МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра «Информационные технологии»

Лабораторная работа №7

**«Взаимодействие клиентского приложения с СУБД»**

по дисциплине

Информационные технологии и программирование

Выполнил: студент гр. БЭИ2202

Кулешов А. С.

Вариант 16

Проверил: Халабия Р.Ф.

Москва, 2024 г

1. Цели и задачи

Изучить основные принципы взаимодействия приложений, разработанных в архитектуре «клиент-сервер», осуществляющее доступ к базе данных по технологии ODBC (или другой технологии взаимодействия с базами данных), изучить основные принципы работы клиентского приложения с API ODBC и с другими технологиями доступа к СУБД.

1. Постановка задачи

Задание:

Добавить в Ваше клиентское приложение, выполненное в прошлом задании, следующие пункты меню:

* специальное меню: с добавлением поля с указанием формата и типа поля, с удалением поля и редактирование поля с указанием формата, и типа поля. Необходимо выбирать все таблицы вашего варианта. Главное, чтобы таблица формируется как новая с помощью вашего меню, или если файл существует, структура считывается из файла.
* экспорт таблиц данных в СУБД MySQL, с сопоставлением всех возможных типов данных и форматов представления и их длины.
* импорт таблиц данных из СУБД MySQL с учетом всех изменений, которые могли произойти в СУБД MySQL.

1. Листинг программы

#include "tinyxml2.cpp"

// Функция для загрузки XML файла

bool LoadXML(const std::string& fileName, XMLDocument& doc) {

XMLError eResult = doc.LoadFile(fileName.c\_str());

cout << eResult << endl;

if (eResult != XML\_SUCCESS) {

std::cerr << "Error loading file: " << fileName << std::endl;

return false;

}

return true;

}

// Функция для сохранения XML файла

bool SaveXML(const std::string& fileName, XMLDocument& doc) {

XMLError eResult = doc.SaveFile(fileName.c\_str());

if (eResult != XML\_SUCCESS) {

std::cerr << "Error saving file: " << fileName << std::endl;

return false;

}

return true;

}

int get\_static\_id(XMLDocument& doc) {

using namespace tinyxml2;

XMLElement\* root = doc.FirstChildElement("Static");

auto attr = root->FindAttribute("id\_counter");

std::cout << "retrieved static\_id " << attr->IntValue() << std::endl;

return attr->IntValue();

}

std::string get\_table\_name(int val) {

vector<std::string> vc = { "Pets", "Owners", "Breeds", "Vaccines", "PetVaccines" };

return vc[val];

}

int update\_static\_id(XMLDocument& doc) {

int static\_id = get\_static\_id(doc) + 1;

XMLElement\* root = doc.FirstChildElement("Static");

root->SetAttribute("id\_counter", static\_id);

cout << root->FindAttribute("id\_counter")->IntValue() << endl;

return static\_id;

}

void AddObject(XMLDocument& doc, DBObject\* obj, const std::string& table) {

XMLElement\* root = doc.FirstChildElement(table.c\_str());

XMLElement\* newObject = doc.NewElement("Object");

obj->SetDefaultAttributes(doc, newObject);

obj->SetAttributes(doc, newObject);

root->InsertEndChild(newObject);

}

void checkFile(const std::string& fileName, XMLDocument& doc) {

// Если файл не существует, создаем новый документ

if (!LoadXML(fileName, doc)) {

XMLElement\* root = doc.NewElement("Pets");

doc.InsertEndChild(root);

root = doc.NewElement("Owners");

doc.InsertEndChild(root);

root = doc.NewElement("Breeds");

doc.InsertEndChild(root);

root = doc.NewElement("Vaccines");

doc.InsertEndChild(root);

root = doc.NewElement("PetVaccines");

doc.InsertEndChild(root);

root = doc.NewElement("Static");

root->SetAttribute("id\_counter", 1);

doc.InsertEndChild(root);

SaveXML(fileName, doc);

}

}

DBObject\* get\_class(int val) {

if (val == 0) return new Pet();

if (val == 1) return new Ownerw();

if (val == 2) return new Breed();

if (val == 3) return new Vaccine();

if (val == 4) return new PetVaccine();

}

class DBObject {

public:

int id;

virtual void SetAttributes(tinyxml2::XMLDocument& doc, tinyxml2::XMLElement\* elem) = 0;

void SetDefaultAttributes(tinyxml2::XMLDocument& doc, tinyxml2::XMLElement\* elem) {

id = update\_static\_id(doc);

elem->SetAttribute("id", id);

