cur.fetchall()
                if rows == ():
                    print("В извлеченных строках нет данных!(")
                    return ret
                print("fetched rows = {}".format(rows))
                count = 0 # счетчик получаемых значений
                for row in rows:
                    if flag_int == 1: # при извлечении единственного значения (из
результата запроса)
                        ret = row[0]
                        break
                    else:  # flag_int == 2: извлекаем все значения по порядку (из
результата запроса)
                        if count == 0:
                            ret = "{}".format(row[0])
                        else:
                            ret = ret + "\n{}".format(row[0])
                        count = count + 1
                print("ret = {}".format(ret))
            elif flag_int == 2: # INSERT/DELETE/UPDATE
                db.commit()
            # общий блок для INSERT/SELECT
            print("Успешно получен результат на запрос:\n{}".format(ret))
        except Exception as e:
            print("Запрос {} не удалось выполнить на сервере!( Ошибка:\n{}\n{}".
                  format(sql_string, e.__repr__(), e.args))
        finally:
            db.close()
        return ret

    # получение списка всех строк с нумерацией
    def _get_list_data(self, data_column):
        print("_get_list_data({})".format(data_column))
        data = data_column.split('\n')
        print("data = {}\n".format(data))
        count = 0
        for row in data:
            if count == 0:
                ret_list = "{} - {}".format(count, row) # нулевая строка
            else:
                ret_list = ret_list + "\n{} - {}".format(count, row) # последующие
```

```
строки
            count = count + 1
        print("ret_list = {}\n".format(ret_list))
        return ret_list

    # получение уникального id по имени из БД; для имени 'AllStickerNames'
выводятся все имена в списке
    def _select_sticker_unique_id_by_sticker_name(self, sticker_name):

print("_select_sticker_unique_id_by_sticker_name({})".format(sticker_name))
        ret_id = None
        if sticker_name != 'AllStickerNames':
            sql_str = "SELECT sticker_unique_id FROM t_stickers WHERE sticker_name
= '{}'".format(sticker_name)
            ret_id = str(self._run_sql(sql_str, 1)) # 1 - для SELECT одной ячейки
        else:
            sql_str = "SELECT sticker_unique_id FROM t_stickers"
            ret_id = str(self._run_sql(sql_str, 0)) # 0 - для SELECT столбца
таблицы
        print("_ret_id = {}".format(ret_id))
        return ret_id

    # получение имени стикера по уникальному id, сохраненному в БД, если есть
    def _select_sticker_name_by_sticker_unique_id(self, sticker_unique_id):

print("_select_sticker_name_by_sticker_unique_id({})".format(sticker_unique_id))
        sql_str = "SELECT sticker_name FROM t_stickers WHERE sticker_unique_id =
'{}'".format(sticker_unique_id)
        ret_id = str(self._run_sql(sql_str, 1)) # 1 - для SELECT одной ячейки
        return ret_id

    # получение уникального id из БД по индексу в общем списке стикеров
    def _select_sticker_unique_id_by_index(self, index):
        print("_select_sticker_unique_id_by_index({})".format(index))
        ret_str = None
        # получение всех id стикеров
        sql_str = "SELECT sticker_unique_id FROM t_stickers"
        rows = str(self._run_sql(sql_str, 0)).split('\n') # 0 - для SELECT всего
столбца таблицы
        print("rows = {}".format(rows))
        if index == -1:
            ret_str = rows
        else:
            ret_str = rows[index]
        return ret_str

    # получение id из БД по индексу в общем списке стикеров
    def _select_sticker_id_by_sticker_unique_id(self, sticker_unique_id):

print("_select_sticker_id_by_sticker_unique_id({})".format(sticker_unique_id))
        sql_str = "SELECT sticker_id FROM t_stickers WHERE sticker_unique_id =
'{}'".format(sticker_unique_id)
        ret_id = str(self._run_sql(sql_str, 1))  # 1 - для SELECT одной ячейки
        return ret_id

    ############### для добавления стикера
##############################################
    def insert_data(self, sticker_unique_id, sticker_download_id, sticker_name):
        ret_str = "-2000^None^ " # ^ отделяет номер ошибки от текста ответного
сообщения
        sql_str = "SELECT sticker_unique_id FROM t_stickers WHERE
sticker_unique_id = '{}'".format(sticker_unique_id)
        ret_id = self._run_sql(sql_str, 1) # 1 - для SELECT одной ячейки
        print("ret_id = {}".format(ret_id))
```

