

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

Направление подготовки 09.03.01 «Информатика и вычислительная техника»
Профиль Вычислительные машины, комплексы, системы и сети

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ
РАБОТА БАКАЛАВРА**

на тему

**Клиент-серверное приложение переноса данных
между базами данных**

Студентка ИИВ- 02.06.18 Нетёсова Ирина Сергеевна
(подпись, дата)

Руководитель Дорошенко И.Н. Дорошенко И.Н.
(подпись, дата)

Нормоконтролёр Кучин А.В. Кучин А.В.
(подпись, дата)

Работа допущена к защите (протокол заседания кафедры от
25.05.2018 № 38-09-18)

Заведующий кафедрой Пашенко Д.В.
(подпись)

Работа защищена с отметкой отм. (протокол заседания ГАК от 19.06.18 № 2)

Секретарь ГАК Гурин Е.И.
(подпись)

Пенза, 2018

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

«Утверждаю»
Заведующий кафедрой
Д.В. Пашенко
« » 2018 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студентка Нетёсова Ирина Сергеевна гр. 14ВВ1 факультета вычислительной техники.
2. Тема работы: «Клиент-серверное приложение переноса данных между базами данных».
Тема утверждена приказом ПГУ №398а/о от "03 " мая 2018 г.
3. Руководитель работы к.т.н., доцент каф. ВТ ПГУ Дорошенко И.Н.
4. Задание на работу (назначение разработки, исходные данные и т.п.).
Назначение разработки – возможность открытия базы данных нескольких форматов, работа с ними и перенос данных в базы других типов.
Разработать клиентское приложение, позволяющее пользователю работать с базой данных. Разработать серверное приложение, которое выполняет обработку полученных от приложений-клиентов запросов к базе данных.
5. Перечень подлежащих разработке вопросов.
Требуется произвести анализ предметной области, выбрать системы управления базами данных, определить форматы, которые будут доступны для работы в приложении, спроектировать клиентское и серверное приложение, реализовать приложения, отладить и провести тестирование разработанных приложений.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10	07.05.18	Дорошенко
2.	Определение перечня используемых СУБД	10	14.05.18	Дорошенко
3.	Проектирование клиентского и серверного приложений	20	14.05.18	Дорошенко
4.	Написание программы клиентского приложения	10	21.05.18	Дорошенко
5.	Написание программы серверного приложения	10	21.05.18	Дорошенко
6.	Отладка и тестирование комплекса приложений	10	28.05.18	Дорошенко
7.	Оформление выпускной работы	30	04.06.18	Дорошенко

Дата выдачи задания "5" марта 2018 г.

Руководитель бакалаврской работы Дорошенко 5.03.18 Дорошенко И.Н.
(подпись, дата)

Задание к исполнению приняла студентка ИИ - 05.03.18 Нетёсова И.С.
(подпись, дата)

Работу к защите допустить
Декан факультета ОИ 08.06.2018 Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Бакалаврская работа содержит 109 листов, 1 таблицу, 64 рисунка, 9 источников, 2 приложения.

СЕРВЕР, БАЗА ДАННЫХ, СЕТЕВОЕ ПРИЛОЖЕНИЕ, ПРОТОКОЛ TCP/IP, SQL-ЗАПРОСЫ, КЛИЕНТ-СЕРВЕР, МИГРАЦИЯ ДАННЫХ.

Цель работы – разработать клиент-серверное приложение, осуществляющее доступ к различным базам данных и работу с ними, в частности, перенос данных из одной базы в другую.

Объектом разработки является комплекс приложений (клиент и сервер), которые должны, взаимодействуя между собой, осуществлять работу с базами данных.

В результате проведенной работы создан проект, удовлетворяющий поставленной задаче, а именно, создано клиентское приложение, имеющее интуитивно понятный интерфейс для отображения и работы с данными из базы, и серверное приложение, принимающее и направляющее SQL-запросы от клиента в требуемую базу данных. Взаимодействие приложений осуществляется по протоколу TCP/IP.

Данный проект рекомендуется использовать при необходимости доступа к различным базам данных и переноса данных между ними. На предприятии, где используются базы данных различных типов и форматов, с целью сокращения используемых СУБД до одной может быть внедрен данный проект.

ПГУ 1.090301.16.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата
Разраб.		Нетёсова И.С.	<i>[Подпись]</i>	26.12
Провер.		Дорошенко И.Н.	<i>[Подпись]</i>	26.12
Н. Контр.		Кучин А.В.	<i>[Подпись]</i>	26.12
Утверд.		Пашенко Д.В.	<i>[Подпись]</i>	

Клиент-серверное приложение
переноса данных между базами
данных

Пояснительная записка

Лист	Лист	Листов
1	4	109

ФВТ, коф. ВТ
 гр. 14551

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ ПРИНЯТЫХ СОКРАЩЕНИЙ.....	7
ВВЕДЕНИЕ	8
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	10
1.1 Определение перечня СУБД	10
1.2 Способы доступа к базам данных	11
1.2.1 ODBC.....	12
1.2.2 OLE DB.....	14
1.3 Архитектура приложения	15
1.4 Связь между клиентом и сервером	18
2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЙ	21
2.1 Проектирование клиентского приложения.....	22
2.2 Проектирование серверного приложения.....	30
3 РЕАЛИЗАЦИЯ ПРОЕКТА.....	34
3.1 Процесс разработки	34
3.1.1 Разработка клиентского приложения.....	35
3.1.2 Разработка серверного приложения.....	38
3.2 Отладка и тестирование.....	40
3.2.1 Тестирование возможности добавления новой таблицы в базу данных	40
3.2.2 Тестирование возможности изменения данных в ячейке	42
3.2.3 Тестирование возможности переноса данных в другую базу	44
4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.....	49
ЗАКЛЮЧЕНИЕ	64
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	65
ПРИЛОЖЕНИЕ А ЛИСТИНГИ ПРОГРАММЫ	67
Приложение А.1 Файл connectionData.cs.....	67

					ПГУ 1.090301.16.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Нетёсова И.С.			Клиент-серверное приложение переноса данных между базами данных Пояснительная записка	Лит.	Лист	Листов
Провер.		Дорошенко И.Н.					5	109
Н. Контр.		Кучин А.В.				ФВТ, каф. ВТ гр. 14ВВ1		
Утверд.		Пащенко Д.В.						

Приложение А.2 Файл MainForm.cs	71
Приложение А.3 Файл server.cs	101
ПРИЛОЖЕНИЕ В ПРЕЗЕНТАЦИЯ.....	104

					ПГУ 1.090301.16.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

ПЕРЕЧЕНЬ ПРИНЯТЫХ СОКРАЩЕНИЙ

СУБД	Система управления базами данных
БД	База данных

					ПГУ 1.090301.16.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

ВВЕДЕНИЕ

Первые ЭВМ создавались для решения математических задач, однако развитие вычислительной техники открыло новые пути использования компьютеров, среди которых особое место занимает хранение и обработка информации. Так, появление производительных устройств внешней памяти, а именно жёстких дисков, позволило создать первые базы данных и системы управления ими (СУБД). Работа с базами осуществлялась через консольный терминал.

В настоящее время, спустя всего несколько десятилетий, базы данных и СУБД претерпели большие изменения. Развитие техники и языков программирования привело к тому, что на серверах по всему миру хранятся огромные базы данных, а интерфейс СУБД больше не представляют из себя консоль – теперь это удобные оконные приложения с большим набором инструментов для работы с данными.

Изменился и способ доступа к данным – всё чаще используются распределённые системы, в которых, например, пользовательский интерфейс располагается на стороне клиента, а непосредственно база данных – на другой машине, на стороне сервера. Причём клиент и сервер могут входить как в локальную сеть, так и связываться по сети Интернет.

Сейчас сложно представить деятельность какой-либо организации без использования базы данных. Системы управления базами данных позволяют создавать отказоустойчивые хранилища данных. Кроме того, хранение информации в электронной форме позволяет значительно экономить время, поскольку современные СУБД предоставляют возможность быстро производить различные операции над данными, а именно: изменение, копирование, добавление и удаление, сортировка, фильтрация и т.д. Всё это заняло бы гораздо больше времени, если бы пользователям, работающим с данными, приходилось осуществлять эти действия вручную.

					ПГУ 1.090301.16.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

Однако активное использование баз данных привело к появлению большого числа СУБД. Если поддержка какой-либо системы управления прекратится, и компания, её использующая, примет решение перейти к использованию другой, то весьма вероятно возникновение проблемы переноса данных для их доступности в другой СУБД.

В данной работе будет разработано сетевое приложение, позволяющее работать с данными в различных базах данных, а также переносить данные между ними.

					ПГУ 1.090301.16.001 ПЗ	Лист
						9
Изм.	Лист	№ докум.	Подпись	Дата		

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Существует множество различных СУБД, предназначенных для работы с базами данных, а именно позволяющих создавать, редактировать, сохранять и производить другие действия над данными, содержащимися в базе. Как правило, каждая СУБД работает с файлами баз определенного формата: для Microsoft Access это формат .accdb или .mdb, для Microsoft SQL Server – .mdf и т.д. Однако СУБД, способные работать с базами, представленными файлами различных форматов, не распространены.

Актуальность моей работы заключается в решении изложенной проблемы и создании приложения, способного открывать файлы различных форматов, содержащие базы данных, а также позволяющего работать с данными и переносить их в файлы других форматов.

Выбор оптимального решения требует проведения анализа предметной области и определения некоторых ключевых моментов, на которых в дальнейшем будет основываться проектирование и разработка приложений.

1.1 Определение перечня СУБД

Цель проекта требует выбора несколько распространенных форматов баз данных. Согласно данным на январь 2018 года [2], в десятке наиболее популярных находятся такие СУБД, как Microsoft SQL Server, MySQL и Microsoft Access. Выбор данных конкретных вариантов продиктован следующими факторами.

СУБД Microsoft Access входит в пакет приложений Microsoft Office, весьма широко использующийся в компаниях, а потому довольно распространена в офисной среде. Для работы в ней не требуется знаний языка SQL (хотя возможность работы с его использованием также предусмотрена) или иных специальных знаний, интерфейс интуитивно понятен и доступен работнику.

					ПГУ 1.090301.16.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

Microsoft SQL Server является классическим коммерческим вариантом СУБД, поскольку позволяет оперировать базами данных довольно крупного размера. Основой работы в такой СУБД является язык запросов Transact-SQL – расширенная версия языка SQL.

MySQL – пример свободно распространяемой СУБД. Работать в ней можно, используя процедурное расширение языка SQL – SQL/PSM (SQL/Persistent Stored Module).

Таким образом, приложение, позволяющее работать с базами данных, созданными в вышеуказанных СУБД, удовлетворит потребности большой доли пользователей, поскольку покрывает разные сегменты рынка СУБД.

1.2 Способы доступа к базам данных

Способы доступа к базе данных из приложения схематически могут быть представлены следующим образом (рисунок 1) [1].

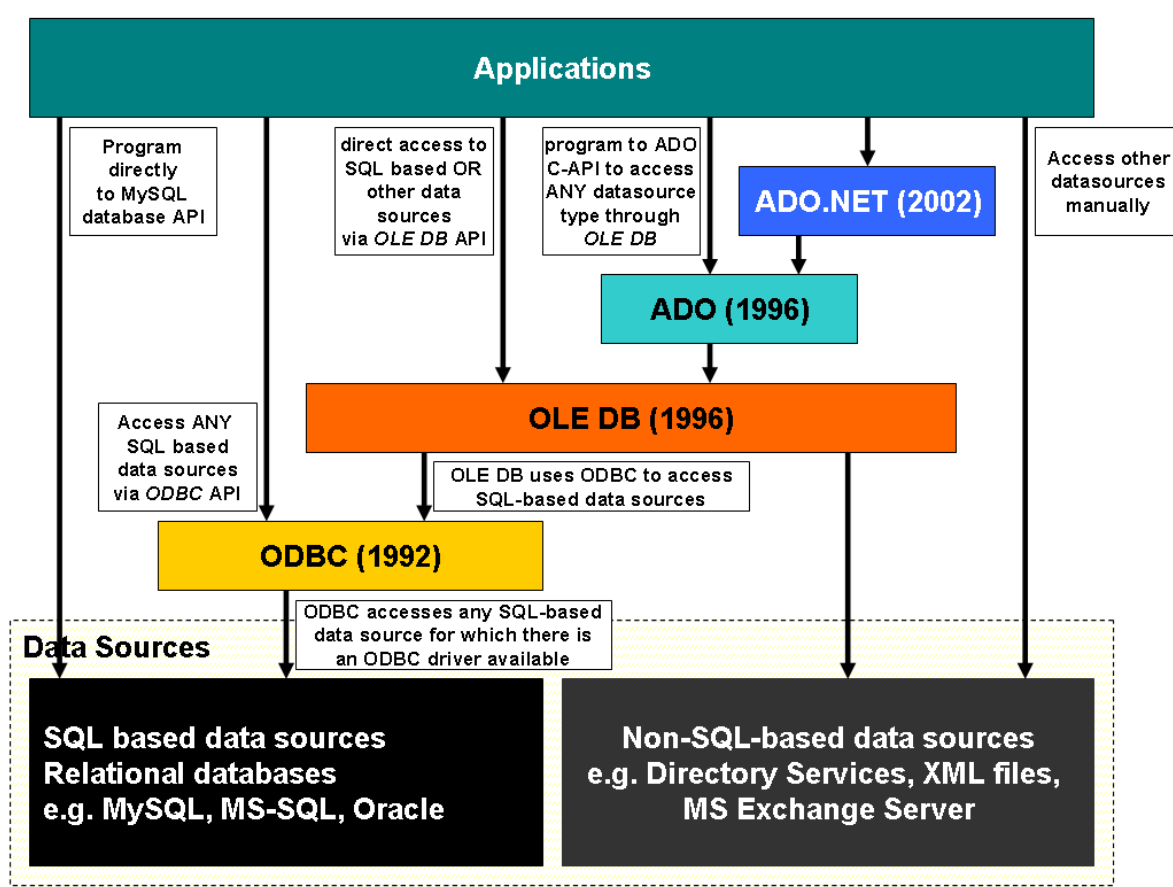


Рисунок 1 – Способы доступа к базе данных

ODBC (Open Database Connectivity) и OLE DB (Object Linking and Embedding, Database) – это драйверы для доступа к данным.

ADO (ActiveX Data Object) – набор классов для работы с данными, предоставляющий лишь инструменты для более удобного взаимодействия с источниками данных, но также требующий для своей работы один из драйверов, упомянутых выше, поэтому исключим его из рассмотрения и выбора подходящего для реализации проекта способа доступа к базе данных.

1.2.1 ODBC

Интерфейс ODBC был создан в начале 90-х годов прошлого столетия, когда существовало несколько поставщиков баз данных, каждый из которых имел собственный интерфейс. При необходимости работы с несколькими источниками данных одновременно требовалось писать программный код для доступа к каждому из них. Для решения этой проблемы Microsoft совместно с рядом других компаний создали интерфейс Open DataBase Connectivity (сокращенно ODBC), что можно перевести как «открытая для модификаций связь с базами данных». Он позволил унифицировать доступ к различным источникам данных. С его помощью программисты могли разрабатывать приложения, не беспокоясь о тонкостях работы с каждым конкретным типом баз данных и не создавая для них специальный программный код.

ODBC является интерфейсом уровня вызова, который позволяет приложениям иметь доступ к любой базе данных, для которой в ODBC есть драйвер. ODBC обеспечивает API, который позволяет приложению быть независимым от системы управления базами данных (СУБД).

Архитектура ODBC включает четыре основных компонента:

- прикладная программа (клиентское приложение);
- менеджер (диспетчер) драйверов;
- драйвер(ы);
- источник или источники данных.

Интерфейс ODBC физически представляет собой набор динамических библиотек DLL, которые обслуживают подключение и работу с конкретным типом базы данных. При запросе на подключение к определенной, заранее описанной базе "активизируется" определенная DLL – драйвер этого типа БД. Для добавления нового типа БД достаточно лишь написать драйвер и зарегистрировать его, что подтверждает определение архитектуры как открытой для модификаций, легко расширяемой. Однако приложения, использующие ODBC, не обращаются к источникам данных напрямую, они работают через менеджер драйверов, применяя SQL (рисунок 2) [9].

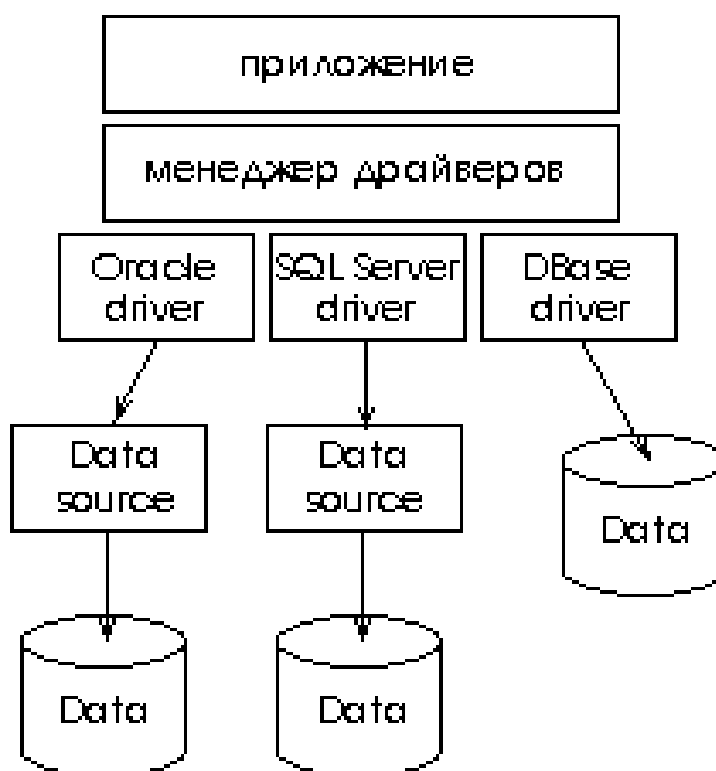


Рисунок 2 – Архитектура ODBC

Пользовательское приложение обращается к физической базе данных через менеджер драйверов безотносительно к ее типу. Иными словами, с точки зрения пользователя все источники данных обладают приблизительно одинаковыми свойствами. Таким образом, можно поменять физический тип базы данных, но взаимодействие с ней можно осуществлять по уже имеющимся алгоритмам. Следует, однако, иметь в виду, что различные типы баз данных могут поддерживать расширения языка SQL, имеющие

некоторые различия в синтаксисе команд, но, как правило, такие различия незначительны.

ODBC, используя конструкции языка SQL для связи с различными системами управления базами данных, обеспечивает следующие функции работы с базами данных:

- создание, изменение и удаление таблиц;
- создание и удаление представлений и индексов;
- отбор, обновление, вставка и удаление полей таблиц.

ODBC позволяет работать с такими источниками данных, как Microsoft Access, Excel, SQL Server, FoxPro и другими.

1.2.2. OLE DB

Интерфейс OLE DB был разработан компанией Microsoft как замена ODBC.

OLE DB позволяет работать с:

- базами данных для мейнфреймов;
- базами данных типа MS Access, Paradox, базами под управлением Oracle Database, Microsoft SQL Server и др.;
- системами электронной почты типа Exchange;
- индексно-последовательными файлами;
- и другими источниками данных.

Главным отличием OLE DB от ODBC, как видно из рисунка 1 и представленного выше списка, является возможность работать с нереляционными базами данных и другими источниками данных, тогда как ODBC предоставляет доступ только к реляционным, основанным на SQL источникам. Однако существуют и некоторые другие различия; они представлены в таблице 1 [3].