}

virtual std::vector<std::string> get\_description() = 0;

//virtual std::vector<std::string> get\_data() = 0;

};

void AddObject(XMLDocument& doc, DBObject\* obj, const std::string& table);

class Pet : public DBObject {

public:

int breedCode;

std::string name;

std::string birthDate;

int ownerCode;

void SetAttributes(tinyxml2::XMLDocument& doc, tinyxml2::XMLElement\* elem) override {

elem->SetAttribute("breedCode", breedCode);

elem->SetAttribute("name", name.c\_str());

elem->SetAttribute("birthDate", birthDate.c\_str());

elem->SetAttribute("ownerCode", ownerCode);

}

Pet() = default;

Pet(const std::string& s1, const std::string& s2, const std::string& s3, const std::string& s4) {

breedCode = atoi(s1.c\_str());

name = s2;

birthDate = s3;

ownerCode = atoi(s4.c\_str());

}

std::vector<std::string> get\_description() override {

return { "breedCode", "name", "birthDate", "ownerCode" };

}

};

class Ownerw : public DBObject {

public:

std::string fullName;

std::string address;

void SetAttributes(tinyxml2::XMLDocument& doc, tinyxml2::XMLElement\* elem) override {

elem->SetAttribute("fullName", fullName.c\_str());

elem->SetAttribute("address", address.c\_str());

}

Ownerw() = default;

Ownerw(const std::string& s1, const std::string& s2) {

fullName = s1;

address = s2;

}

std::vector<std::string> get\_description() override {

return { "fullName", "address" };

}

};

class Breed : public DBObject {

public:

std::string breedName;

void SetAttributes(tinyxml2::XMLDocument& doc, tinyxml2::XMLElement\* elem) override {

elem->SetAttribute("breedName", breedName.c\_str());

}

Breed() = default;

Breed(const std::string& s1) {

breedName = s1;

}

std::vector<std::string> get\_description() override {

return { "breedName" };

}

};

class Vaccine : public DBObject {

public:

std::string vaccineName;

std::string comment;

void SetAttributes(tinyxml2::XMLDocument& doc, tinyxml2::XMLElement\* elem) override {

elem->SetAttribute("vaccineName", vaccineName.c\_str());

elem->SetAttribute("comment", comment.c\_str());

}

Vaccine() = default;

Vaccine(const std::string& s1, const std::string& s2) {

vaccineName = s1;

comment = s2;

}

std::vector<std::string> get\_description() override {

return { "vaccineName", "comment"};

}

};

class PetVaccine : public DBObject {

public:

int petCode;

int vaccineCode;

std::string date;

void SetAttributes(tinyxml2::XMLDocument& doc, tinyxml2::XMLElement\* elem) override {

elem->SetAttribute("petCode", petCode);

elem->SetAttribute("vaccineCode", vaccineCode);

elem->SetAttribute("date", date.c\_str());

}

PetVaccine() = default;

PetVaccine(const std::string& s1, const std::string& s2, const std::string& s3) {

petCode = atoi(s1.c\_str());

vaccineCode = atoi(s2.c\_str());

date = s3;

}

std::vector<std::string> get\_description() override {

return { "petCode", "vaccineCode", "date"};

}

};