```
        if ret_id is not None:
            #index =
            # name =
            ret_str = "2001^{}^Такой стикер уже есть в базе!".format(ret_id)
        else:
            sql_str = "INSERT INTO t_stickers (sticker_unique_id, sticker_id,
sticker_name) VALUE ('{}', '{}', '{}')".\
                format(sticker_unique_id, sticker_download_id, sticker_name)
            add_id = self._run_sql(sql_str, 2) # 2 - для изменения данных
INSERT/DELETE/UPDATE
            ret_str = "2002^{}^Стикер добавлен в базу!".format(add_id)
        print("ret_str = {}".format(ret_str))
        return ret_str

    ############### для получения стикера по введенному значению
########################
    def select_data(self, input_value):
        ret_str = "-1000^None^ " # ^ отделяет номер ошибки от id и от текста
ответного сообщения
        sql_str = "SELECT COUNT(*) FROM t_stickers"
        num_of_data = int(self._run_sql(sql_str, 1)) #, "COUNT(*)"))  # 0 - для
SELECT одной ячейки
        print("num_of_data = {}".format(num_of_data))
        if num_of_data < 1:
            ret_id = "None\nВ базе данных стикеров нет!"
            ret_str = "-1001^None^{}".format(ret_id)
        else:
            try:
                input_value = int(input_value) ### если введено число, то ищем по
индексу
                if input_value < 0 or input_value >= num_of_data:
                    ret_str = "-1002^None^По индексу {} стикера нет!( Общее число
стикеров: {}".\
                        format(input_value, num_of_data)
                else:
                    ret_id = self._select_sticker_unique_id_by_index(input_value)
# для числа '0' находим все индексы
                    sticker_id =
self._select_sticker_id_by_sticker_unique_id(ret_id)
                    ret_str = "1001^{}^Получен ответ поиска по индексу {}:\
n{}".format(sticker_id, input_value, ret_id)
            except ValueError: ### иначе, если получить число не получилось, ищем
по имени
                ret_id =
self._select_sticker_unique_id_by_sticker_name(input_value) # для
'AllStickerNames' находим все индексы
                if ret_id is None:
                    ret_str = "-1003^None^Имя '{}' не существует в таблице
стикеров!( Общее число стикеров: {}".\
                        format(input_value, num_of_data)
                elif input_value != 'AllStickerNames':
                    sticker_id =
self._select_sticker_id_by_sticker_unique_id(ret_id)
                    ret_str = "1002^{}^{} ({})".format(sticker_id, ret_id,
input_value)
                else:
                    ret_data = self._get_list_data(ret_id)
                    ret_str = "1003^None^{}".format(ret_data)
        print("ret_str = {}".format(ret_str))
        return ret_str
```

tg_mysql_bot.py

```
from telegram.ext import Updater, CommandHandler
from telegram.ext import MessageHandler, Filters
```

```
from telegram import ChatAction, ParseMode, Sticker, StickerSet, Document, Audio
import config
import db_mysql
#import urllib.request

class tg_bot:
    def __init__(self):
        self.updater = Updater(token=config.API_TOKEN)
        self.dispatcher = self.updater.dispatcher
        self.dbase = db_mysql.dbase_mysql()
        # обработчик команды '/start'
        start_handler = CommandHandler('start', self.func_start)
        self.dispatcher.add_handler(start_handler)

        # обработка стикеров
        sticker_handler = MessageHandler(Filters.sticker, self.func_sticker)
        self.dispatcher.add_handler(sticker_handler)

        # обработка документов
        docs_handler = MessageHandler(Filters.document, self.func_document)
        self.dispatcher.add_handler(docs_handler)

        # обработка фотографий
        photo_handler = MessageHandler(Filters.document, self.func_photo)
        self.dispatcher.add_handler(photo_handler)

        # обработчик текстовых сообщений
        text_handler = MessageHandler(Filters.text & (~Filters.command),
self.func_text)
        self.dispatcher.add_handler(text_handler)

        # обработчик команды '/insert'
        insert_handler = CommandHandler('insert', self.func_insert)
        self.dispatcher.add_handler(insert_handler)

        # обработчик команды '/select'
        select_handler = CommandHandler('select', self.func_select)
        self.dispatcher.add_handler(select_handler)