Таблица 1 – Интерпретация основных терминов в ODBC и OLE DB

Термин	ODBC	OLE DB
Источник данных	Реляционная СУБД	Модуль, который взаимодействует с хранилищем информации и показывает его содержимое как набор строк и столбцов
Таблица	Таблица в реляционной базе данных	Набор данных, полученный агрегированием связанных строк и однородных столбцов
Результирующий набор данных	Часть памяти клиента, содержащая последовательность записей	Независимый автономный модуль, показывающий последовательность записей
Команда	Команда SQL, интерпретируемая подключенной СУБД	Текстовая строка, которую понимает источник данных

Так, для реализации проекта в качестве средства доступа к базам данных был выбран интерфейс OLE DB, поскольку он позволяет охватить более широкий круг источников данных, что может иметь ключевое значение в перспективе, при решении совершенствования разработанного приложения и добавления возможности работы с нереляционными базами данных или иными источниками.

1.3 Архитектура приложения

Проект, согласно заданию, должен иметь клиент-серверную архитектуру. Из названия следует, что он должен состоять из двух частей – клиента и сервера. Обратимся к их определениям.

Клиент – сторона (ЭВМ, программа или пользователь), запрашивающая и использующая информацию и/или ресурсы у сервера в среде клиент-

сервер [4]. Следовательно, сервер – сторона (ЭВМ, программа или пользователь), эту информацию и/или ресурсы предоставляющая.

Такая архитектура обладает как достоинствами, так и определенными недостатками [7]. К достоинствам можно отнести:

- Отказоустойчивость. Если одна машина выйдет из строя, остальная сеть все равно сможет продолжать работу;
- Мощность. Некоторые рабочие станции столь же мощны, как мэйнфреймы, но их стоимость на порядок ниже. Предоставляя вычислительные мощности за меньшие деньги, система позволяет тратить сэкономленные средства на другие приобретения или на увеличение доходов;
- Открытость. Аппаратуру, программы и услуги можно приобретать у разных поставщиков;
- Масштабируемость. Систему нетрудно модернизировать при изменении потребностей;
- Гибкость. Можно объединять компьютерные платформы, подбирая их под конкретные нужды подразделений и пользователей.

Недостатками клиент-серверной архитектуры являются:

- Слабая поддержка. Отдельные части системы не всегда корректно работают вместе. Бывает довольно трудно найти причину неисправности;
- Недостаток инструментальных средств обслуживания. При использовании архитектуры клиент-сервер часто приходится искать инструментальные средства на рынке или разрабатывать их самостоятельно;
- Необходимость переобучения. Философия программирования для Mac или Windows существенно отличается от философии программирования на языке COBOL или С.

Клиент-серверная архитектура подразумевает, что на сервере находится как база данных, так и СУБД, а на рабочих станциях установлены

клиентские приложения с бизнес-логикой, осуществляющие доступ к базам. Таковую архитектуру следует отличать от файл-серверной, особенностью которой является то, что СУБД находится не на серверной, а на клиентской стороне.

Схема простейшей модели взаимодействия клиента и сервера представлена на рисунке 3 [5]:

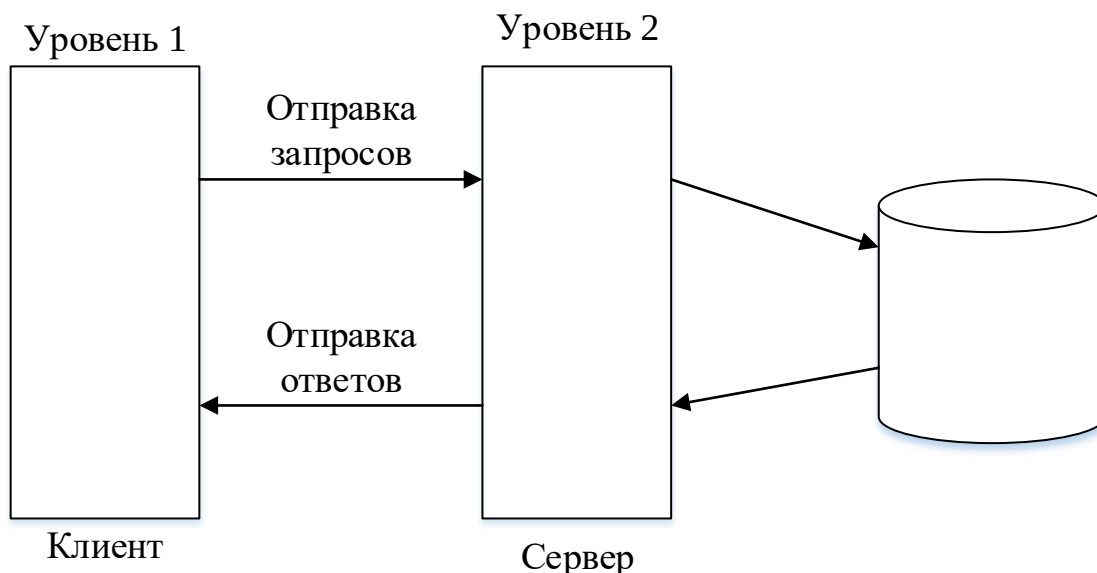


Рисунок 3 – Клиент-серверная архитектура

Из рисунка видно, что клиент посылает запросы на сервер, тот направляет их к базе данных, а затем отправляет ответ клиенту.

Архитектура, изображенная на рисунке 5, называется двухзвенной. Существует также многоуровневая архитектура, частным случаем которой является трехзвенный вариант (рисунок 4) [5]:

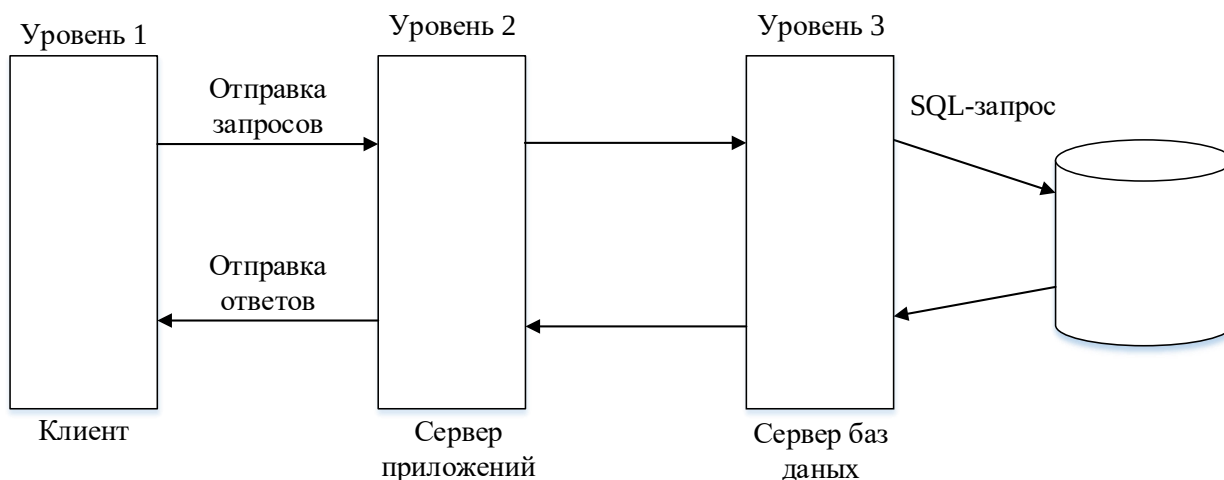


Рисунок 4 – Трехзвенная клиент-серверная архитектура

Такой вариант подразумевает разделение серверной части на сервер приложений и сервер баз данных. Он позволяет снизить нагрузку на сервер и обеспечить работу большего числа клиентов, однако данный подход менее надежен, чем двухзвенная архитектура. Кроме того, время ожидания клиентом ответа несколько увеличивается из-за наличия промежуточного звена (рисунок 5) [6]:

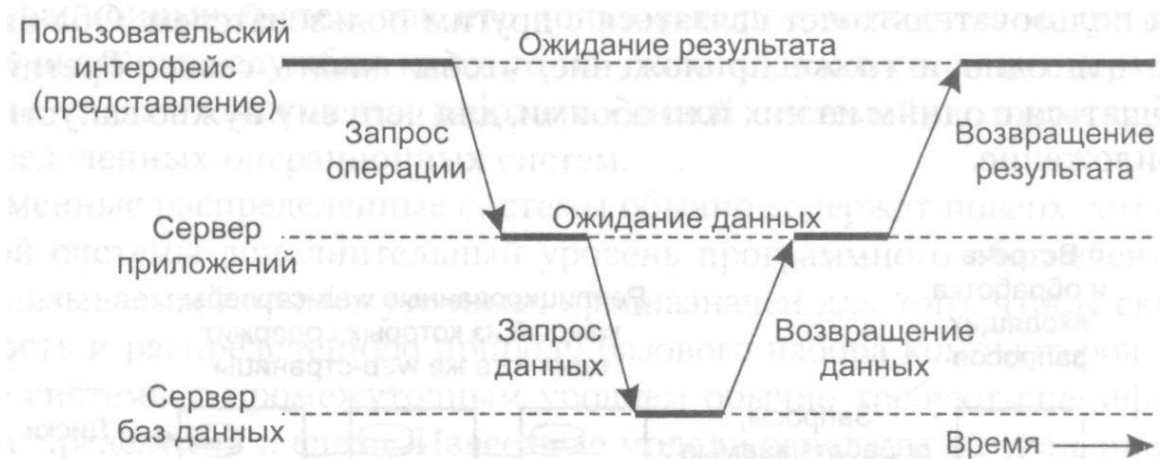


Рисунок 5 – Составляющие времени ожидания ответа

Базы данных Microsoft Access хранятся в виде файлов на компьютере. Базы данных Microsoft SQL Server и MySQL физически являются файлом или совокупностью файлов, однако доступ к ним необходимо осуществлять через соответствующий сервер. Из этого следует, что работа с базами данных Access будет организована, как в двухзвенной клиент-серверной архитектуре, а работа с двумя другими типами баз – как в трехзвенной; приложение-сервер, которое будет разработано в рамках данного проекта, будет выступать в качестве сервера приложений.

1.4 Связь между клиентом и сервером

Расположение клиента и сервера относительно друг друга может быть различным. Существует три варианта конфигурации (рисунок 6) [8]:

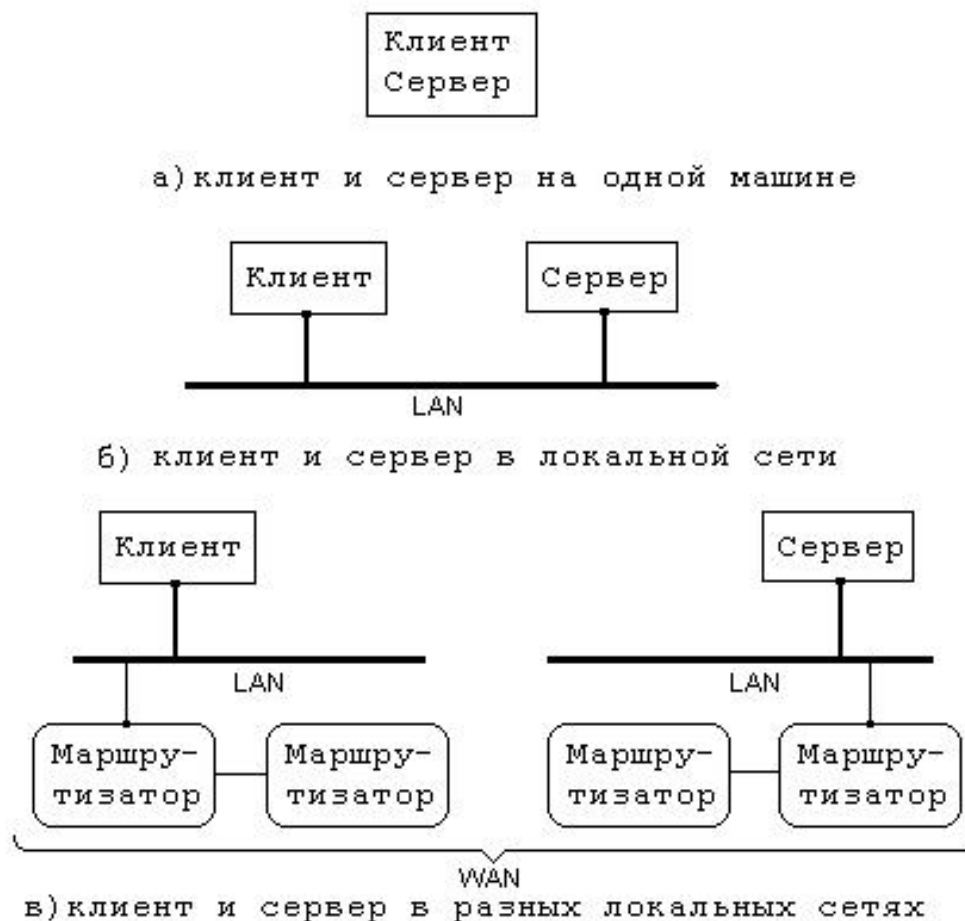


Рисунок 6 – Варианты связи клиента и сервера

На рисунке 6а изображен самый простой случай организации клиента и сервера, поскольку физическая сеть отсутствует. Такой способ организации применим для однопользовательских программ. Однако для приложений баз данных, применяющихся на предприятиях, такой подход нецелесообразен.

На втором примере конфигурации (рисунок 6б) клиент и сервер расположены на разных машинах, но находятся в одной локальной сети. В третьем примере (рисунок 6в) клиент и сервер работают на разных компьютерах, связанных глобальной сетью. Этой сетью может быть Internet или корпоративная Intranet, но главное – приложения уже не находятся внутри одной локальной сети. Для обеих указанных конфигураций необходимо организовать связь по сети, локальной или глобальной. Одним из способов обеспечения связи по сети является использование стека протоколов TCP/IP.

Протокол TCP (Transfer Control Protocol) обеспечивает установление надежного соединения между двумя машинами и собственно передачу данных, контролируя оптимальный размер пакета передаваемых данных и осуществляя повторную посылку в случае сбоя. Число одновременно устанавливаемых соединений между абонентами сети не ограничивается, т. е. любая машина может в некоторый промежуток времени обмениваться данными с любым количеством других машин по одной физической линии. Другое важное преимущество сети с протоколами TCP/IP состоит в том, что по нему могут быть объединены машины с разной архитектурой и разными операционными системами, например, Unix, VAX VMS, MacOS, MS-DOS, MS Windows и т.д. [9]

Суть передачи данных с использованием TCP/IP заключается в следующем: на стороне сервера создается «слушающий» сокет – пара значений, образованная IP-адресом и портом, ожидающий входящего соединения. Клиент может связаться с сервером, послав данные по указанному адресу на заданный порт. При возникновении соединения сервер выходит из состояния ожидания и выполняет те или иные действия по обработке данных, принимаемых от клиента. Таким образом, передача данных с помощью TCP/IP является надежной и простой в реализации, а потому для связи клиента и сервера в данном проекте целесообразным будет использование именно такого способа организации передачи данных.

2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЙ

Архитектура «клиент-сервер» подразумевает, что доступ к базе данных будет осуществляться со стороны сервера, к которому клиентское приложение будет направлять запросы. Связь между клиентом и сервером должна быть организована по протоколу TCP/IP, что позволит обеспечить работу комплекса приложений по сети.

Приложение-клиент должно обладать интуитивно понятным интерфейсом и позволять производить различные действия над данными базы, в частности, перенос данных из одной базы в другую. Запросы к базе данных должны конструироваться на основании того или иного действия пользователя и представлять из себя команды на языке SQL, поскольку большинство СУБД поддерживает возможность взаимодействия с базой таким образом. Затем сформированные запросы следует отправлять на сервер для дальнейшей их передачи в базу.

Приложение-сервер может не иметь графического интерфейса, поскольку не требует для своей работы ввода каких-либо данных. Его работа должна заключаться в создании «слушающего» сокета и ожидания входящего соединения от клиента. При установлении соединения должен осуществляться прием данных от клиента и их обработка, в частности, отправка запроса клиента в базу данных. В случае успешного выполнения действий сервер должен отправить клиенту соответствующее сообщение, а затем снова перейти в режим ожидания.

Взаимодействие частей системы в общем виде отражено на диаграмме кооперации (рисунок 7).

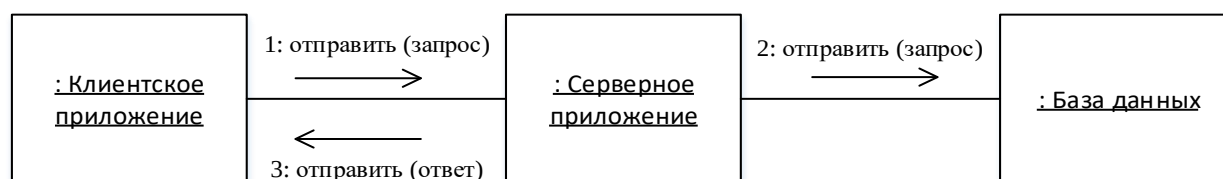


Рисунок 7 – UML-диаграмма кооперации

Рассматривая работу приложений во временном аспекте, их взаимодействие можно представить на UML-диаграмме последовательности (рисунок 8):

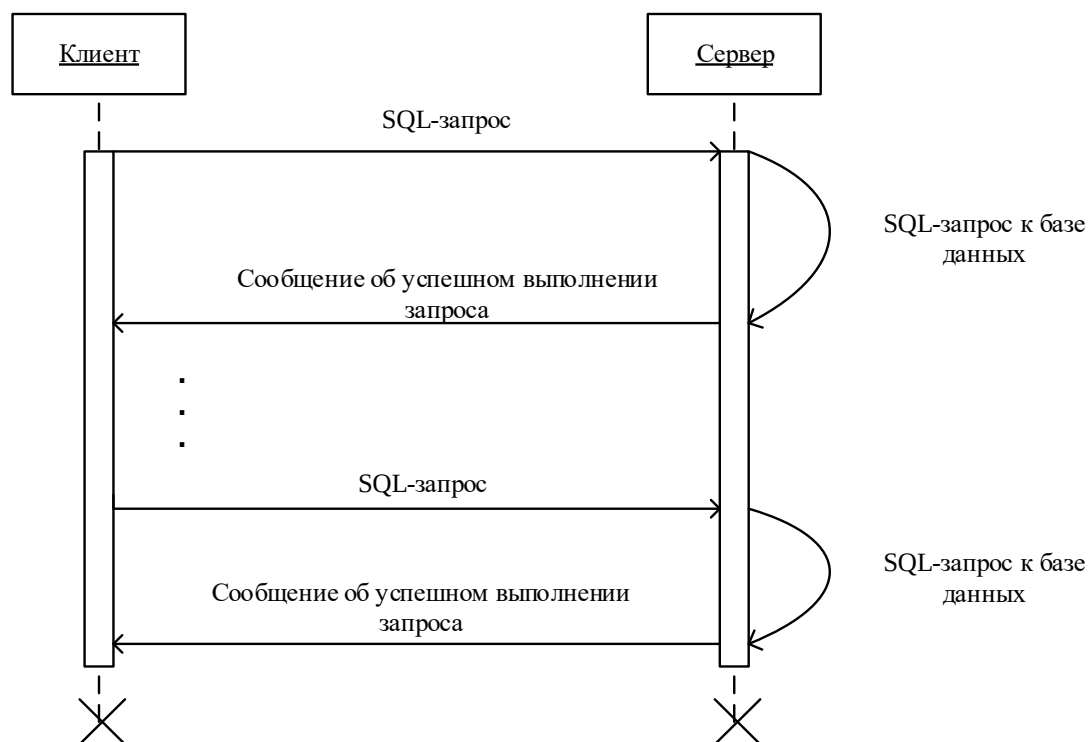


Рисунок 8 – Диаграмма последовательности

2.1 Проектирование клиентского приложения

Интерфейс OLE DB позволяет представить источник данных в виде набора строк и столбцов; кроме того, для выполнения проекта выбраны реляционные СУБД и, соответственно, базы данных, созданные с их помощью. Следовательно, основным элементом интерфейса, отображающим данные, будет таблица. Кроме того, необходимо создать такой элемент, который позволит переключаться между различными таблицами базы. Исходя из этого, рассмотрим, как пользователь сможет взаимодействовать с базой данных. Возможные варианты представлены на UML-диаграмме вариантов использования (рисунок 9).