1. Схема классов

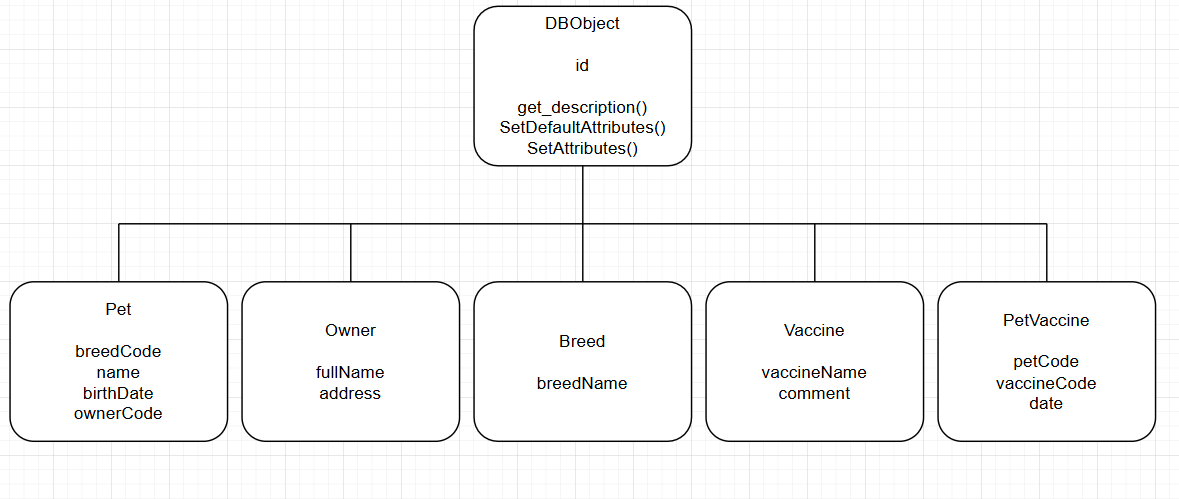


Рисунок 1 – Схема классов

1. Алгоритмы. Псевдокод

Метод **SaveSQL**

Шаг 1. Открываем схему SQL базы данных

Шаг 2. Удаляем данные из каждой таблицы внутри выбранной SQL схемы

Шаг 3. Загружаем данные в каждую SQL таблицу, согласно созданным правилам XML разметки (у таблицы «Pets» элементы – «Pet», у «Owners» – «Owner» и т.д.)

Шаг 4. Конец

Функция **LoadSQL**

Шаг 1. Открываем схему SQL базы данных

Шаг 2. Если файл db.xml существует, то очищаем его, иначе создаём пустой файл

Шаг 3. Переносим данные из каждой SQL таблицы во временный .xml файл

Шаг 4. Форматируем временные файлы под нашу структуру данных. Дочерние элементы становятся атрибутами элементов

Шаг 5. Конец

1. Контрольный тест. Результаты программы

Результаты запуска программы на контрольных данных можно увидеть на рисунках 2-13.