        # обработчик не распознанных команд
        unknown_handler = MessageHandler(Filters.command, self.func_unknown)
        self.dispatcher.add_handler(unknown_handler)

        # запуск прослушивания сообщений
        self.updater.start_polling()
        # обработчик нажатия Ctrl+C
        self.updater.idle()

    def about(self):
        return "Для добавления (insert) и чтения (select) данных используйте
команды:" \
           "/insert u_id, d_id, s_name, где u_id - уникальный id стикера, d_id -
id для скачивания, s_name - имя;" \
           "/select data, где data - индекс/имя в таблице стикеров, начиная с 1; -
1 - для вывода всех стикеров"

    # функция обработки команды '/start'
    def func_start(self, update, context):
        context.bot.send_message(chat_id=update.effective_chat.id,
            text="Начало работы бота работы с данными)")

    def func_sticker(self, update, context):
        text_out = "Получен стикер: file_unique_id = " +\
                   update.message.sticker.file_unique_id +\
```

```
                    "\n Поиск по собственной базе стикеров для добавления!\n"
        # извлекаем файл в формате webp и сохраняем
        #file_info = context.bot.get_file(update.message.sticker.file_id)
        #print("file_path = {}".format(file_info.file_path))
        #str_url =
'http://api.telegram.org/file/bot{}/{}'.format(config.API_TOKEN,
file_info.file_path)
        #print("str_url = {}".format(str_url))
        #urllib.request.urlretrieve(str_url, file_info.file_path) # для скачивания
по прямой ссылке
        text_out = text_out + self.dbase.insert_data(
            update.message.sticker.file_unique_id, # уникальный ID, НЕ для
скачивания
            update.message.sticker.file_id,         # ID ДЛЯ скачивания по прямой
ссылке
            update.message.sticker.set_name)        # имя стикера
        text_out = text_out + self.about()
        context.bot.send_message(chat_id=update.effective_chat.id, text=text_out)

    def func_document(self, update, context):
        text_out = "Заглушка для обработки документов!"
        context.bot.send_message(chat_id=update.effective_chat.id, text=text_out)

    def func_photo(self, update, context):
        text_out = "Заглушка для обработки фотографий!"
        context.bot.send_message(chat_id=update.effective_chat.id, text=text_out)

    # функция обработки текстовых сообщений
    def func_text(self, update, context):
        text_out = ''
        index = update.message.text
        sticker_info = self.dbase.select_data(index)
        num_reply = int(sticker_info.split('^')[0]) # по 0-му индексу строк ответа
- номер ответа
        if num_reply >= 0 and index != -1 and index != 'AllStickerNames': # если
нет ошибки
            sticker_id = sticker_info.split('^')[1] # по 1-му индексу строк ответа
- уникальный id стикера
            print("tg_bot: sticker_id = {}".format(sticker_id))
            context.bot.send_sticker(chat_id=update.effective_chat.id,
sticker=sticker_id) # тогда отправляем стикер
        text_out = sticker_info.split('^')[2]  # по идексу 2 - текст ответа (без
ошибок или с ошибками)
        print(text_out)
        context.bot.send_message(chat_id=update.effective_chat.id, text=text_out)
# выводим текст ответа

    # функция обработки команды '/insert'
    def func_insert(self, update, context):
        if len(context.args) == 3:
            text_reply = self.dbase.insert_data(context.args[0], context.args[1],
context.args[2])
            text_data = text_reply.split('^')[2] # по 2му индексу - текст ответа)
            context.bot.send_message(chat_id=update.effective_chat.id,
                text=text_data)
            return text_reply.split('^')[0] # по 0му индексу - номер ответа
        else:
            context.bot.send_message(chat_id=update.effective_chat.id,
                text='для команды /insert необходимо указать 3 аргумента
(уникальный id, id для скачивания и имя);'
                     'указано число аргументов: {}'.format(len(context.args)))
            return -1