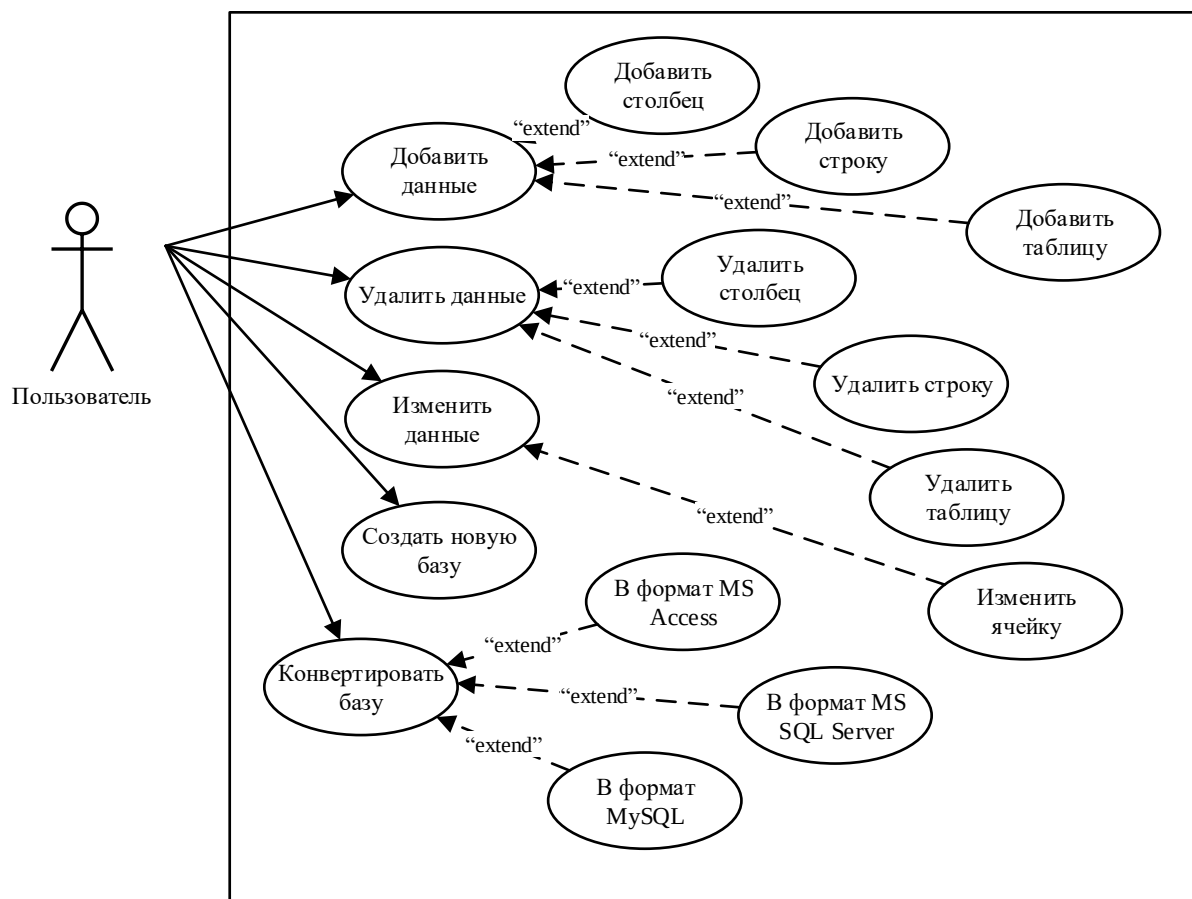


Рисунок 9 – Диаграмма вариантов использования

Для реализации этих возможностей необходимо будет разместить в окне приложения кнопки, меню или предусмотреть иные интуитивно понятные способы осуществления указанных действий. По возможности следует разместить их так, чтобы не отвлекать внимание пользователя от таблицы с данными из базы. Некоторые из вариантов использования следует рассмотреть подробнее.

Добавление нового столбца потребует от пользователя определения как минимум двух параметров: названия столбца и типа данных, которые в нем будут содержаться. Опциональным является указание, может ли значение ячейки в столбце принимать значение NULL, является ли поле ключевым, имеется ли значение, которым ячейка заполняется по умолчанию и т.д.

Различные СУБД предоставляют широкий выбор типов данных, хранящихся в базе. Наиболее часто используемыми среди них являются:

- числовой;

- текстовый;
- логический;
- дата и время.

Такой набор типов данных позволяет создать любую базу, поскольку другие типы можно свести к одному из указанных, например, данные столбца базы Microsoft Access типа «Денежный» легко перенести в столбец числового типа. Таким образом, при добавлении нового столбца необходимо предоставлять пользователю выбор как минимум из данных четырех типов.

Добавление строки и изменение данных в ячейке следует производить только после проверки на соответствие типа введенных данных типу столбца, в который их помещает пользователь. В случае несоответствия типов следует вывести сообщение об ошибке и предоставить пользователю возможность изменить введенные данные и попытаться добавить их снова.

Отдельного рассмотрения требует также пункт «Открыть базу данных». Поскольку предполагается, что комплекс приложений будет работать по сети, клиентскому приложению потребуется сначала подключиться к серверу, чтобы начать работу с базой данных. Исходя из этого, в целях логического отделения основной работы приложения от стадии подключения к серверу, было принято решение о создании вспомогательного окна с полями, в которые пользователь должен внести все необходимые данные для подключения к серверу, а затем – к требуемой базе данных.

Набор полей ввода должен содержать, по меньшей мере, следующие:

- IP-адрес сервера;
- Порт, по которому сервер принимает данные;
- Тип базы данных (Microsoft Access, Microsoft SQL Server или MySQL);
- Название базы, к которой необходимо подключиться.

Возможность хранения вводимых в поля данных и подсказки или автозаполнения на их основе существенно сократит время, которое

пользователь затрачивает на подключение к базе, и значительно повысит эффективность работы. Для достижения такой цели необходимо обеспечить хранение данных в файле, а также своевременное его дополнение новыми данными, если подключение к серверу и открытие базы данных с их использованием произошло успешно. При подключении к серверу и открытии требуемой базы следует закрыть вспомогательное окно и вызвать основное окно приложения, в противном случае – отобразить сообщение об ошибке введенных данных.

Требования, перечисленные выше, касаются внешнего вида приложения-клиента и его взаимодействия с пользователем. Рассмотрим более подробно работу клиента с точки зрения его внутреннего устройства.

Запуск программы должен вызывать окно подключения к серверу, но не должен вызывать основное окно приложения, чтобы пользователь не был вынужден закрывать лишние окна в случае ошибочного запуска приложения или невозможности подключиться к серверу. При нажатии на соответствующую кнопку должна осуществляться попытка подключения к серверу на основе введенных пользователем данных, а затем – попытка открытия целевой базы данных. В случае успешного выполнения указанных действий окно подключения должно быть закрыто и вызвано основное окно приложения. В случае возникновения ошибки на том или ином этапе следует вывести соответствующее сообщение и предоставить пользователю изменить данные в полях ввода и попробовать подключиться к серверу снова. Схема алгоритма работы части клиентского приложения, связанной с окном подключения представлена на рисунке 10.

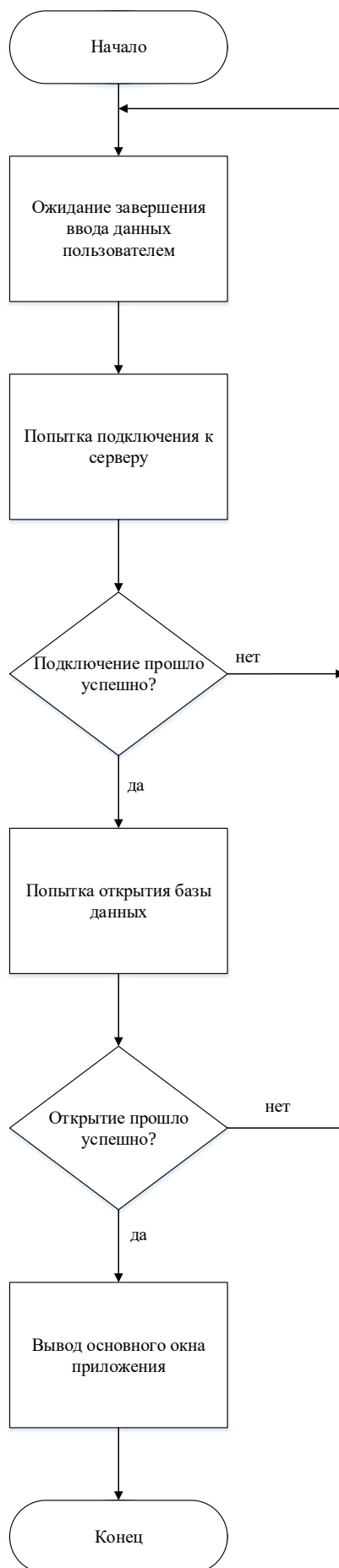


Рисунок 10 – Алгоритм работы части клиентского приложения

Изм.	Лист	№ докум.	Подпись	Дата

После запуска основного окна приложения следует отобразить данные из запрошенной базы данных. Переключение между различными таблицами базы должно быть простым и интуитивно понятным. Заголовки столбцов следует выделить для наглядности и более удобного их восприятия пользователем.

Действия пользователя должны вызывать соответствующие обработчики. Поскольку выбранные СУБД поддерживают возможность взаимодействия с базой данных посредством выполнения запросов на языке SQL, в обработчиках должны конструироваться команды, в которые подставляются названия таблиц и/или столбцов в соответствии с синтаксисом той или иной конкретной команды. Однако следует учитывать, что выбранные системы управления базами данных используют расширения языка SQL, что может послужить причиной некоторых различий в синтаксисе.

Передача сконструированного запроса на сервер должна осуществляться посредством создания сокета с введенными на этапе подключения данными и последующего соединения с сервером. Помимо запроса следует передавать строку подключения к базе данных, поскольку именно с помощью нее осуществляется доступ к конкретной базе данных через интерфейс OLE DB. Такая строка также должна формироваться на этапе подключения к серверу и попытки доступа к базе данных.

Получение ответа от сервера будет признаком того, что выполнение запроса прошло успешно. В противном случае на каком-то этапе произошла ошибка – тогда необходимо вывести пользователю соответствующее сообщение. Если повторный запрос также приведет к возникновению ошибки, то причиной такой ошибки может быть, например, отключение сервера, перенос или удаление базы данных, и тогда пользователю следует обратиться к администратору для решения проблемы.

Следующим шагом после успешного выполнения запроса должно быть обновление интерфейса приложения в соответствии с внесенными

изменениями. Однако, поскольку отрисовка элементов интерфейса занимает относительно длительный промежуток времени и напрямую зависит от количества данных (то есть отображение 1 таблицы с 1 столбцом и 1 строкой займет в несколько раз меньше времени, чем отображение 100 таблиц со 100 столбцами и 100 строками в каждой), необходимо, по возможности сократить объем перерисовываемых элементов. Так, при изменении данных в рамках одной таблицы будем обновлять только ее. Если же пользователь оперирует таблицами, то необходимо обновить всю базу целиком. Если была добавлена новая таблица, то после обновления для удобства навигации целесообразно переместить фокус на нее. Если какая-либо таблица была удалена, то фокус следует переместить на другую: это может быть первая таблица, следующая или предыдущая после удаленной. Частным случаем является удаление последней таблицы в базе. Тогда большая часть основного окна пользовательского приложения будет пустой до тех пор, пока пользователь не добавит новую таблицу в базу.

Схема алгоритма работы клиентского приложения представлена на рисунке 11.

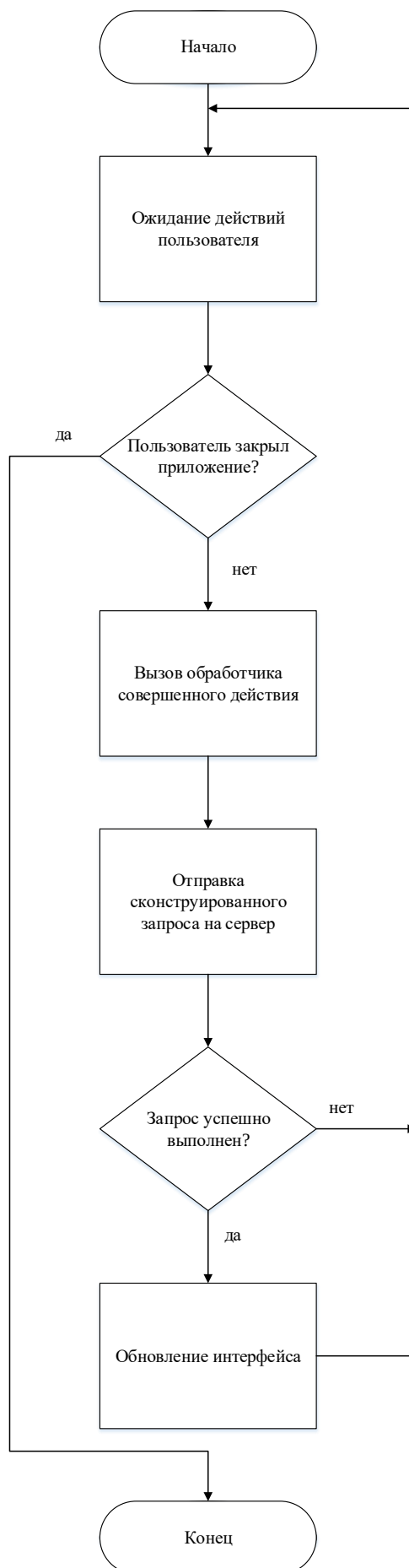


Рисунок 11 – Алгоритм работы клиентского приложения

Тогда принимаемые клиентским приложением состояния и связи между ними можно изобразить с помощью UML-диаграммы состояний (рисунок 12).

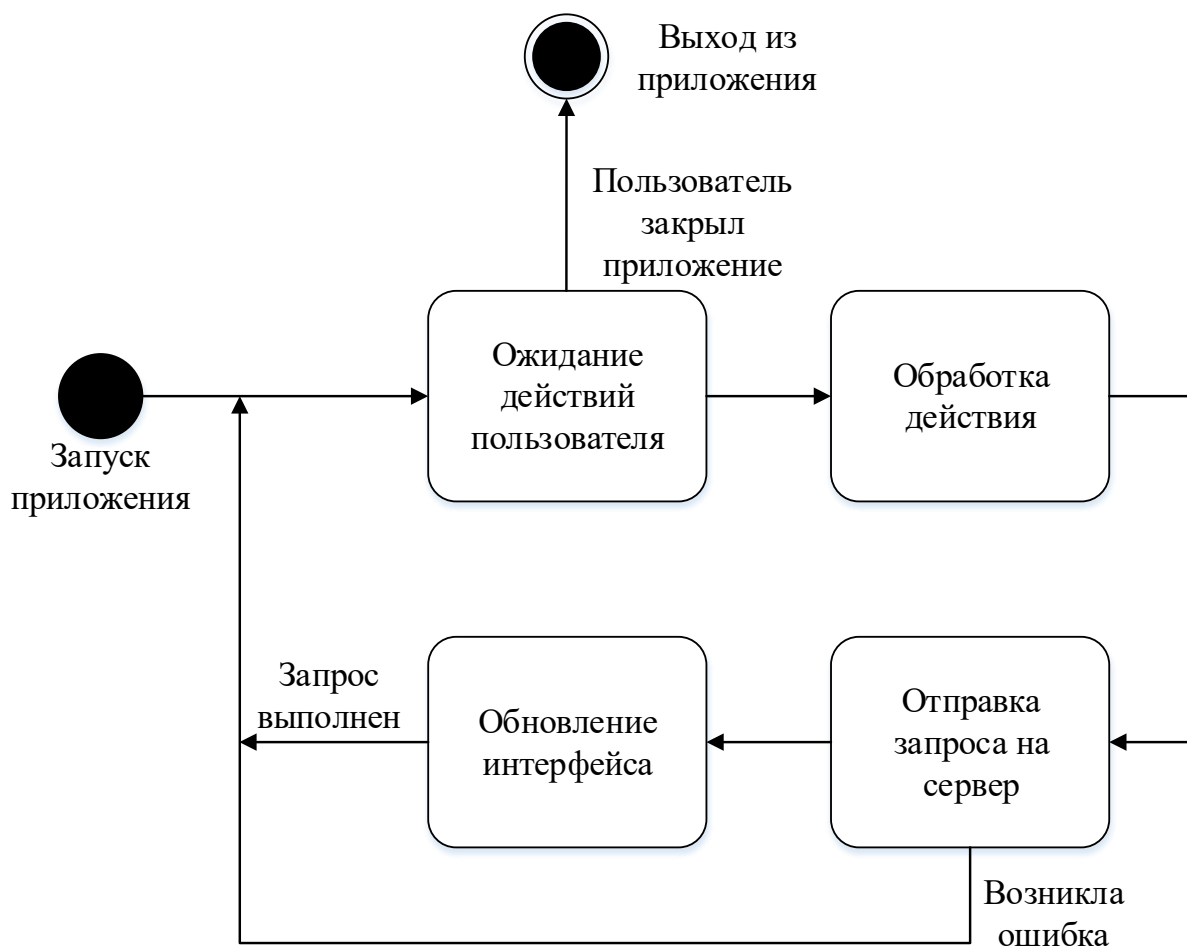


Рисунок 12 – Диаграмма состояний клиентского приложения

2.2 Проектирование серверного приложения

Сервер при запуске создаст сокет с IP-адресом машины, на которой запущено приложение, и портом 8080. Сокет будет переводиться в режим «слушания», т.е. ожидания входного соединения.

Серверное приложение при установлении соединения будет осуществлять прием запроса и строки подключения к базе от клиента по установленному соединению. Затем будет создано подключение к базе данных с помощью указанной строки через интерфейс OLE DB и попытка выполнения запроса. Сообщение об успешном выполнении запроса либо его

результаты будут пересылаться обратно клиентскому приложению. Затем сервер снова перейдет в режим ожидания, пока клиентом не будет инициировано новое соединение.

Связь с клиентом будет организована по протоколу TCP/IP. Поскольку для работы сервера не требуется ввода никаких данных, интерфейс приложения можно сделать консольным, при необходимости можно выводить данные о подключениях, выполняемых запросах или иную вспомогательную информацию.

Схема алгоритма работы серверного приложения представлена на рисунке 13.