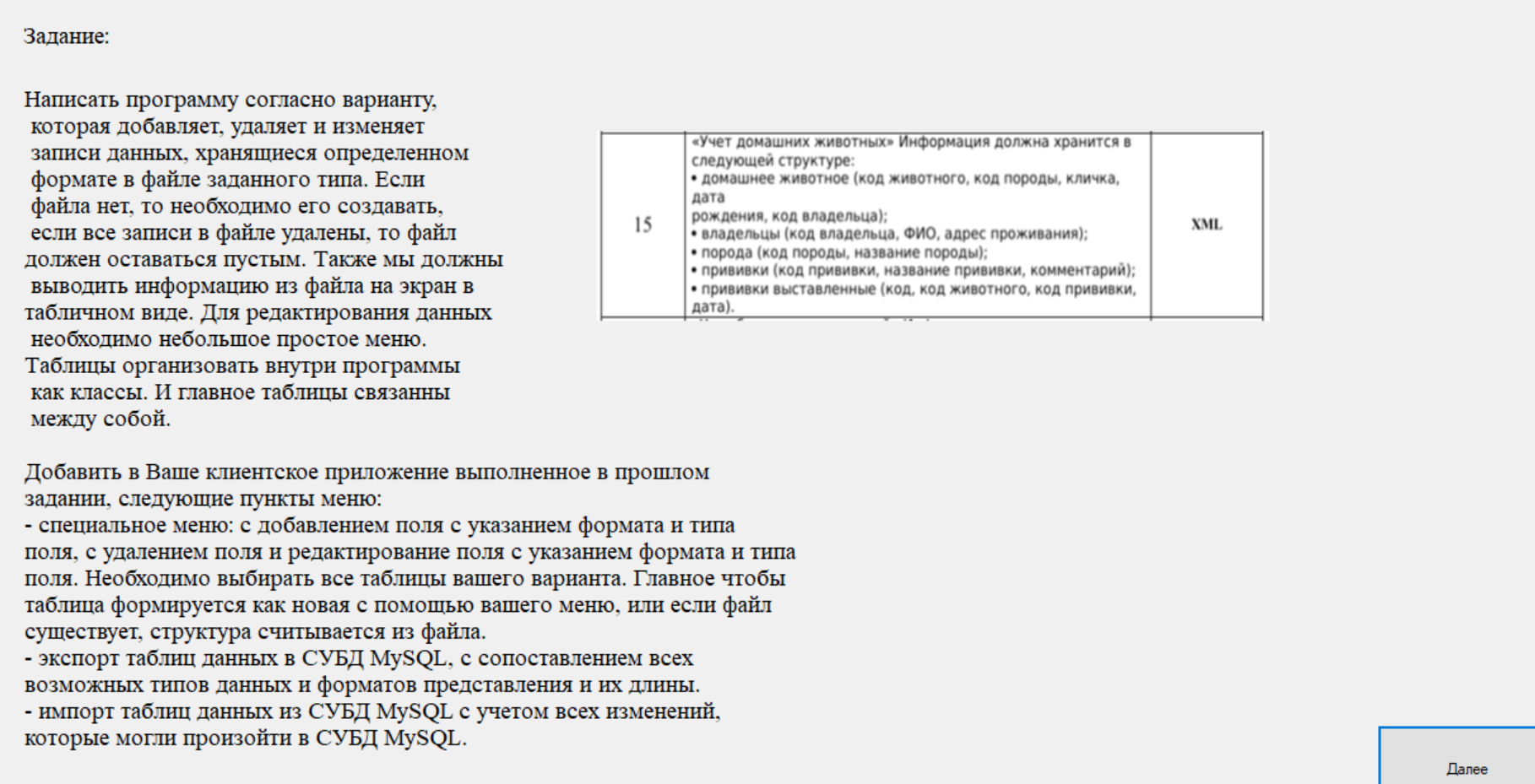


Рисунок 2 – входной экран

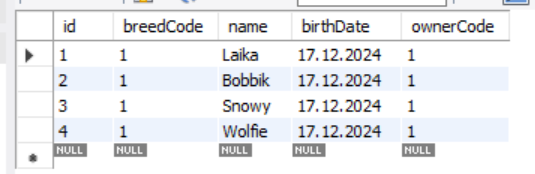


Рисунок 3 – просмотр SQL таблицы

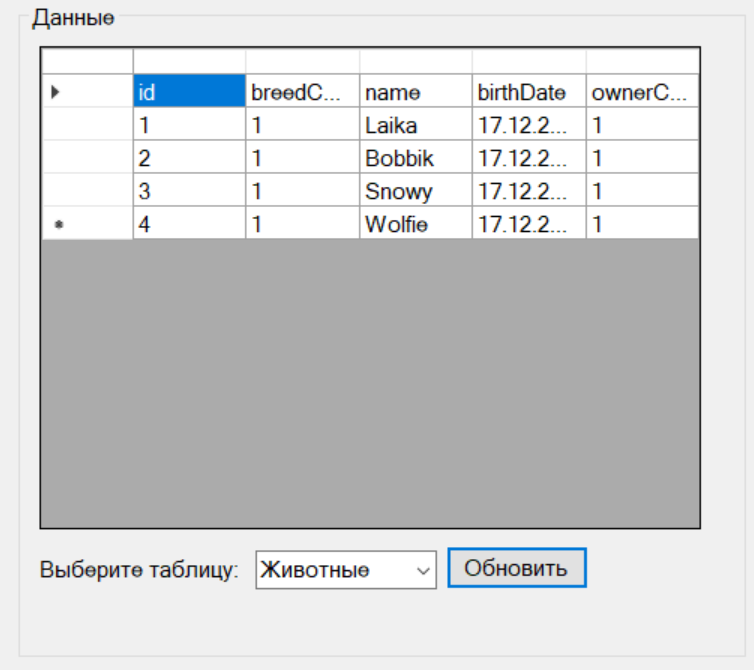


Рисунок 4 – Изначальное содержание xml базы данных

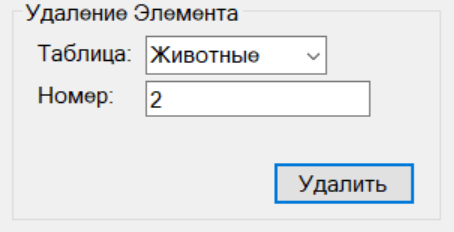


Рисунок 5 – Удаление элемента

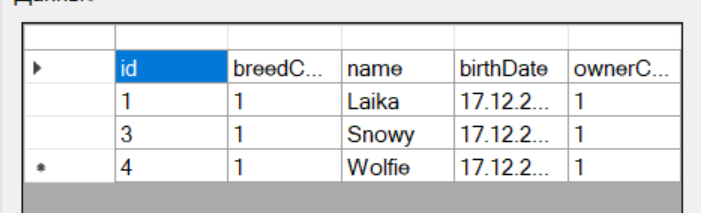


Рисунок 6 – Элемент удалён

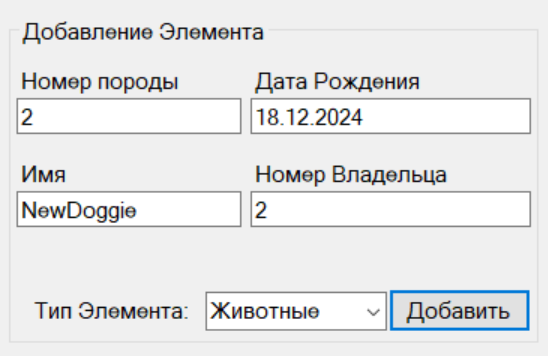


Рисунок 7 – Добавление элемента

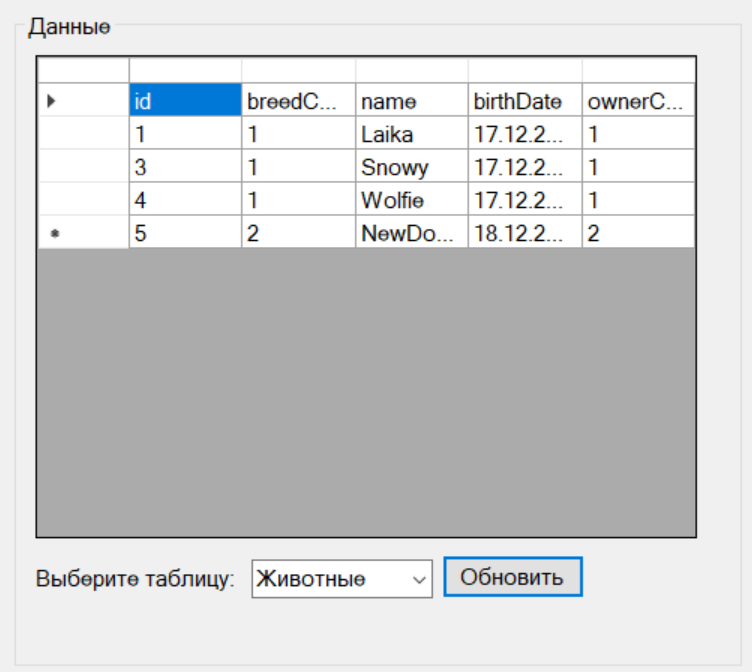


Рисунок 8 – Элемент добавлен