    # функция обработки команды '/select'
```

```
    def func_select(self, update, context):
        param_select = None
        if len(context.args) == 0: # без аргументов
            param_select = 'AllStickerNames' # выводим все стикеры по умолчанию
        elif len(context.args) == 1: # с 1м аргументом
            param_select = context.args[0] # аргумент, который ввел пользователь

        if len(context.args) <= 1: # без аргументов или с 1 аргументом
            text_reply = self.dbase.select_data(param_select)
            text_data = text_reply.split('^')[1] # для вывода ответа (по индексу 0
- номер результата)
            context.bot.send_message(chat_id=update.effective_chat.id,
text=text_data)
            return int(text_reply.split('^')[0])
        else: # 2 или более аргументов
            context.bot.send_message(chat_id=update.effective_chat.id,
                text='для команды /select необходимо указать 1 аргумент/n'
                     '(индекс или имя стикера), либо без аргументов (для вывода
всех стикеров);'
                     'указано число аргументов: {}'.format(len(context.args)))
            return -1

    # функция обработки не распознанных команд
    def func_unknown(update, context):
        context.bot.send_message(chat_id=update.effective_chat.id,
            text="Введена неизвестная команда!(")
        return -2

if __name__ == '__main__':
    new_bot = tg_bot()
```

Запуск осуществляется для файла tg_mysql_bot: ПКМ (правая клавиша мыши), выполнить запуск (Run; в дальнейшем - на зеленый треугольник справа вверху).

В результате таблица t_stickers заполнена значениями, фрагмент которой приведен на рис. 1, фрагменты работы в приложения в мессенджере приведены на рис. 2-3.

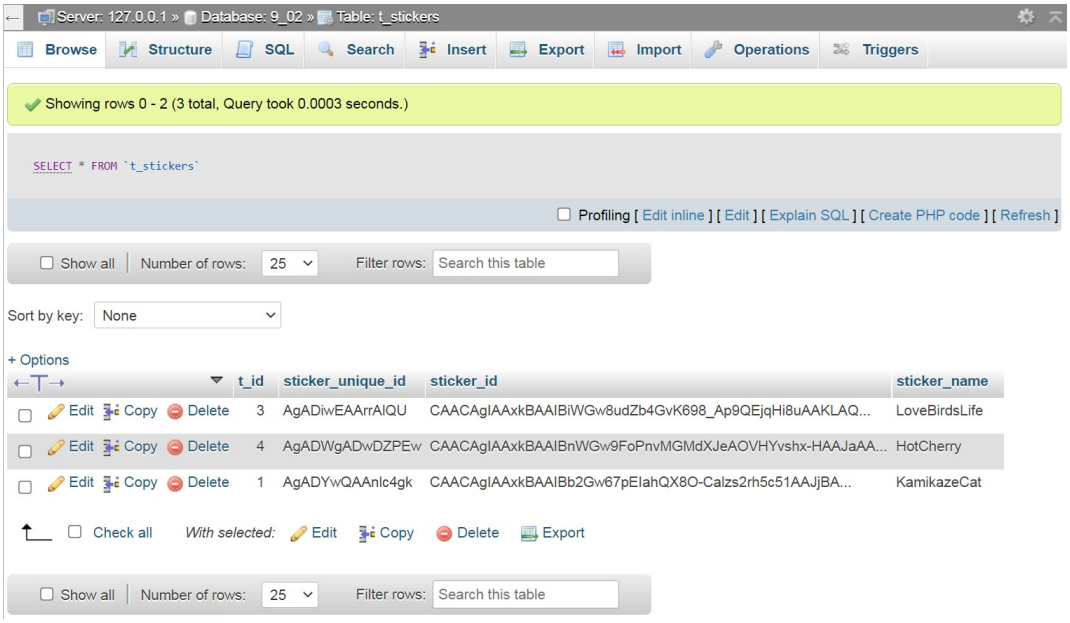

Рис. 1. Фрагмент тестового наполнения таблицы t_stickers [показать свой скриншот]

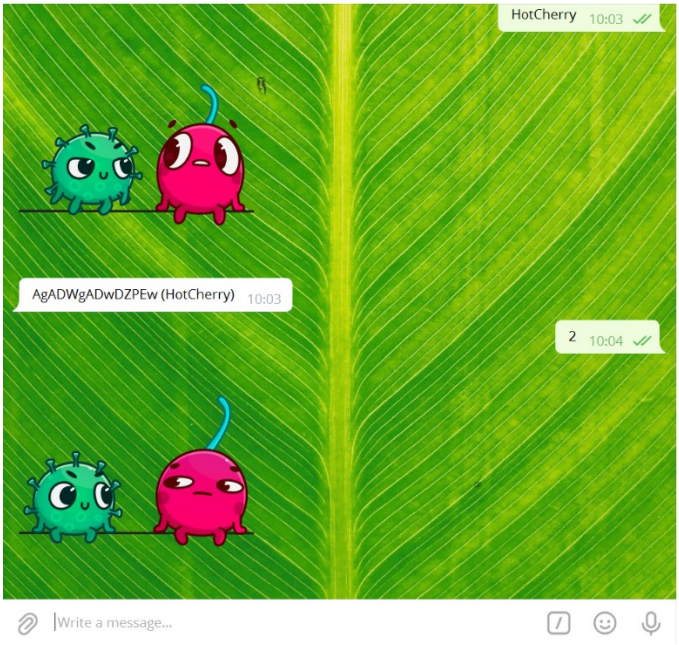

Рис. 2. Фрагмент работы бота в результате наполнения таблицы t_stickers [добавить и объяснить свой скриншот]

Рис. 3. Фрагмент работы бота в результате обращения к данным таблицы t_stickers [добавить и объяснить свой скриншот]

Полученные результаты на рис. 1 и 2 объясняются тем, что … [3-4 строки: объяснить соответствие запросов с персональной информацией и соответствующих стикеров].

Лабораторная работа № 3: Юнит-тестирование функций управления данными на языке Python

Для проверки работы функций управления данными MySQL составлены 4 класса по 2 функции-теста в каждом (успешный и неуспешный); реализация выполнена в виде файла pymysql.py (табл. 1):

[также создать 8 функций-методов (успешных и неуспешных): в 2, 3 или 4 классах]