					ПГУ 1.090301.16.001 ПЗ	Лист
						31
Изм.	Лист	№ докум.	Подпись	Дата		

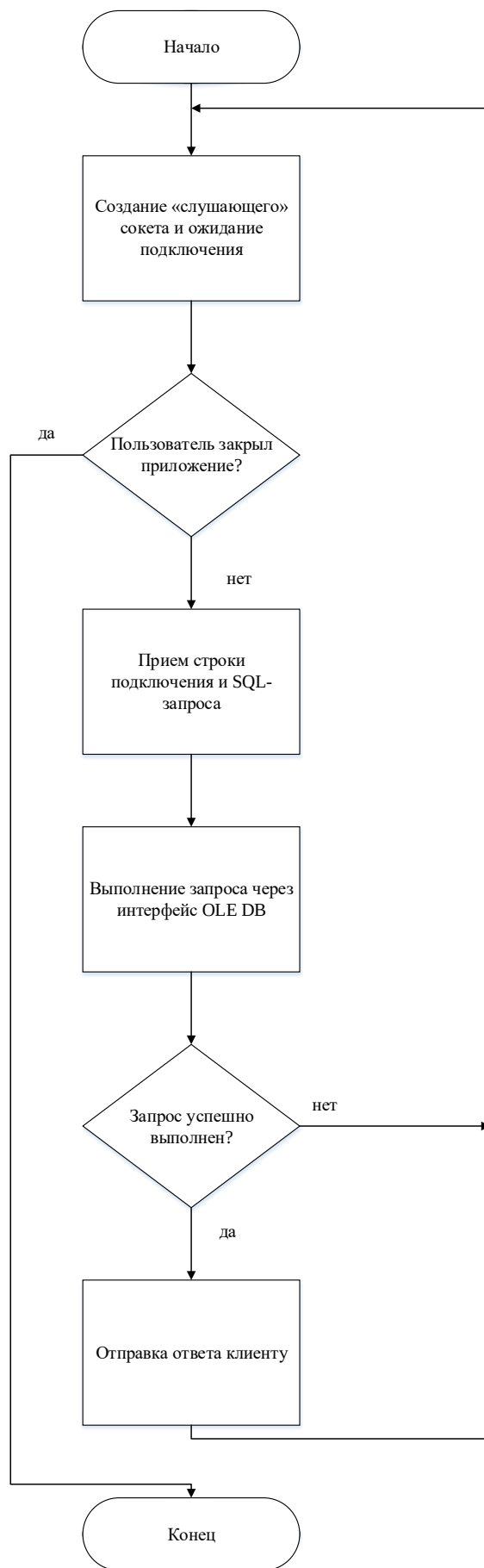


Рисунок 13 – Алгоритм работы серверного приложения

Таким образом, приложение-сервер может принимать следующие состояния (рисунок 14):

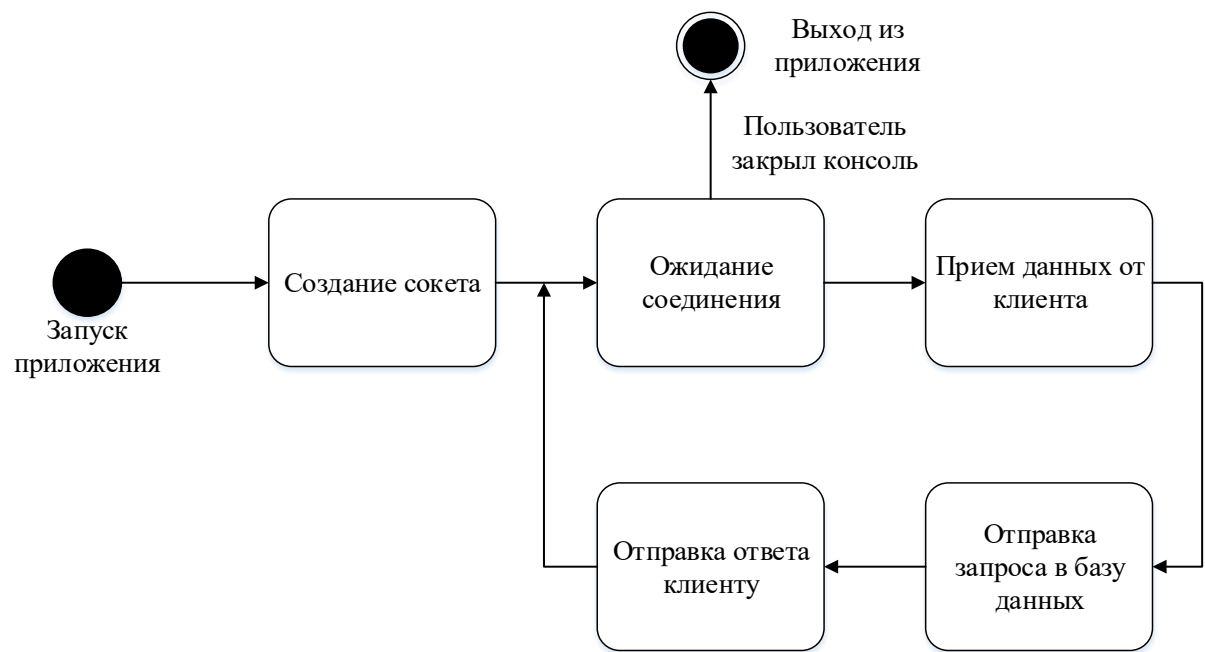


Рисунок 14 – Диаграмма состояний серверного приложения

3 РЕАЛИЗАЦИЯ ПРОЕКТА

Реализация проекта заключается в кодировании полученного на предыдущем этапе прототипа с учетом выделенных особенностей функционирования, в дальнейшей отладке программы и проведении тестов с целью достижения ее корректной работы.

Процесс реализации в рамках данной работы разделен на две части: первая относится к реализации клиентского приложения, вторая – к реализации приложения-сервера. Затем предлагаются результаты нескольких тестов, позволяющие сделать вывод о корректности работы программ и выполнении поставленной в данной работе цели.

3.1 Процесс разработки

Реализация каждого из приложений начиналась с выбора наиболее подходящего интерфейса. Дальнейший процесс разработки проходил с учетом сделанного выбора. Для написания программ использовался язык C# и среда разработки Visual Studio 2010.

Проект клиента был реализован первым с целью создания и отладки всего необходимого функционала. Программа взаимодействовала с базами данных напрямую, без участия сервера. После тестирования и получения результатов, которые подтверждали корректность работы созданного приложения, началась разработка серверного приложения.

Реализация сервера заключалась в создании связи между программами, а также в переносе участков кода, связанных с непосредственным доступом к базе данных, с клиента на сторону сервера.

Рассмотрим описанный выше процесс разработки приложений более подробно.

3.1.1 Разработка клиентского приложения

Реализация клиентского приложения началась с выбора интерфейса, который позволил разместить в основном окне программы все необходимые для отображения данных элементы, а также инструменты для работы с данными. Для этого был выбран интерфейс программирования приложений Windows Forms, предоставляющий широкий круг разнообразных элементов управления, позволяющих создать интуитивно понятный пользовательский интерфейс. Действия пользователей обрабатываются с помощью механизма событий.

В процессе разработки программы применялся смешанный метод, так как это позволяет сразу придать желаемый вид окну приложения и продумать все его особенности, и только затем на их основе запрограммировать обработку тех или иных событий независимо друг от друга.

Разработка программы началась с создания интерфейса основной формы. Для удобства восприятия большую часть формы занимает элемент TabControl, позволяющий создавать несколько вкладок, что необходимо для отображения нескольких таблиц. При открытии базы данных создается количество вкладок, соответствующее количеству таблиц в базе, и на них размещаются элементы DataGridView, содержимое которых заполняется соответствующими данными. Заголовки столбцов в таблицах выделяются цветом для их визуального отделения от содержимого столбца. Для осуществления тех или иных действий над базой данных в верхнюю часть окна помещен элемент MenuStrip, содержащий несколько логически сгруппированных выпадающих меню.

Также были созданы и отлажены обработчики нажатий на кнопки: в заранее сконструированную строку SQL-запроса подставляются необходимые данные, например, название таблицы, поля и т.д.

На заключительном этапе разработки было добавлено окно подключения, содержащее поля ввода данных, необходимых для соединения с сервером и открытия требуемой базы данных.

Разработанное приложение работает по алгоритму, представленному на UML-диаграмме активности (рисунок 15).

					ПГУ 1.090301.16.001 ПЗ	Лист
						36
Изм.	Лист	№ докум.	Подпись	Дата		



Рисунок 15 – Диаграмма активности клиентского приложения

Этап «Обработка действий пользователя» разворачивается в следующую диаграмму (рисунок 16):

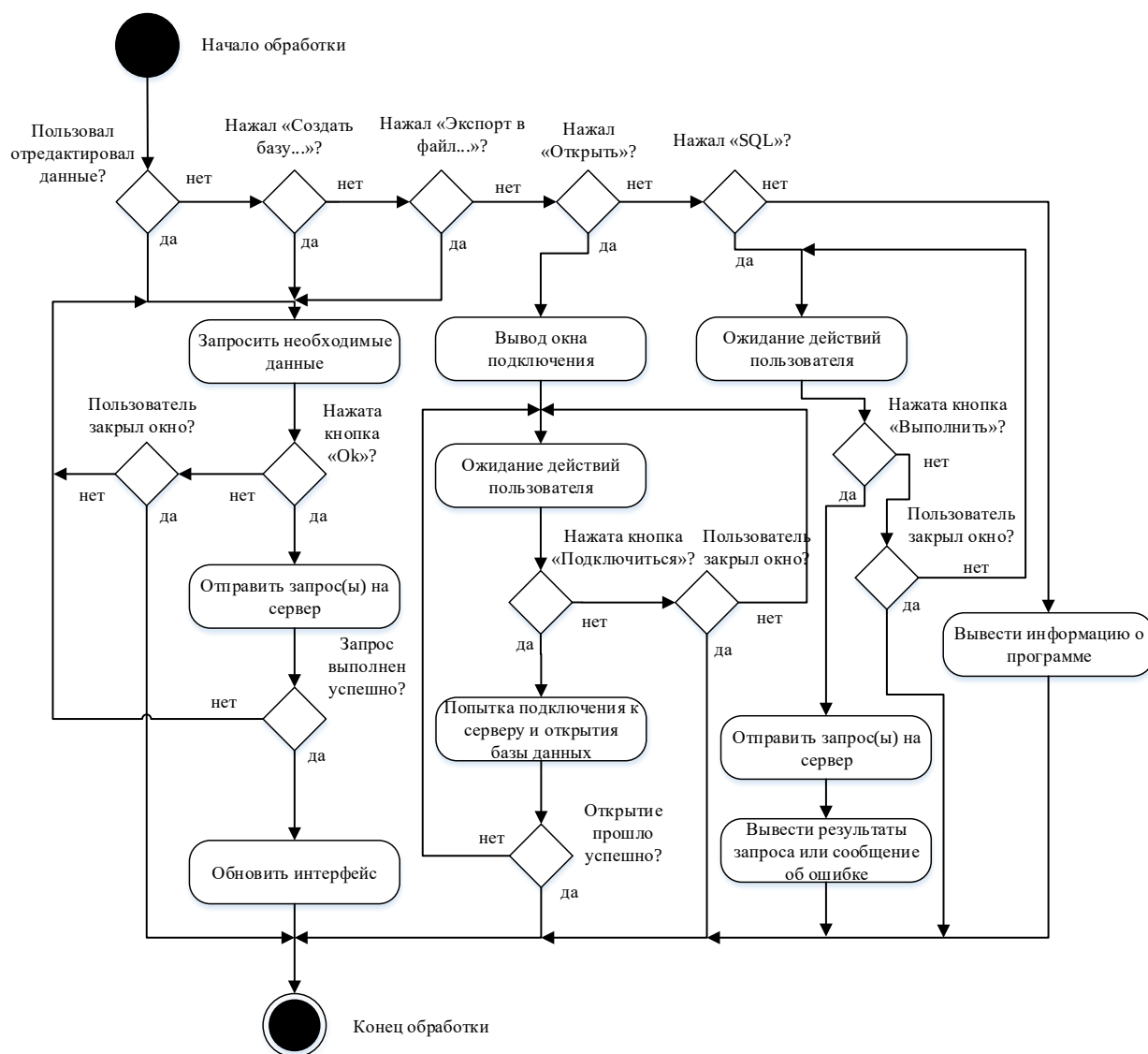


Рисунок 16 – Часть диаграммы активности клиентского приложения

3.1.2 Разработка серверного приложения

Приложение-сервер не требует ввода и вывода данных, для его реализации была выбрана консоль. Основная работа сервера заключается в приеме SQL-запроса со стороны клиента и отправке его в базу данных.

Подключение пространства имен System.Data.OleDb позволило использовать в программе объекты таких классов, как OleDbConnection, OleDbCommand, OleDbDataReader, что помогает осуществлять доступ к базе

данных с помощью интерфейса OLE DB, который был выбран для реализации данного проекта.

Подключение пространств имен System.Net и System.Net.Sockets позволило использовать классы, с помощью объектов которых можно организовать связь между клиентом и сервером по протоколу TCP/IP.

Суть работы серверного приложения сводится к следующему: создается сокет с IP-адресом машины, на которой запускается приложение, и портом 8080. Он переводится в режим ожидания входящих соединений. При возникновении соединения сервер принимает данные, затем вызывается функция их обработки.

Разработанное приложение-сервер работает по алгоритму, созданному на этапе проектирования и изображенному на рисунке 13.

В общем виде созданные на этапе разработки компоненты проекта и связи между ними можно представить на UML-диаграмме развертывания (рисунок 17).

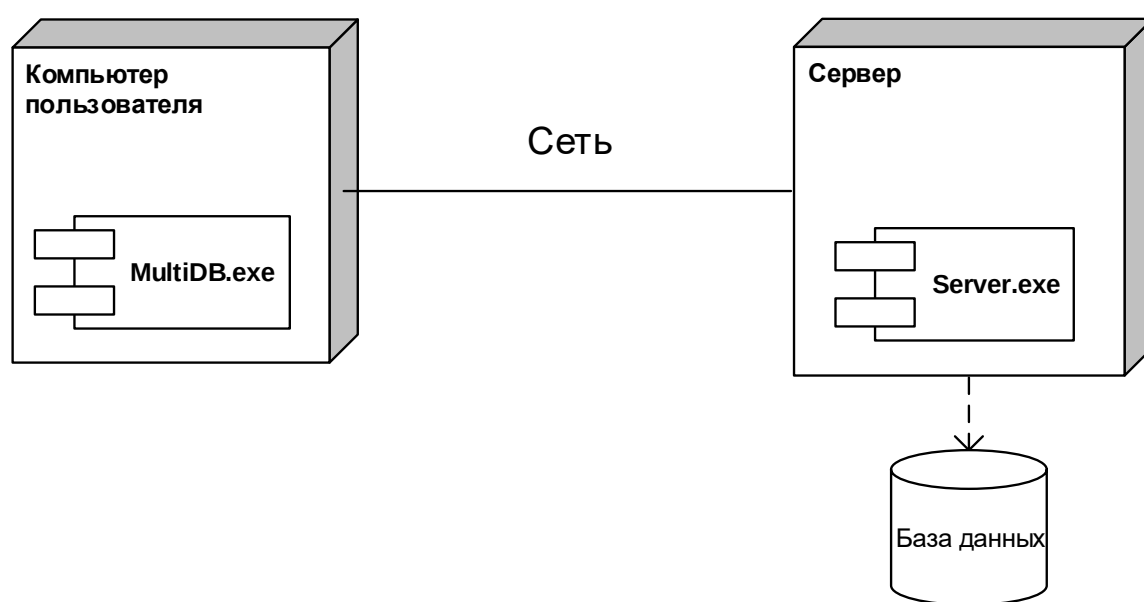


Рисунок 17 – Диаграмма развертывания

3.2 Отладка и тестирование

Программы написаны и отлажены с использованием среды программирования Visual Studio 2010. Для тестирования и оценки корректности работы приложения использовались такие системы управления базами данных, как Microsoft Access 2013, Microsoft SQL Server Management Studio 2017 и MySQL Workbench 6.3 CE.

Отладка проводилась с помощью таких инструментов, как точка останова, выполнение кода по шагам, анализ содержимого переменных.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с синтаксисом конструируемых в программе SQL-запросов, взаимодействием клиентского и серверного приложений (в частности, пересылкой данных и корректным их приемом), а также непосредственным доступом к базе данных. В качестве наглядного доказательства корректной работы комплекса программ и достижения основной цели данного проекта приводятся следующие тесты.

3.2.1 Тестирование возможности добавления новой таблицы в базу данных

Произведем добавление новой таблицы в базу данных Microsoft Access. Исходная база содержит 3 таблицы: Author, Book и Publisher (рисунок 18). Каждая из них содержит соответствующие данные о книгах в магазине.

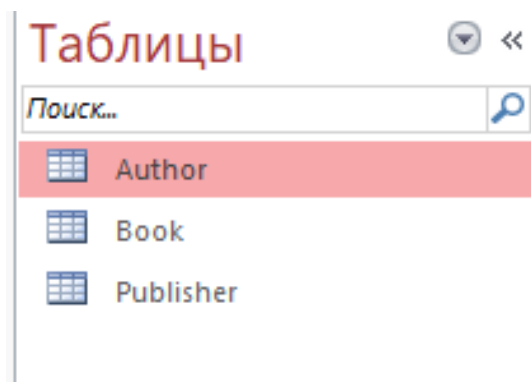


Рисунок 18 – Таблицы исходной базы данных

Откроем эту базу данных в разработанном приложении (рисунок 19).

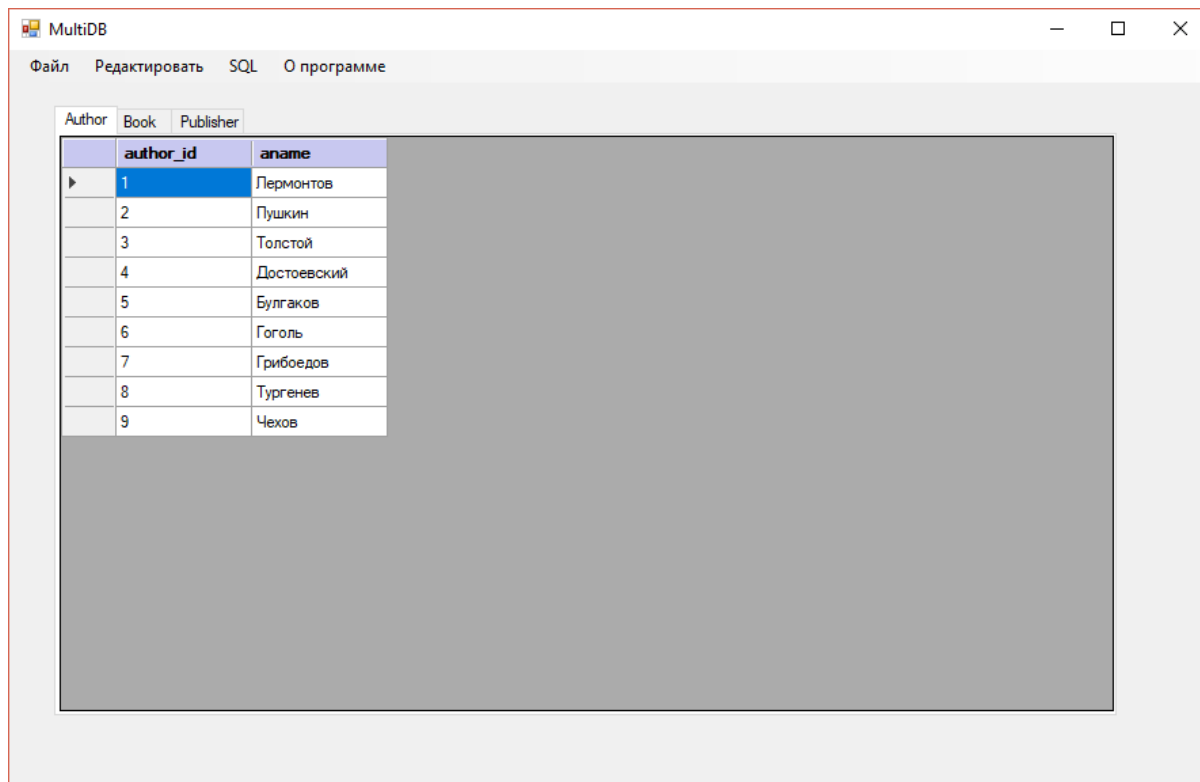


Рисунок 19 – Исходная база данных открыта в созданном приложении

Для добавления новой записи кликнем по кнопке «Редактировать» в верхней части окна и выберем пункт «Добавить таблицу». Откроется окно с полем, в которое необходимо ввести название добавляемой таблицы (рисунок 20).