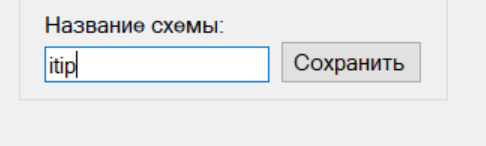


Рисунок 9 – Сохранение xml базы данных в sql

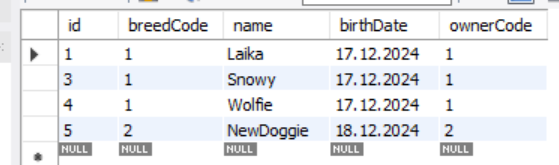


Рисунок 10 – Обновлённая SQL база данных

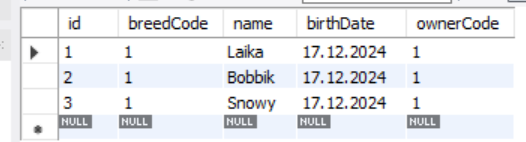


Рисунок 11 – Принудительное очищение SQL базы данных

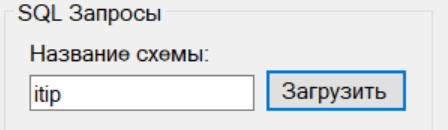


Рисунок 12 – Загрузка данных из SQL

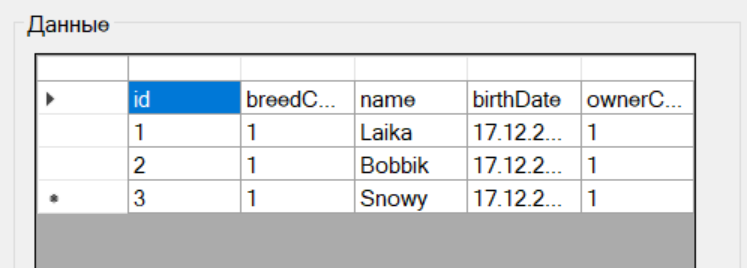


Рисунок 13 – Загруженные данные

1. Выводы по работе

В процессе работы были изучены основные принципы взаимодействия приложений с СУБД.