```
import unittest
from db_mysql import dbase_mysql

class TestsCreate(unittest.TestCase):
    def test_create_success(self):
        print("Выполнение test_create_success")
        db = dbase_mysql()
        self.assertEqual(0, db.insert_data(['\'rozaroza\'',300,2]))

    def test_create_fail(self):
        print("Выполнение test_create_fail")
        db = dbase_mysql()
        self.assertNotEqual(0,
                            db.insert_data(['\'rozaroza\'', 2]))

class TestsUpdate(unittest.TestCase):
    def test_update_success(self):
        print("Выполнение test_create_success")
        db = dbase_mysql()
        self.assertEqual(10, db.update_data(1, 24))  # add assertion here

    def test_update_fail(self):
        print("Выполнение test_create_fail")
        db = dbase_mysql()
        self.assertNotEqual(10, db.update_data(1, 24))


class TestsRead(unittest.TestCase):
    def test_read_success(self):
        print("Выполнение test_create_success")
        db = dbase_mysql()
        self.assertEqual(['20, db.select_data(11))

    def test_read_fail(self):
        print("Выполнение test_create_fail")
        db = dbase_mysql()
```

```
        self.assertNotEqual(20, db.select_data(10))

class TestsDelete(unittest.TestCase):
    def test_delete_success(self):
        print("Выполнение test_create_success")
        db = dbase_mysql()
        self.assertEqual(30, db.delete_data(4))

    def test_delete_fail(self):
        print("Выполнение test_create_fail")
        db = dbase_mysql()
        self.assertNotEqual(30, db.delete_data(10))

if __name__ == '__main__':
    unittest.main()
```

Таблица 1

Результаты юнит-тестирования функций работы с базой данных MySQL

Номер теста п/п	Тестовая функция и значения параметров	Ожидаемый результат	Фактический результат	Результат теста
1	***assertEqual***: db.insert_data(['\'rozaroza\",300,2])	0	0	OK
2	***asserNotEqual:*** db.insert_data(['\'rozaroza\", 2]))	0	-1	OK
3	***assertEqual***: db.update_data(1, 24)	10	10	OK
4	***assertNotEqual***: db.update_data(1, 24)	10	-2	OK
5	***assertEqual***: db.select_data(11)	20	0	Failed
6	***assertNotEqual***: db.select_data(10)	20	10	Failed
7	***assertEqual***: db.delete_data(4)	30	30	OK
8	***assertNotEqual***: db.delete_data(10)	30	-1	Failed

В табл. 1 показаны следующие тесты:

- № 1, 2, 3, 4 и 7 выполнены успешно (OK), т.к. ожидаемое значение и фактическое совпали;

- № 5, 6, и 8 выполнены неуспешно (Failed), т.к. в БД нет идентификатора "11" (№5), данные существуют в БД и могут быть извлечены (№6), данные существуют в БД и могут быть успешно удалены (№8).