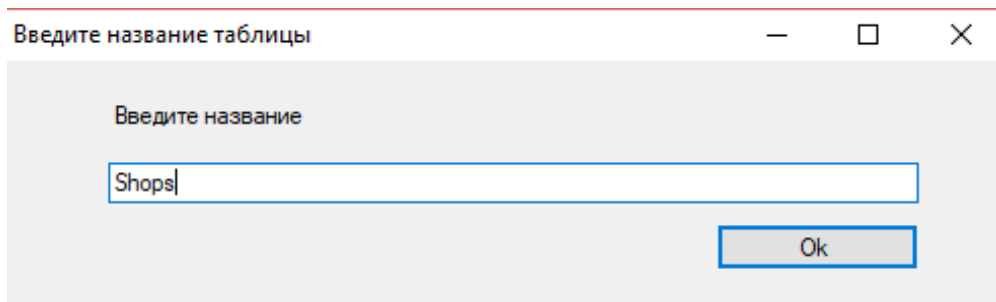


Рисунок 20 – Добавление таблицы "Shops"

Кликнем по кнопке «Ok». Окно приложения обновится, в нем появится вкладка с названием добавленной базы (рисунок 21).

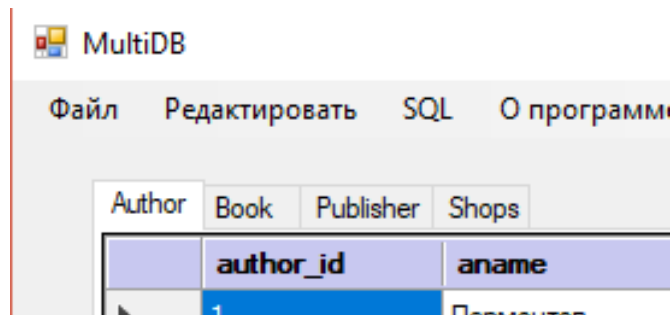


Рисунок 21 – Вкладки с названиями таблиц

Откроем базу данных в СУБД Microsoft Access и убедимся, что таблица действительно добавилась (рисунок 22).

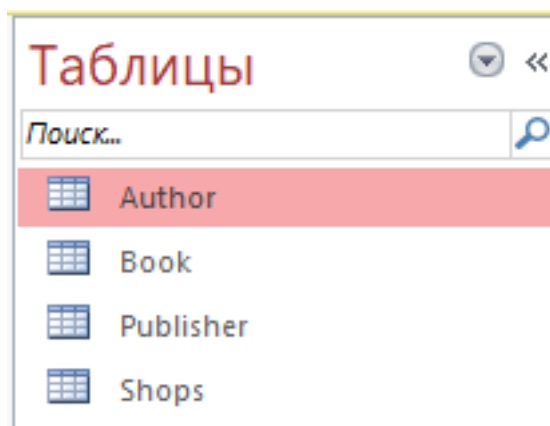


Рисунок 22 – База данных с добавленной таблицей открыта в MS Access

Таким образом, можно сделать вывод о том, что функция добавления новой таблицы в базу работает корректно.

3.2.2 Тестирование возможности изменения данных в ячейке

Используя ту же базу данных, что и в предыдущем тестировании, а именно базу с данными о книгах в магазине, произведем редактирование какой-либо ячейки. Исходная таблица «Authors» выглядит следующим образом (рисунок 23).

Author		
	author_id	aname
+	1	Лермонтов
+	2	Пушкин
+	3	Толстой
+	4	Достоевский
+	5	Булгаков
+	6	Гоголь
+	7	Грибоедов
+	8	Тургенев
+	9	Чехов

Рисунок 23 – Исходная таблица "Authors"

Для изменения ячейки в разработанной программе достаточно щелкнуть по ней для выбора, а затем щелкнуть еще раз для редактирования значений. Изменим автора с author_id = 7, поменяв «Грибоедов» на «Куприн» (рисунок 24), и нажмем клавишу Enter для завершения редактирования (значок в левой части строки показывает, что в одной из ячеек идет редактирование).

...	7	Куприн
0		Тургенев

Рисунок 24 – Изменение данных в ячейке

Откроем базу данных в СУБД Microsoft Access и убедимся, что значение ячейки действительно изменилось (рисунок 25).

Author		
	author_id	aname
+	1	Лермонтов
+	2	Пушкин
+	3	Толстой
+	4	Достоевский
+	5	Булгаков
+	6	Гоголь
+	7	Куприн
+	8	Тургенев
+	9	Чехов

Рисунок 25 – Таблица "Authors" в MS Access

Таким образом, можно сделать вывод о том, что функция изменения значения ячейки работает корректно.

3.2.3 Тестирование возможности переноса данных в другую базу

Произведем перенос базы данных Microsoft Access в формат Microsoft SQL Server. Исходная база содержит 3 таблицы: Author, Book и Publisher, их содержимое приведено на рисунках 26, 27 и 28 соответственно.

Author		
	author_id	aname
+	1	Лермонтов
+	2	Пушкин
+	3	Толстой
+	4	Достоевский
+	5	Булгаков
+	6	Гоголь
+	7	Грибоедов
+	8	Тургенев
+	9	Чехов

Рисунок 26 – Исходная таблица "Authors"

Book				
	book_id	book_name	author_id	pub_id
	1	Герой нашего	1	1
	2	Капитанская д	2	1
	3	Евгений Онег	2	3
	5	Война и мир	3	2
	13	Воскресение	3	2
	6	Анна Каренин	3	3
	4	Преступление	4	2
	8	Собачье серд	5	3
	12	Шинель	6	3
	10	Горе от ума	7	2
	11	Отцы и дети	8	1
	15	Вишневый сад	9	3

Рисунок 27 – Исходная таблица "Book"

Publisher			
	pub_id	pub_name	
+	1	МоскваПресс	
+	2	СпбПресс	
+	3	Мнемозина	
*			

Рисунок 28 – Исходная таблица "Publisher"

Схема связей между таблицами отображена на рисунке 29.

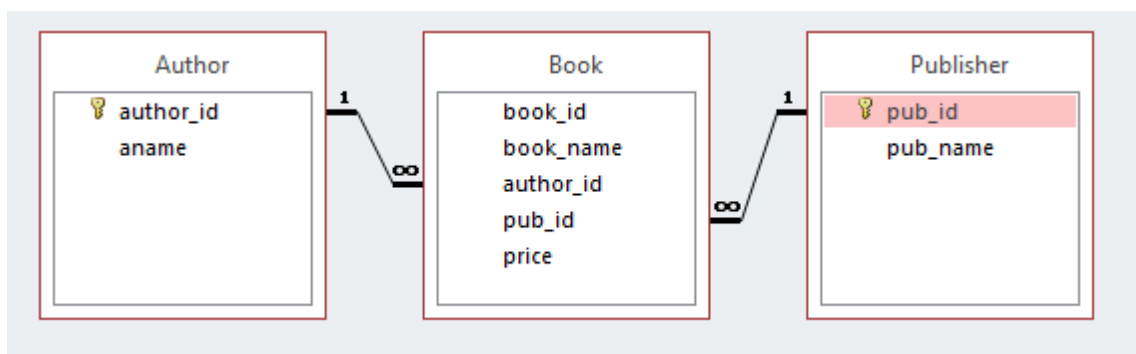


Рисунок 29 – Связи между таблицами

Заметим, что до выполнения переноса с помощью разработанного приложения такой базы нет на целевом сервере (рисунок 30).

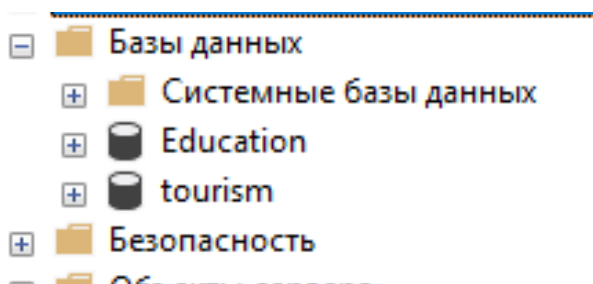


Рисунок 30 – Базы данных на целевом сервере

Откроем эту базу данных в разработанном приложении (рисунок 31).

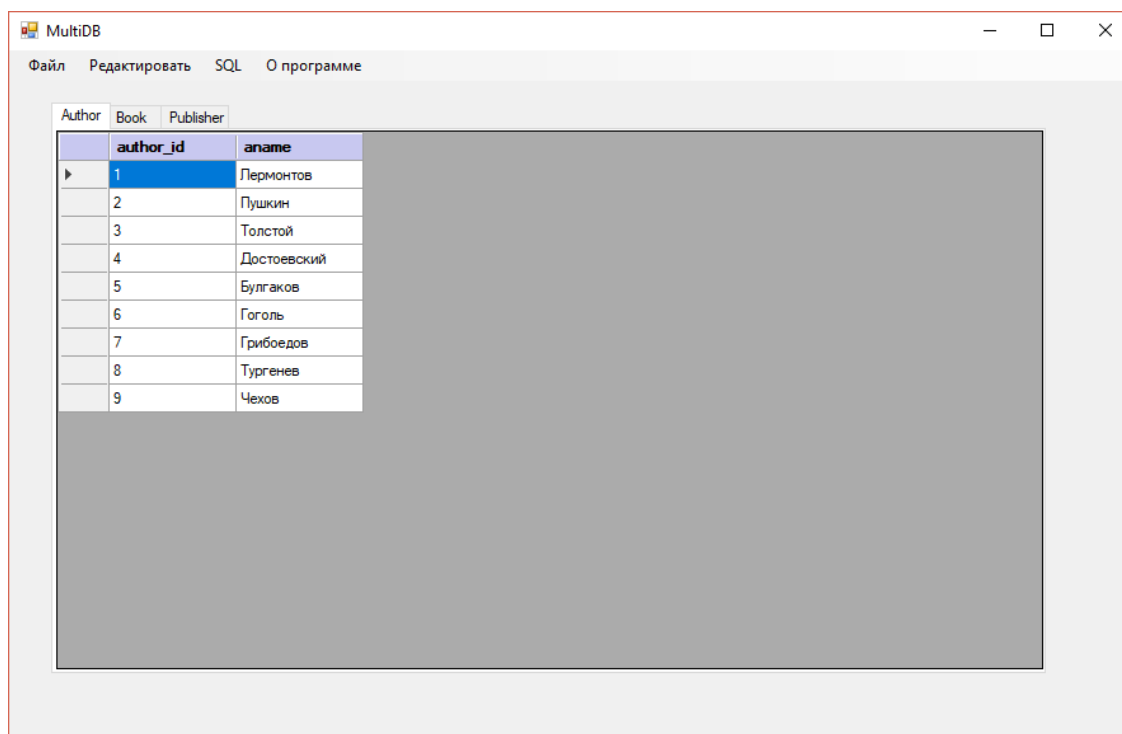


Рисунок 31 – База данных открыта в разработанном приложении

Для переноса базы в формат Microsoft SQL Server кликнем по кнопке «Файл» в верхней части окна, в раскрывшемся меню выберем пункт «Экспорт в файл...» и из предоставленных вариантов кликнем по «MS SQL Server» (рисунок 32).

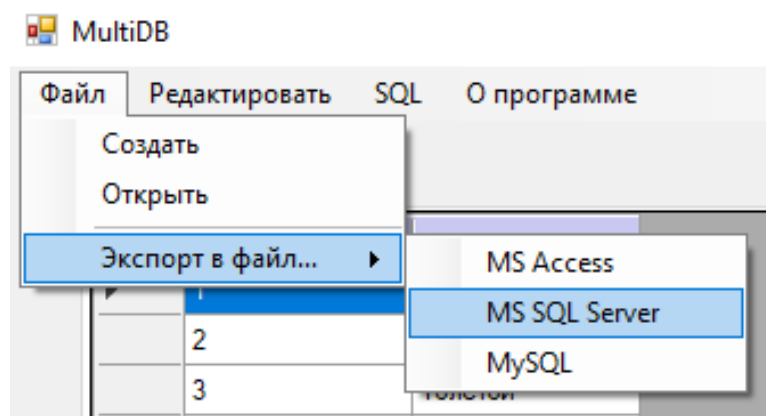


Рисунок 32 – Экспорт в файл MS SQL Server

В появившихся диалоговых окнах введем название новой базы – «bookstore» и название сервера, на который происходит перенос, в данном случае – «\SQLEXPRESS». Затем откроем Microsoft SQL Server Management Studio, чтобы убедиться в корректности выполненного переноса. В списке баз данных появилась база с именем bookstore (рисунок 33).

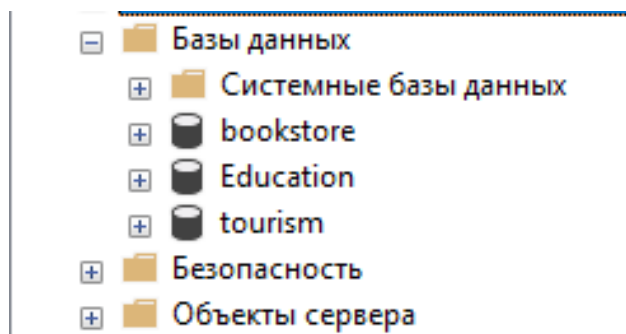


Рисунок 33 – Базы данных на сервере после экспорта

Убедимся, что содержимое базы перенесено корректно. Данные перенесенных таблиц представлены на рисунках 34, 35 и 36.

	author_id	aname
1	1	Лермонтов
2	2	Пушкин
3	3	Толстой
4	4	Достоевский
5	5	Булгаков
6	6	Гоголь
7	7	Грибоедов
8	8	Тургенев
9	9	Чехов

Рисунок 34 – Таблица "Authors"

	book_id	book_name	author_id	pub_id	price
1	1	Герой нашего времени	1	1	1000
2	2	Капитанская дочка	2	1	600
3	3	Евгений Онегин	2	3	700
4	4	Преступление и наказание	4	2	450
5	5	Война и мир	3	2	1500
6	8	Собачье сердце	5	3	550
7	10	Горе от ума	7	2	400
8	11	Отцы и дети	8	1	640
9	12	Шинель	6	3	500
10	13	Воскресение	3	2	310
11	15	Вишневый сад	9	3	260
12	6	Анна Каренина	3	3	490

Рисунок 35 – Таблица "Book"

	pub_id	pub_name
1	1	МоскваПресс
2	2	СпбПресс
3	3	Мнемозина

Рисунок 36 – Таблица "Publisher"

Верно перенесены и связи между таблицами (рисунок 37).

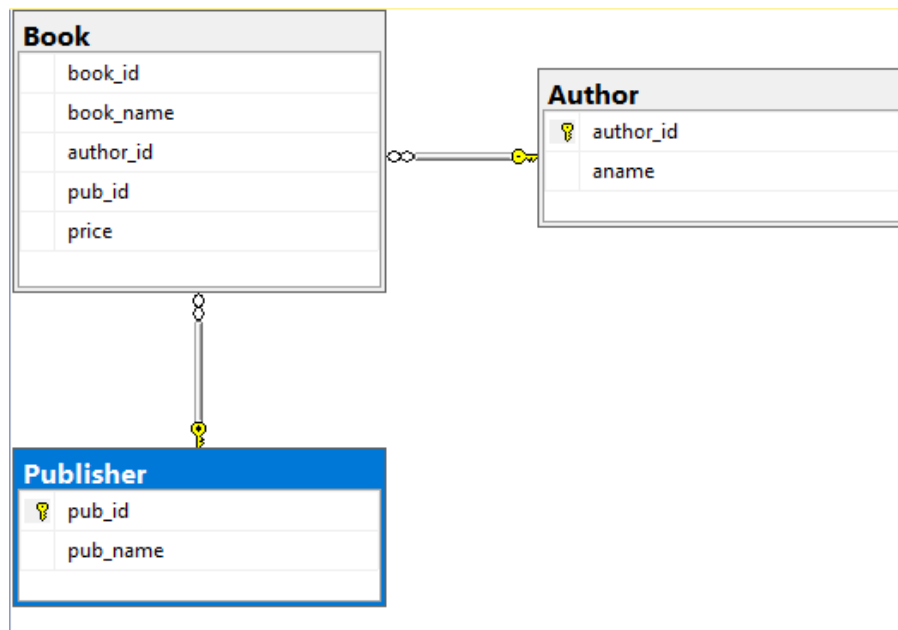


Рисунок 37 – Связи между таблицами

Таким образом, можно сделать вывод о том, что функция переноса данных между базами данных работает корректно, основная цель проекта достигнута.

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Работа комплекса приложений начинается с запуска приложения-сервера. Оно имеет консольный вид и не требует ввода каких-либо данных. Вид запущенного приложения представлен на рисунке 38.

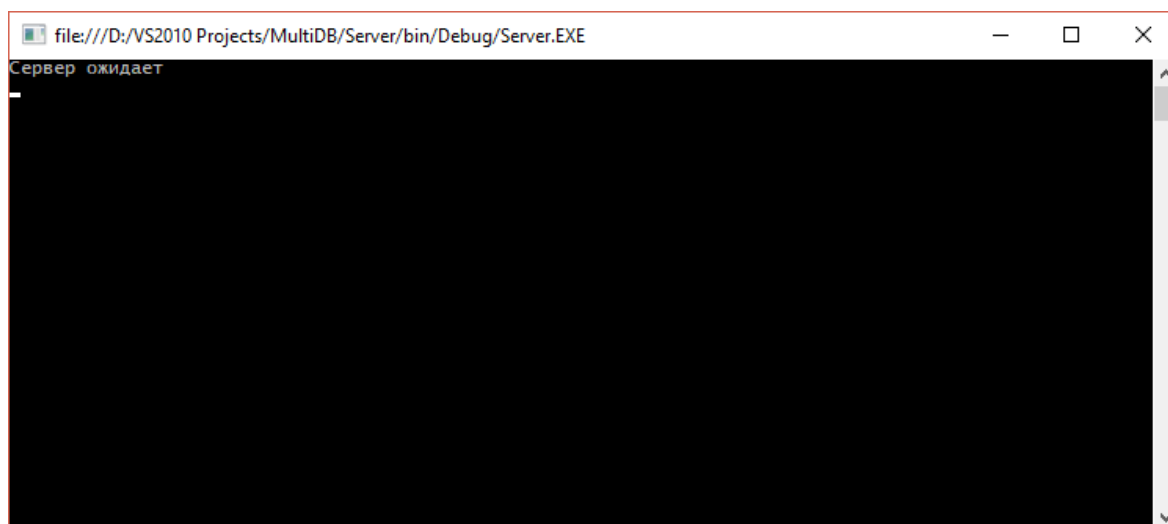


Рисунок 38 – Приложение-сервер

После запуска серверного приложения можно запускать приложение-клиент. Работа приложения начинается с вывода окна с полями, в которые необходимо ввести данные для подключения к серверному приложению и к базе данных. Стоит заметить, что не все поля обязательны для заполнения, поскольку подключение к разным базам данных требует различных параметров подключения. При вводе данных под строкой предлагаются варианты заполнения на основании предыдущих успешных подключений с такими данными (рисунок 39).

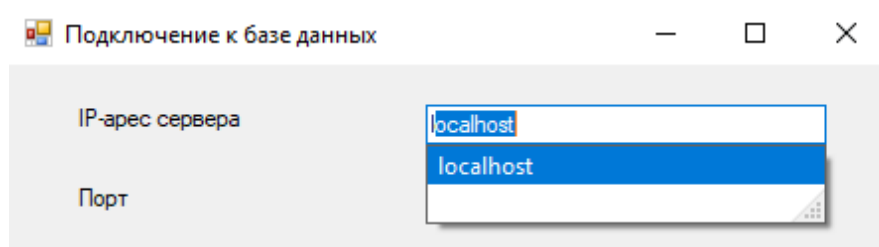


Рисунок 39 – Варианты заполнения

Вид окна с примером заполнения полей представлен на рисунке 40.

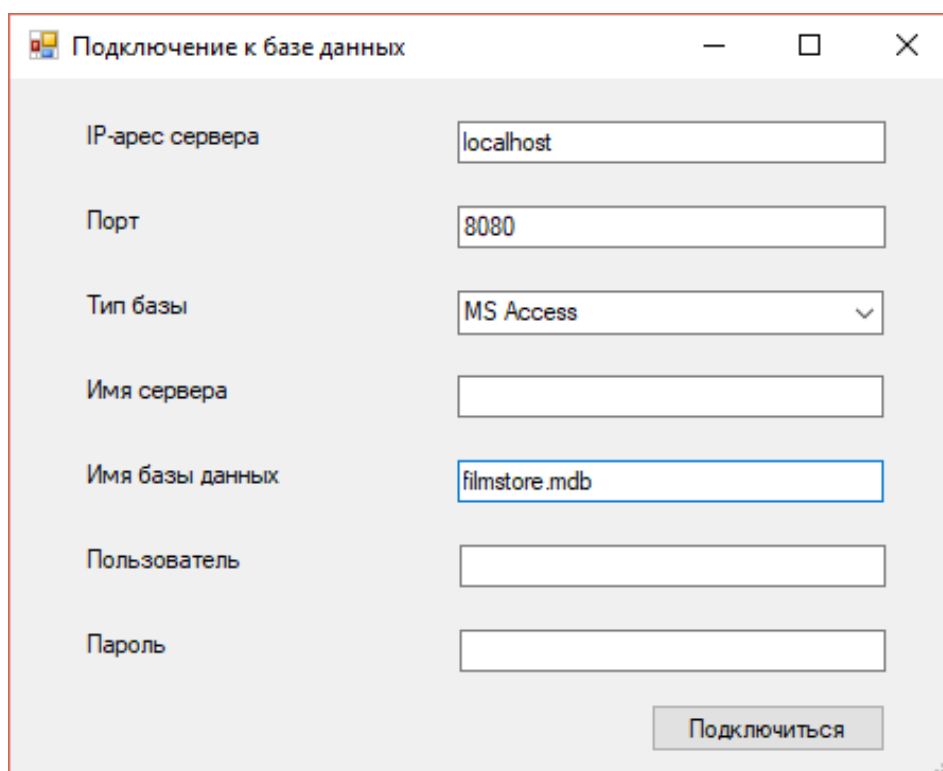


Рисунок 40 – Окно подключения к базе данных

По окончании ввода необходимо нажать кнопку «Подключиться». Окно подключения закроется, откроется основное окно приложения, в котором будут отображены данные из базы. Вид основного окна приложения после подключения представлен на рисунке 41.

film_id	film_name	director_id	country_id	time
1	Чарли и шоколадная фабрика (2...	4	1	115
2	Кошмар перед Рождеством (1993)	4	1	76
3	Унесенные призраками (2001)	2	3	125
4	Навсикая из долины ветров (19	2	3	112
5	Властелин колец: Братство кол...	3	1	178
6	Хоббит: Битва пяти воинств (2014)	3	1	144
7	Валериан и город тысячи плане...	1	2	137
8	Леон (1994)	1	2	133
9	Властелин колец: Две крепости...	3	1	179
10	Властелин колец: Возвращение ...	3	1	201
11	Хоббит: Нежданное путешествие...	3	1	169
12	Хоббит: Пустошь Смауга (2013)	3	1	161
13	Никита (1990)	1	2	117
14	Люси (2014)	1	2	89
15	Пятый элемент (1997)	1	2	126
17	Принцесса Мононоке (1997)	2	3	134
18	Небесный замок Лапута (1986)	2	3	125
19	Ходячий замок (2004)	2	3	119

Рисунок 41 – Основное окно приложения

Основное пространство окна занимает одна из таблиц базы. Переключиться на другую таблицу можно с помощью вкладок, расположенных над таблицей. Наименования вкладок соответствуют названиям таблиц.

Рассмотрим основные возможности, которые предоставляет приложение. В верхней части окна расположены несколько меню. При нажатии на кнопку «Файл», раскрывается меню следующего вида (рисунок 42):

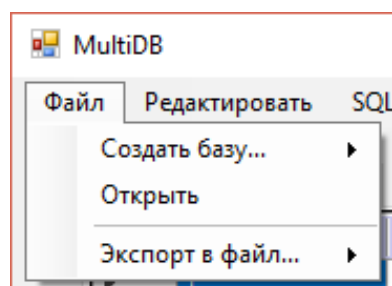


Рисунок 42 – Меню "Файл"

Как видно из рисунка, пункты «Создать базу...» и «Экспорт в файл...» имеют подменю. Их вид представлен на рисунках 43 и 44 соответственно.

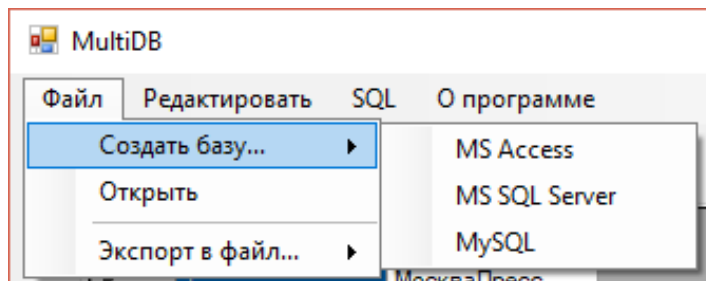


Рисунок 43 – Подменю пункта "Создать базу..."

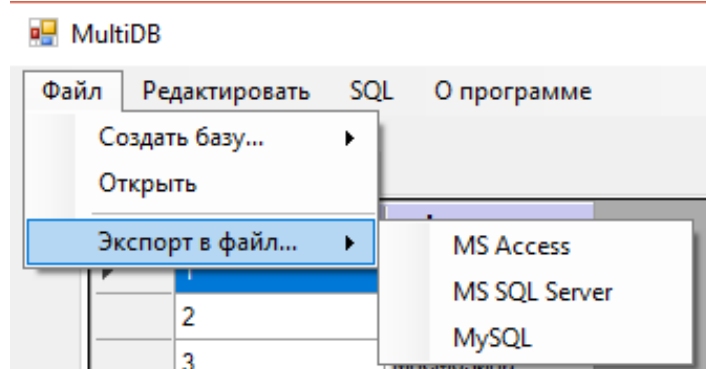


Рисунок 44 – Подменю пункта "Экспорт в файл..."

Действия, выполняемые при выборе какого-либо из пунктов меню «Создать базу» заключаются в выводе окон, запрашивающих у пользователя название новой базы (рисунок 45), а если исходной базой является база данных Microsoft Access и ее требуется конвертировать в базу другого типа, то необходимо также ввести данные сервера (рисунок 46). После ввода всей необходимой информации осуществляется создание пустой базы данных, имеющей указанное пользователем имя.

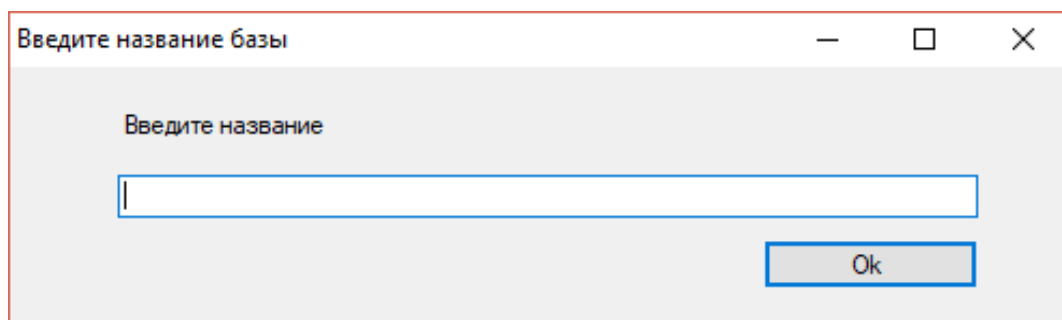


Рисунок 45 – Запрос на ввод имени новой базы

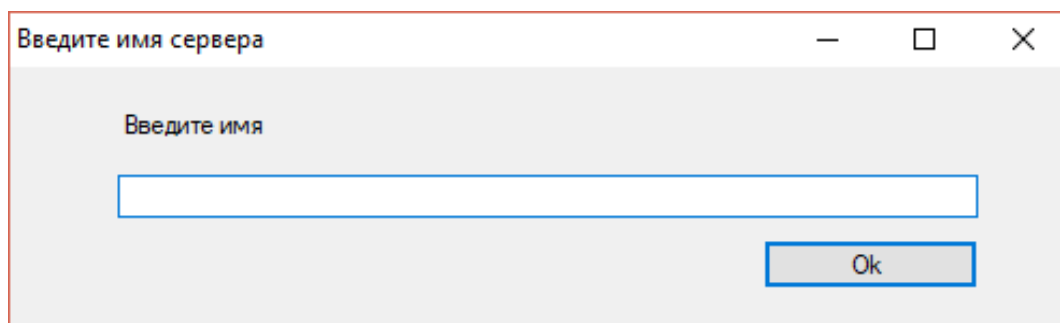


Рисунок 46 – Запрос на ввод имени целевого сервера

Нажатие на кнопку «Открыть» вызовет окно подключения, которое выводилось при запуске клиентского приложения. После ввода всех необходимых данных будет произведена попытка подключения к серверу и открытия целевой базы данных.

Отдельного внимания заслуживает пункт «Экспорт в файл...», поскольку конвертация баз данных в другие форматы является одной из основных задач, решаемых в данном проекте. Как видно из выпадающего меню на рисунке 9, можно конвертировать базу данных в формат СУБД Microsoft Access, Microsoft SQL Server или же MySQL. Для этого необходимо будет ввести название новой базы данных в открывшемся окне (рисунок 45), нажать «Ok». Если исходной базой является база данных Microsoft Access и ее требуется конвертировать в базу другого типа, то необходимо также ввести данные сервера (рисунок 46), на который следует ее загрузить и снова нажать «Ok».

При нажатии на кнопку меню «Редактировать» откроется меню следующего вида (рисунок 47):

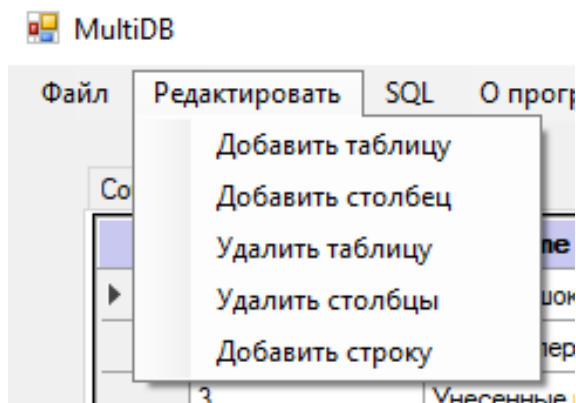


Рисунок 47 – Меню "Редактировать"

Пункты этого меню позволяют производить редактирование базы данных тем или иным образом. Рассмотрим их работу более подробно.

Нажатие на кнопку «Добавить таблицу» вызывает окно следующего вида (рисунок 48):

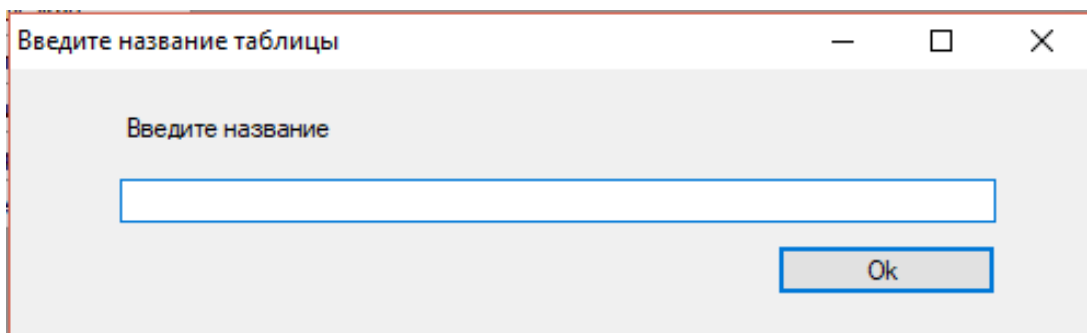


Рисунок 48 – Запрос на ввод названия новой таблицы

После ввода названия и нажатия кнопки «Ok» окно приложения будет обновлено и добавится еще одна вкладка с таблицей, имеющей название, которое указал пользователь.

Нажатие на кнопку «Добавить столбец» вызывает окно следующего вида (рисунок 49):

Рисунок 49 – Окно ввода данных для создания нового столбца

В первое поле требуется ввести название столбца. По нажатию на стрелку сбоку следующего поля выпадает список доступных типов, один из которых необходимо выбрать для создания столбца (рисунок 50).

Рисунок 50 – Типы данных

И, наконец, третье поле используется для задания значения по умолчанию и не является обязательным к заполнению. В случае несовпадения типа столбца и значения по умолчанию будет выведено сообщение об ошибке ввода (рисунок 51)

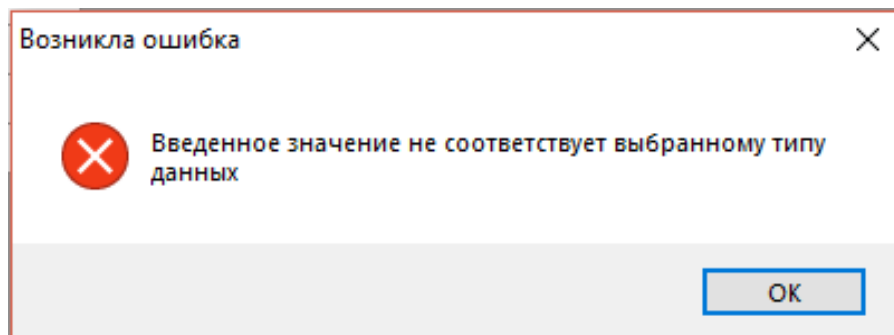


Рисунок 51 – Сообщение об ошибке

Нажатие на кнопку «Удалить таблицу» производит удаление открытой в данный момент таблицы. Следует помнить, что отменить данное действие будет нельзя.

Нажатие на кнопку «Удалить столбцы» вызывает окно следующего вида (рисунок 52):

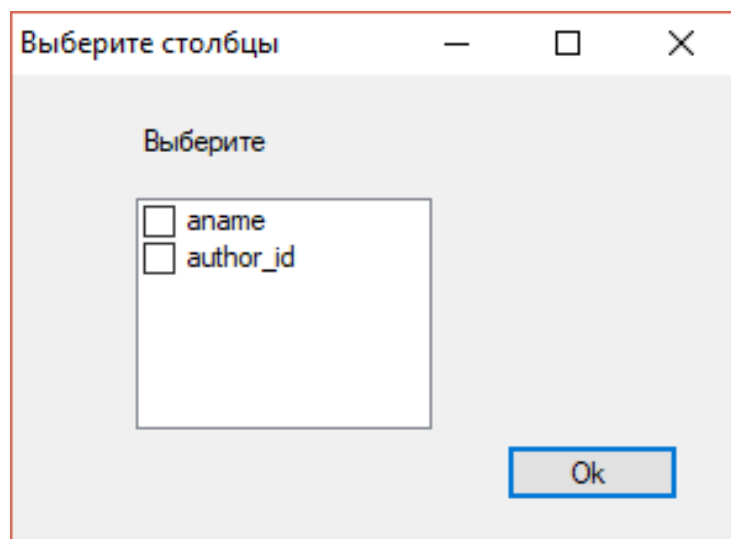


Рисунок 52 – Окно выбора столбцов для удаления

Для выбора столбца необходимо кликнуть по квадрату около его названия, в квадрате появится галочка (рисунок 53).

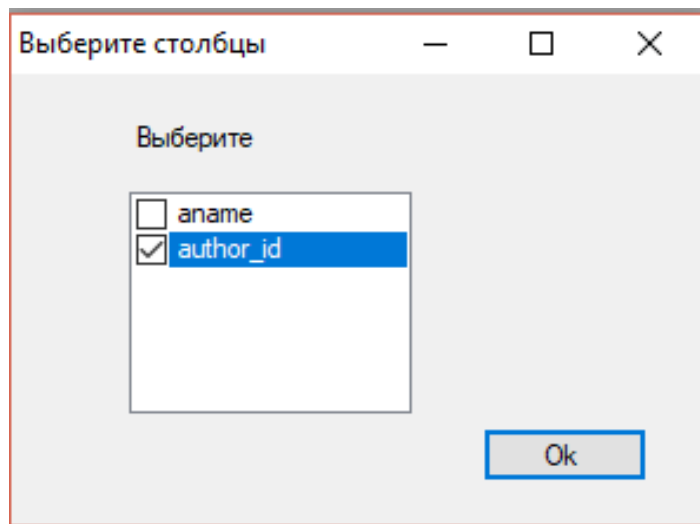


Рисунок 53 – Выбор столбца

После нажатия на кнопку «Ok» все выделенные таким образом столбцы будут удалены. Однако следует помнить, что это может повлечь за собой каскадное удаление записей из таблиц, связанных с текущей, если поле было ключевым. Также нельзя выделить все столбцы, это расценивается программой как ошибка, поскольку удаление всех столбцов означает удаление таблицы, а для этого предусмотрена отдельная кнопка.

Нажатие на кнопку «Добавить строку» вызывает окно приблизительно следующего вида (рисунок 54):

Рисунок 54 – Окно ввода данных для добавления строки

В каждом отдельном случае это окно будет содержать свой собственный набор полей ввода, их список получается из полей текущей таблицы. Таким образом, при выборе другой таблицы окно принимает вид (рисунок 55):

Рисунок 55 – Окно ввода данных для добавления строки

Как видно из рисунка, если таблица содержит множество столбцов, то в правой части окна появляется полоса прокрутки, позволяющая перемещаться между полями ввода. Внизу окна располагается кнопка «Добавить», ее следует нажать после ввода всех необходимых данных. Стоит отметить, что если столбец имеет значение по умолчанию, а при добавлении в соответствующее поле ничего не было введено, то в добавленной строке в ячейке этого столбца будет значение по умолчанию.

Кроме того, можно производить редактирование базы непосредственно в таблице. Например, для удаления строки следует нажать на треугольник слева от нее (рисунок 56) или же выделить несколько строк, кликая по ним и одновременно держа нажатой клавишу Ctrl на клавиатуре (рисунок 57), а затем нажать клавишу Delete на клавиатуре.

6	Хоббит: Битва пяти воинств (2014)	3	1	144
7	Валериан и город тысячи плане...	1	2	137
▶ 8	Леон (1994)	1	2	133
9	Властелин колец: Две крепости...	3	1	179
10	Властелин колец: Возвращение ...	3	1	201

Рисунок 56 – Выбор строки

2	Капитанская до	2	1	600
3	Евгений Онегин	2	3	700
4	Преступление и...	4	2	450
5	Война и мир	3	2	1500
8	Собачье сердце	5	3	550
10	Горе от ума	7	2	400
▶ 11	Отцы и дети	8	1	640
12	Шинель	6	3	500

Рисунок 57 – Выбор нескольких строк

Аналогично удалению столбцов, удаление строки может повлечь за собой каскадное удаление записей из таблиц, связанных с текущей. По этой причине при нажатии клавиши Delete выводится сообщение следующего вида (рисунок 58):

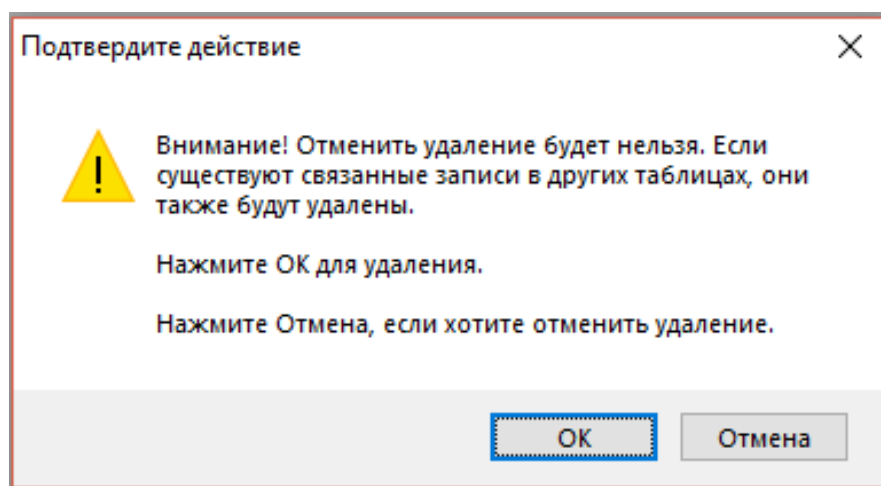


Рисунок 58 – Окно подтверждения удаления

Как видно из рисунка, отменить удаление строк можно, нажав кнопку «Отмена». Подтвердив удаление нажатием кнопки «ОК», вы уже не сможете отменить его.

Для редактирования данных в ячейке достаточно кликнуть по ней для ее выделения, а затем кликнуть снова для редактирования. Для завершения редактирования следует нажать клавишу Enter на клавиатуре. При редактировании слева от текущей строки отображается соответствующий значок (рисунок 59):



Рисунок 59 – Редактирование строки

Сортировку данных в столбце можно осуществить, нажав на заголовок столбца. Около названия столбца появится изображение треугольника, обозначающего направление сортировки: по возрастанию или по убыванию. Пример сортировки данных в таблице по одному из столбцов по возрастанию представлен на рисунке 60:

Author	Book	Publisher			
	book_id	book_name ▲	author_id	pub_id	price
	6	Анна Каренина	3	3	490
►	15	Вишневый сад	9	3	260
	5	Война и мир	3	2	1500
	13	Воскресение	3	2	310
	1	Герой нашего в...	1	1	1000
	10	Горе от ума	7	2	400
	3	Евгений Онегин	2	3	700
	2	Капитанская до	2	1	600
	11	Отцы и дети	8	1	640
	4	Преступление и...	4	2	450
	8	Собачье сердце	5	3	550
	12	Шинель	6	3	500

Рисунок 60 – Сортировка столбца "book_name" по возрастанию (по алфавиту)

Пункт меню «SQL» осуществляет вызов окна следующего вида (рисунок 61):

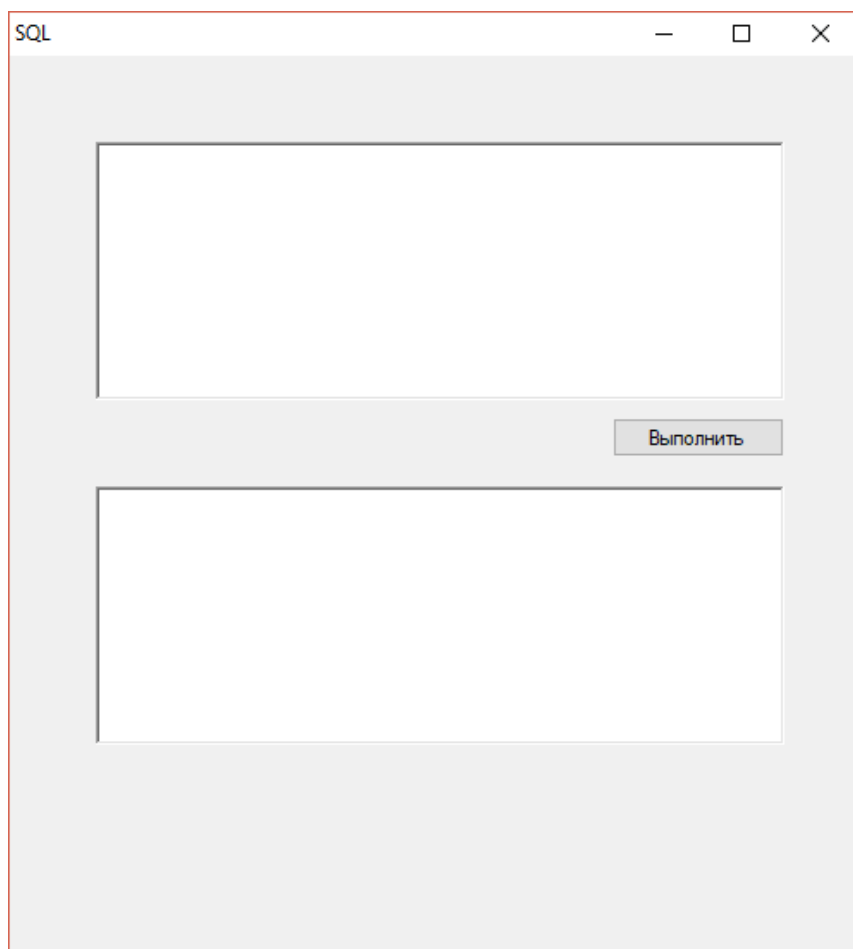


Рисунок 61 – Окно работы с базой данных посредством SQL-команд

Область в верхней части окна предназначена для ввода команды на языке SQL, в нижней части выводится результат запроса. Ключевые слова языка выделяются синим цветом для более удобного написания команд. Эта функция предназначена для того, чтобы осуществлять действия, не предусмотренные в других пунктах меню, пользоваться ей рекомендуется пользователям, обладающим навыками написания запросов на SQL, в противном случае есть вероятности случайной порчи данных в базе.

Примеры работы с окном «SQL» представлены на рисунках 62 и 63.

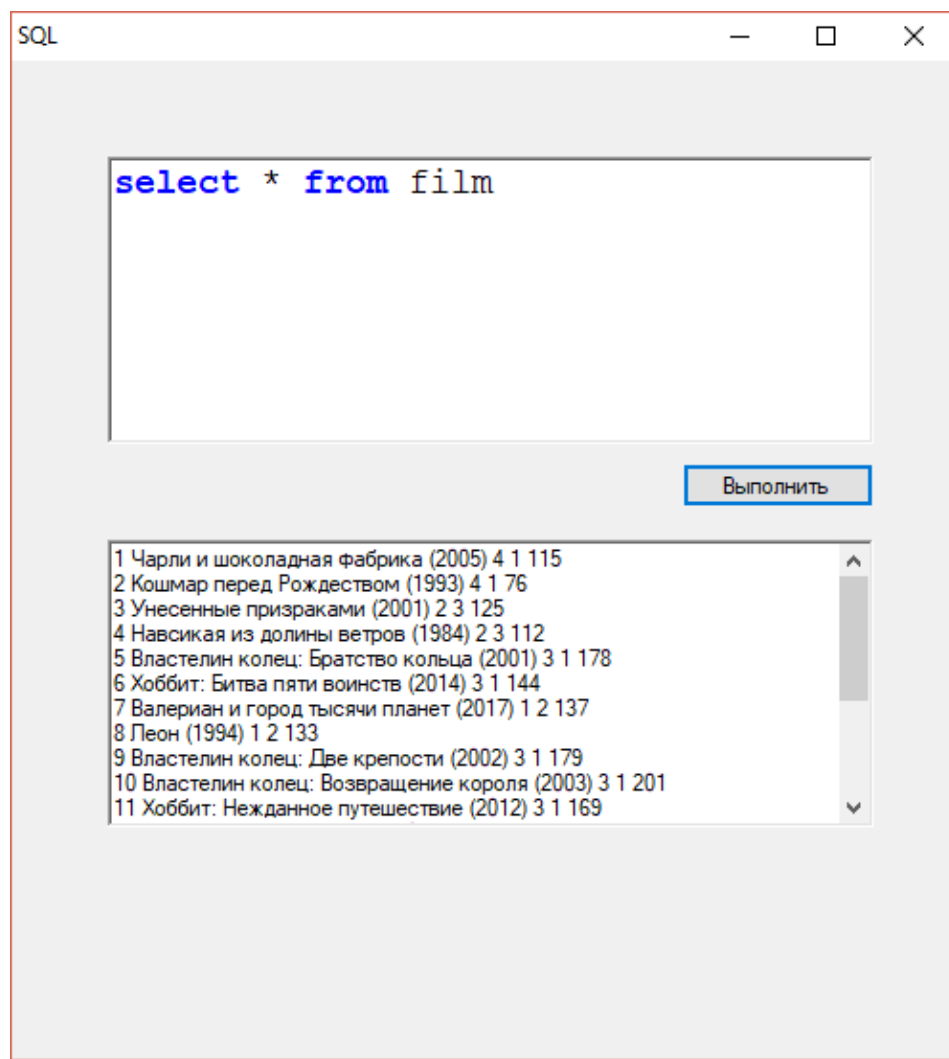


Рисунок 62 – Пример работы с окном "SQL"

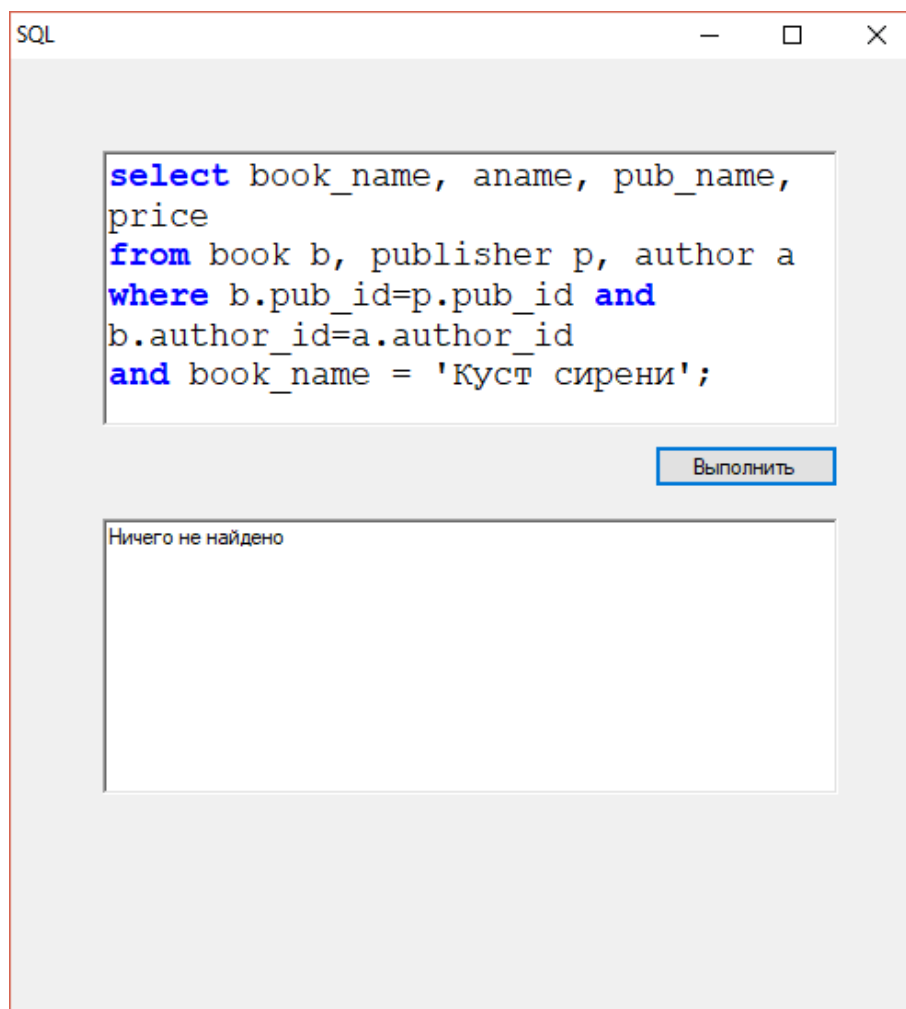


Рисунок 63 – Пример работы с окном "SQL"

Заключительный пункт меню «О программе» позволяет вывести краткую информацию о приложении (рисунок 64).

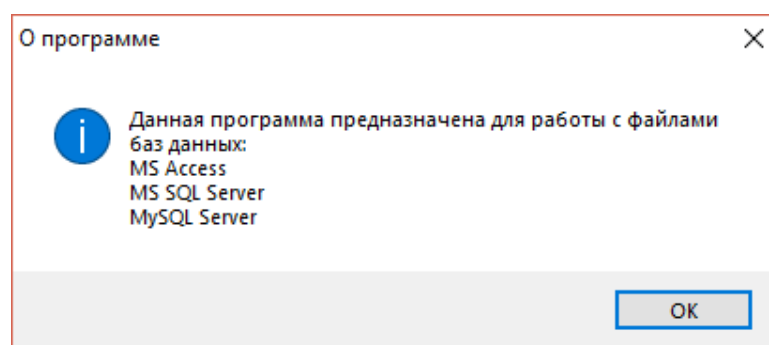


Рисунок 64 – Окно "О программе"

После окончания работы с программой следует закрыть приложение, нажав на крестик в правом верхнем углу окна.

ЗАКЛЮЧЕНИЕ

Результатом работы является клиент-серверное приложение с интуитивно понятным пользовательским интерфейсом. Приложение позволяет работать с различными базами данных, предоставляет возможность переноса данных из одной БД в другую, осуществляет обработку различных данных в БД. Данный проект полностью удовлетворяет поставленной цели.

Отладка и тестирование, проведенное на нескольких наборах баз данных, созданных с помощью выбранных СУБД, позволили исправить ошибки в программе и добиться получения корректных результатов ее работы.

Работа над проектом позволила расширить знания и улучшить навыки создания приложений с клиент-серверной архитектурой, разработки сетевых приложений, работы с базами данных.

Программа может быть улучшена в дальнейшем в соответствии с потребностями пользователей, например, можно добавить возможность работы с базами данных других форматов. Кроме того, можно добавить возможность фильтрации данных, форматирования текста, выгрузки данных в текстовый или иной файл и др.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Roff Jason. ADO ActiveX Data Objects. 2001 г.
2. OLE DB или ODBC? Семь раз отмерь. [Электронный ресурс]. – Электрон.дан. – URL: <https://www.osp.ru/winitpro/2000/01/174144/> – 20.05.2018
3. Анализ популярности языков программирования и СУБД в 2017 году [Электронный ресурс]. – Электрон.дан. – URL: <https://typicalit.net/analysis-of-the-popularity-of-programming-languages-and-dbms-in-2017/> – 20.05.2018
4. Мегаэнциклопедия Кирилла и Мефодия [Электронный ресурс]. – Электрон.дан. – URL: <http://megabook.ru/article/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82-1%D0%B5%D1%80%D0%B2%D0%B5%D1%80> – 20.05.2018
5. О модели взаимодействия клиент-сервер простыми словами. Архитектура «клиент-сервер» с примерами [Электронный ресурс]. – Электрон.дан. – URL: <http://zametkinapolyah.ru/servera-i-protokoly/o-modeli-vzaimodejstviya-klient-server-prostymi-slovami-arxitektura-klient-server-s-primerami.html> – 20.05.2018
6. Файловый архив студентов [Электронный ресурс]. – Электрон.дан. – URL: <https://studfiles.net/preview/2622854/page:16/> – 20.05.2018
7. Программирование TCP/IP [Электронный ресурс]. – Электрон.дан. – URL: <http://www.realcoding.net/articles/glava-1-vvedenie.html> – 20.05.2018
8. Протоколы TCP/IP [Электронный ресурс]. – Электрон.дан. – URL: <http://www.nsc.ru/win/fedotov/inter/tcp-ip.html> – 20.05.2018

9. Программное управление источниками данных ODBC
 [Электронный ресурс]. – Электрон.дан. – URL:
<http://www.codenet.ru/db/other/odbc/> – 20.05.2018

					ПГУ 1.090301.16.001 ПЗ	Лист
						66
Изм.	Лист	№ докум.	Подпись	Дата		

ПРИЛОЖЕНИЕ А

ЛИСТИНГИ ПРОГРАММЫ

Приложение А.1 Файл connectionData.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.Xml.Serialization;
using System.IO;
using System.Xml;
using System.Xml.Linq;

namespace MultiDB
{
    public partial class connectionData : Form
    {
        public connectionData()
        {
            InitializeComponent();
            this.StartPosition = FormStartPosition.CenterScreen;
            suggestion(IP, "//ip");
            suggestion(port, "//port");
            suggestion(serverName, "//server");
        }

        private void connect_Click(object sender, EventArgs e)
        {
            try
            {
                IPEndPoint ipHost;
                IPAddress ipAddr;
                IPEndPoint ipEndPoint;
                Socket sSender;

                ipHost = Dns.Resolve(IP.Text);
                ipAddr = ipHost.AddressList[0];
                ipEndPoint = new IPEndPoint(ipAddr,
                Convert.ToInt32(port.Text));
            }
        }
    }
}
```

```

        sSender = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);

        sSender.Connect(ipEndPoint);
        Program.IP = IP.Text;
        Program.port = port.Text;
        createConnectionString();

sSender.Send(BitConverter.GetBytes(Program.connectionString.Length));

sSender.Send(Encoding.UTF8.GetBytes(Program.connectionString));
        sSender.Send(BitConverter.GetBytes(("select
*").Length));
        sSender.Send(Encoding.UTF8.GetBytes("select
*"));
        sSender.Close();

        addToXml(IP, "ip");
        this.DialogResult = DialogResult.OK;
    }
    catch
    {
        MessageBox.Show("Ошибка соединения", "Возникла
ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

public void createConnectionString()
{
    if (type.SelectedIndex == 0)
    {
        Program.connectionString =
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + dbName.Text;
    }
    if (type.SelectedIndex == 1)
    {
        Program.connectionString =
"Provider=SQLOLEDB;Data Source=" + serverName.Text + ";Initial
Catalog=" + dbName.Text + ";Integrated Security=SSPI;";
    }
    if (type.SelectedIndex == 2)
    {
        Program.connectionString =
"Provider=MySQLPROV;Data Source=" + serverName.Text + ";Initial
Catalog=" + dbName.Text + ";user=" + user.Text +
";port=3306;password=" + password.Text;
    }
}

void suggestion(TextBox tb, string data)
{

```

```

        try
        {
            XmlDocument reader = new XmlDocument();
            reader.Load("params.xml");

            tb.AutoCompleteSource =
AutoCompleteSource.CustomSource;
            AutoCompleteStringCollection source = new
AutoCompleteStringCollection();

            XmlNode root = reader.DocumentElement;
            XmlNodeList nodeList = root.SelectNodes(data);
            foreach (XmlNode question in nodeList)
            {
                source.Add(question.InnerText);
            }

            tb.AutoCompleteCustomSource = source;
            tb.AutoCompleteMode =
AutoCompleteMode.SuggestAppend;
        }
        catch
        {
        }
    }

    void addToXml(TextBox tb, string data)
    {
        if (!checkExistInXml(tb, data))
        {
            try
            {
                FileStream fStream = new
FileStream("params.xml", FileMode.Open, FileAccess.Read,
FileShare.ReadWrite);
                XmlDocument xmlDoc = new XmlDocument();
                xmlDoc.Load(fStream);
                fStream.Close();
                XmlElement newItem;
                XmlElement newOUTERitem =
xmlDoc.CreateElement(data);
                newOUTERitem.InnerText = tb.Text;
                xmlDoc.DocumentElement.InsertAfter(
newOUTERitem,
xmlDoc.DocumentElement.LastChild);
                FileStream WRITER = new
FileStream("params.xml", FileMode.Truncate, FileAccess.Write,
FileShare.ReadWrite);
                xmlDoc.Save(WRITER);
                WRITER.Close();
            }
            catch
            {
            }
        }
    }

```

```

    }
}

bool checkExistInXml(TextBox tb, string data)
{
    XmlDocument reader = new XmlDocument();
    reader.Load("params.xml");
    XmlNode root = reader.DocumentElement;
    XmlNodeList nodeList = root.SelectNodes(data);
    int found = 0;
    foreach (XmlNode node in nodeList)
    {
        if (node.InnerText == tb.Text)
            found++;
    }

    if (found == 0)
        return false;
    else
        return true;
}
}

```

Приложение А.2 Файл mainForm.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.Odbc;
using System.Data.OleDb;
using System.Text.RegularExpressions;
using System.IO;
using System.Net;
using System.Net.Sockets;

namespace MultiDB
{
    public partial class mainForm : Form
    {
        public string[] keywords = { "ABSOLUTE", "EXEC",
"OVERLAPS", "ACTION", "EXECUTE", "PAD", "ADA", "EXISTS",
"PARTIAL", "ADD", "EXTERNAL", "PASCAL", "ALL", "EXTRACT",
"POSITION", "ALLOCATE", "FALSE", "PRECISION", "ALTER", "FETCH",
"PREPARE", "AND", "FIRST", "PRESERVE", "ANY", "FLOAT",
"PRIMARY", "ARE", "FOR", "PRIOR", "AS", "FOREIGN", "PRIVILEGES",
"ASC", "FORTRAN", "PROCEDURE", "ASSERTION", "FOUND", "PUBLIC",
"AT", "FROM", "READ", "AUTHORIZATION", "FULL", "REAL", "AVG",
"GET", "REFERENCES", "BEGIN", "GLOBAL", "RELATIVE", "BETWEEN",
"GO", "RESTRICT", "BIT", "GOTO", "REVOKE", "BIT_LENGTH",
"GRANT", "RIGHT", "BOTH", "GROUP", "ROLLBACK", "BY", "HAVING",
"ROWS", "CASCADE", "HOUR", "SCHEMA", "CASCADED", "IDENTITY",
"SCROLL", "CASE", "IMMEDIATE", "SECOND", "CAST", "IN",
"SECTION", "CATALOG", "INCLUDE", "SELECT", "CHAR", "INDEX",
"SESSION", "CHAR_LENGTH", "INDICATOR", "SESSION_USER",
"CHARACTER", "INITIALLY", "SET", "CHARACTER_LENGTH", "INNER",
"SIZE", "CHECK", "INPUT", "SMALLINT", "CLOSE", "INSENSITIVE",
"SOME", "COALESCE", "INSERT", "SPACE", "COLLATE", "INT", "SQL",
"COLLATION", "INTEGER", "SQLCA", "COLUMN", "INTERSECT",
"SQLCODE", "COMMIT", "INTERVAL", "SQLERROR", "CONNECT", "INTO",
"SQLSTATE", "CONNECTION", "IS", "SQLWARNING", "CONSTRAINT",
"ISOLATION", "SUBSTRING", "CONSTRAINTS", "JOIN", "SUM",
"CONTINUE", "KEY", "SYSTEM_USER", "CONVERT", "LANGUAGE",
"TABLE", "CORRESPONDING", "LAST", "TEMPORARY", "COUNT",
"LEADING", "THEN", "CREATE", "LEFT", "TIME", "CROSS", "LEVEL",
"TIMESTAMP", "CURRENT", "LIKE", "TIMEZONE_HOUR", "CURRENT_DATE",
"LOCAL", "TIMEZONE_MINUTE", "CURRENT_TIME", "LOWER", "TO",
"CURRENT_TIMESTAMP", "MATCH", "TRAILING", "CURRENT_USER", "MAX",
"TRANSACTION", "CURSOR", "MIN", "TRANSLATE", "DATE", "MINUTE",
"TRANSLATION", "DAY", "MODULE", "TRIM", "DEALLOCATE", "MONTH",

```

					ПГУ 1.090301.16.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		71

```
"TRUE", "DEC", "NAMES", "UNION", "DECIMAL", "NATIONAL",
"UNIQUE", "DECLARE", "NATURAL", "UNKNOWN", "DEFAULT", "NCHAR",
"UPDATE", "DEFERRABLE", "NEXT", "UPPER", "DEFERRED", "NO",
"USAGE", "DELETE", "NONE", "USER", "DESC", "NOT", "USING",
"DESCRIBE", "NULL", "VALUE", "DESCRIPTOR", "NULLIF", "VALUES",
"DIAGNOSTICS", "NUMERIC", "VARCHAR", "DISCONNECT",
"OCTET_LENGTH", "VARYING", "DISTINCT", "OF", "VIEW", "DOMAIN",
"ON", "WHEN", "DOUBLE", "ONLY", "WHENEVER", "DROP", "OPEN",
"WHERE", "ELSE", "OPTION", "WITH", "END", "OR", "WORK", "END-
EXEC", "ORDER", "WRITE", "ESCAPE", "OUTER", "YEAR", "EXCEPT",
"OUTPUT", "ZONE", "EXCEPTION", };
```

```
public static DataGridView[] grid;
public string[] changedcell = new string[3];
public List<string> PK = new List<string>();
public List<string> FK = new List<string>();
public List<string> NK = new List<string>();

public IPEndPoint ipHost;
public IPAddress ipAddr;
public IPEndPoint ipEndPoint;
public Socket sSender;

public MainForm()
{
    InitializeComponent();
    this.StartPosition =
FormStartPosition.CenterScreen;
}

private void serverConnect(byte[] sqlcom)
{
    try
    {
        ipHost = Dns.Resolve(Program.IP);
        ipAddr = ipHost.AddressList[0];
        ipEndPoint = new IPEndPoint(ipAddr,
Convert.ToInt32(Program.port));

        sSender = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);

        sSender.Connect(ipEndPoint);
        sSender.Send(BitConverter.GetBytes(0));

        sSender.Send(BitConverter.GetBytes(Program.connectionString.Length));

        sSender.Send(Encoding.UTF8.GetBytes(Program.connectionString));

        sSender.Send(BitConverter.GetBytes(sqlcom.Length));
        sSender.Send(sqlcom);
    }
}
```

```

    }
    catch (Exception ex)
    {

    }
}

private void Form1_Load(object sender, EventArgs e)
{
    connectionData cd = new connectionData();
    if (!(cd.ShowDialog() == DialogResult.OK))
    {
        this.Close();
    }
    else
    {
        if
(Program.connectionString.Contains("Jet") || Program.connectionStr
ing.Contains("SQLOLEDB"))
            ShowTables(Program.connectionString);
        else
            ShowTablesMySQL(Program.connectionString);
    }
}

private void Open_Click(object sender, EventArgs e)
{
    connectionData cd = new connectionData();
    if ((cd.ShowDialog() == DialogResult.OK)) //форма
подключения
    {
        PK.Clear();
        FK.Clear();
        NK.Clear();
        ShowTables(Program.connectionString);
    }
}

public void ShowTables(string fn)
{
    tabControl1.TabPages.Clear();
    OleDbConnection dbMySQL = new OleDbConnection(fn);

    try
    {
        dbMySQL.Open();
        OleDbCommand sqlCommand1 =
dbMySQL.CreateCommand();
        DataTable data_table2 =
dbMySQL.GetSchema("Tables");
        int k = 0;
        foreach (DataRow tbl in data_table2.Rows)

```

```

        {
            if (((string)tbl["TABLE_TYPE"]) == "TABLE"
&& (string)tbl["TABLE_NAME"]!="sysdiagrams")
                k++;
        }

        grid = new DataGridView[k];
        k = 0;

        foreach (DataRow tbl in data_table2.Rows)
        {
            if (((string)tbl["TABLE_TYPE"]) == "TABLE"
&& (string)tbl["TABLE_NAME"] != "sysdiagrams")
            {
                TabPage p = new
TabPage(Convert.ToString(tbl["TABLE_NAME"]));
                p.Name =
Convert.ToString(tbl["TABLE_NAME"]);
                tabControl1.TabPages.Add(p);
                tabControl1.SelectTab(k); //k-1
                grid[k] = new DataGridView();
                grid[k].CellValueChanged += ChangeCell;
                grid[k].AllowUserToAddRows = false;
                grid[k].KeyDown += DeleteWarning;
                TabPage page = tabControl1.SelectedTab;
                page.Controls.Add(grid[k]);
                grid[k].Size = page.Size;

                OleDbDataAdapter dbAdapter1 = new
OleDbDataAdapter(@"select * from [" +
Convert.ToString(tbl["TABLE_NAME"]) + "]", dbMySQL);
                DataTable dataTable = new DataTable();
                dbAdapter1.Fill(dataTable);
                string sqlcom = "select * from [" +
Convert.ToString(tbl["TABLE_NAME"]) + "];";
                sqlCommand1.CommandText = sqlcom;
                OleDbDataReader sqlReader1 =
sqlCommand1.ExecuteReader();
                DataTable data_table =
dbMySQL.GetSchema("Columns", new string[] { null, null,
Convert.ToString(tbl["TABLE_NAME"]) });
                int i = data_table.Rows.Count;
                string[] head = new string[i];
                grid[k].ColumnCount = i;
                i = 0;
                foreach (var item in dataTable.Columns)
                {
                    grid[k].Columns[i].Name =
item.ToString();

                    i++;
                }

                //СВЯЗИ

```

```

        string[] fkRestrictions = new string[] {
null, null, null, null, null,
Convert.ToString(tbl["TABLE_NAME"]) };
        DataTable dt =
dbMySQL.GetOleDbSchemaTable(OleDbSchemaGuid.Foreign_Keys,
fkRestrictions);
        foreach (DataRow row in dt.Rows)
        {
            PK.Add((string)row["PK_TABLE_NAME"]
+ "." + (string)row["PK_COLUMN_NAME"]);
            FK.Add((string)row["FK_TABLE_NAME"]
+ "." + (string)row["FK_COLUMN_NAME"]);
        }

        DataGridViewCellStyle columnHeaderStyle
= new DataGridViewCellStyle();
        columnHeaderStyle.BackColor =
Color.FromArgb(200, 200, 240);
        columnHeaderStyle.Font = new
Font(grid[k].DefaultCellStyle.Font, FontStyle.Bold);
        grid[k].EnableHeadersVisualStyles =
false;
        grid[k].ColumnHeadersDefaultCellStyle =
columnHeaderStyle;

        object[] tablerow = new object[i];
        int j = 0;
        while (sqlReader1.Read())
        {
            for (j = 0; j < i; j++)
            {
                try
                {
                    var value = sqlReader1[j];
                    if (value.ToString() == "")
                    {
                        value = null;
                    }
                    else
                    {
                        grid[k].Columns[j].ValueType = value.GetType();
                        tablerow[j] = value;
                    }
                }
                catch (InvalidCastException)
                {
                    tablerow[j] = null;
                }
            }
            grid[k].Rows.Add(tablerow);
        }
    }

```

```

        sqlReader1.Close();
        k++;
    }
}
dbMySQL.Close();
}

catch
{
    MessageBox.Show("При открытии базы данных
возникла ошибка", "Возникла ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
}

finally
{
    if (dbMySQL != null)
        dbMySQL.Close();
}
}

public void RefreshTable(int index)
{
    OleDbConnection dbMySQL = new
OleDbConnection(Program.connectionString);
    try
    {
        dbMySQL.Open();
        OleDbCommand sqlCommand1 =
dbMySQL.CreateCommand();
        tabControl1.SelectTab(index);
        grid[index] = new DataGridView();
        grid[index].CellValueChanged += ChangeCell;
        grid[index].AllowUserToAddRows = false;
        grid[index].KeyDown += DeleteWarning;
       TabPage page = tabControl1.SelectedTab;
        page.Controls.Clear();
        page.Controls.Add(grid[index]);
        grid[index].Size = page.Size;
        OleDbDataAdapter dbAdapter1 = new
OleDbDataAdapter(@"select * from [" + page.Text + "]", dbMySQL);
        DataTable dataTable = new DataTable();
        dbAdapter1.Fill(dataTable);
        string sqlcom = "select * from [" + page.Text +
        "]);

        sqlCommand1.CommandText = sqlcom;
        OleDbDataReader sqlReader1 =
sqlCommand1.ExecuteReader();
        DataTable data_table =
dbMySQL.GetSchema("Columns", new string[] { null, null,
page.Text });
        int i = 0;
        foreach (DataRow row in data_table.Rows)

```

```

        {
            i++;
        }
        string[] head = new string[i];
        grid[index].ColumnCount = i;
        i = 0;
        foreach (var item in dataTable.Columns)
        {
            grid[index].Columns[i].Name =
item.ToString();
            i++;
        }
        DataGridViewCellStyle columnHeaderStyle = new
DataGridViewCellStyle();
        columnHeaderStyle.BackColor =
Color.FromArgb(200, 200, 240);
        columnHeaderStyle.Font = new
Font(grid[index].DefaultCellStyle.Font, FontStyle.Bold);
        grid[index].EnableHeadersVisualStyles = false;
        grid[index].ColumnHeadersDefaultCellStyle =
columnHeaderStyle;

        object[] tablerow = new object[i];
        int j = 0;
        while (sqlReader1.Read())
        {
            for (j = 0; j < i; j++)
            {
                try
                {
                    var value = sqlReader1[j];
                    if (value.ToString() == "")
                    {
                        value = null;
                    }
                    else
                    {
                        grid[index].Columns[j].ValueType
= value.GetType();
                    }
                    tablerow[j] = value;
                }
                catch (InvalidCastException)
                {
                    tablerow[j] = null;
                }
            }
            grid[index].Rows.Add(tablerow);
        }
        sqlReader1.Close();
        dbMySQL.Close();
    }

```

```

        catch
        {
            MessageBox.Show("При открытии базы данных  
возникла ошибка", "Возникла ошибка", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        }

        finally
        {
            if (dbMySQL != null)
                dbMySQL.Close();
        }
    }

    public void ChangeCell(object sender,
    DataGridViewCellEventArgs e)
    {
        int i = tabControl1.SelectedIndex;

        changedcell[0] = e.ColumnIndex.ToString();
        changedcell[1] = e.RowIndex.ToString();
        changedcell[2] =
        grid[i].Rows[e.RowIndex].Cells[e.ColumnIndex].Value.ToString();

        string sqlcom = "update [" +
        tabControl1.SelectedTab.Text + "] set [" +
        grid[i].Columns[Convert.ToInt32(changedcell[0])].Name + "] = \"
+ changedcell[2] + \"' \" + \" where [" + grid[i].Columns[0].Name
+ "] = \" +
        grid[i].Rows[Convert.ToInt32(changedcell[1])].Cells[0].Value;
        try
        {
            serverConnect(Encoding.UTF8.GetBytes(sqlcom));
        }
        catch
        {
            MessageBox.Show("При изменении возникла ошибка",
            "Возникла ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        System.Threading.Thread.Sleep(1000);
        RefreshTable(tabControl1.SelectedIndex);
    }

    public void DeleteWarning(object sender, KeyEventArgs e)
    {
        int i = tabControl1.SelectedIndex;
        if (e.KeyCode == Keys.Delete)
        {
            if (MessageBox.Show("Внимание! Отменить удаление  
будет нельзя. Если существуют связанные записи в других  
таблицах, они также будут удалены.\n\nНажмите ОК для  
удаления.\n\nНажмите Отмена, если хотите отменить удаление.",

```

```

"Подтвердите действие", MessageBoxButtons.OKCancel,
MessageBoxIcon.Warning) == DialogResult.OK)
{
    foreach (DataGridViewRow row in
grid[i].SelectedRows)
    {
        string id =
row.Cells[0].Value.ToString();
        DeleteRow(id);
    }
    RefreshTable(i);
}
else
{
    e.SuppressKeyPress = true;
}
}

}

private string GetFromDB(string sqlcom)
{
    OleDbConnection dbMySQL = new
OleDbConnection(Program.connectionString);

    try
    {
        dbMySQL.Open();
        OleDbCommand sqlCommand1 =
dbMySQL.CreateCommand();
        sqlCommand1.CommandText = sqlcom;
        OleDbDataReader sqlReader1 =
sqlCommand1.ExecuteReader();
        string table = "";

        int j = 0;
        while (sqlReader1.Read())
        {
            while (true)
            {
                table += sqlReader1[j].ToString() + " ";
                try
                {
                    sqlReader1[j+1].ToString();
                }
                catch
                {
                    break;
                }
                j++;
            }
            table += Environment.NewLine;
            j = 0;
        }
    }
}

```

```

        }
        sqlReader1.Close();
        dbMySQL.Close();
        if (String.IsNullOrEmpty(table) == false)
        {
            return table;
        }
        else
        {
            return "Ничего не найдено";
        }
    }
    catch (OdbcException ex)
    {
        return ex.ToString() + Environment.NewLine;
    }
    finally
    {
        if (dbMySQL != null)
            dbMySQL.Close();
        int n = tabControl1.SelectedIndex;
        ShowTables(Program.connectionString);
        tabControl1.SelectTab(n);
    }
}

private void Command_TextChanged(object sender,
EventArgs e)
{
    RichTextBox Command = (RichTextBox)sender;
    Command.Font = new Font("Courier New", 14);
    foreach (string kw in keywords)
    {
        string pattern = String.Format("(\\b{0}\\b)",
kw); //слово не является частью другого (с обеих сторон пробелы)
        MatchCollection allwords =
Regex.Matches(Command.Text, pattern, RegexOptions.IgnoreCase);

        foreach (Match findword in allwords)
        {
            int caretindex = Command.SelectionStart;

            Command.SelectionStart = findword.Index;
            Command.SelectionLength = findword.Length;
            Command.SelectionColor = Color.FromArgb(0,
0, 255);

            Command.SelectionFont = new Font("Courier
New", 14, FontStyle.Bold);

            Command.SelectionStart = caretindex;
            Command.SelectionLength = 0;

```

```

        Command.SelectionColor = Color.FromArgb(0,
0, 0);
    }
}

public static string EnterTableName(string text, string
caption)
{
    Form dialogForm = new Form()
    {
        Width = 500,
        Height = 150,
        FormBorderStyle = FormBorderStyle.FixedDialog,
        Text = caption,
        StartPosition = FormStartPosition.CenterScreen
    };
    Label textLabel = new Label() { Left = 50, Top = 20,
Text = text };
    TextBox textBox = new TextBox() { Left = 50, Top =
50, Width = 400 };
    Button confirmation = new Button() { Text = "Ok",
Left = 350, Width = 100, Top = 80, DialogResult =
DialogResult.OK };
    confirmation.Click += (sender, e) => {
dialogForm.Close(); };
    dialogForm.Controls.Add(textBox);
    dialogForm.Controls.Add(confirmation);
    dialogForm.Controls.Add(textLabel);
    dialogForm.AcceptButton = confirmation;

    return dialogForm.ShowDialog() == DialogResult.OK ?
textBox.Text : "";
}

public static string[] EnterColName(string text, string
caption)
{
    Form dialogForm = new Form()
    {
        Width = 500,
        Height = 270,
        FormBorderStyle = FormBorderStyle.FixedDialog,
        Text = caption,
        StartPosition = FormStartPosition.CenterScreen
    };
    Label textLabel = new Label() { Left = 50, Top = 20,
Text = text };
    TextBox textBox = new TextBox() { Left = 50, Top =
50, Width = 400 };
    Label textLabel2 = new Label() { Left = 50, Top =
80, Text = "Выберите тип данных" };

```

```

        ComboBox combo = new ComboBox() { Left = 50, Top =
110, Width = 200 };
        Label textLabel3 = new Label() { Left = 50, Top =
140, Text = "Значение заполнения по умолчанию" };
        TextBox textBox2 = new TextBox() { Left = 50, Top =
170, Width = 400 };
        combo.Items.AddRange(new object[] { "Целое число",
            "Строка",
            "Логическое",
            "Число с плавающей запятой",
            "Дата" });
        Button confirmation = new Button() { Text = "OK",
Left = 350, Width = 100, Top = 200, DialogResult =
DialogResult.OK };
        dialogForm.Controls.Add(textBox);
        dialogForm.Controls.Add(textBox2);
        dialogForm.Controls.Add(confirmation);
        dialogForm.Controls.Add(textLabel);
        dialogForm.Controls.Add(textLabel2);
        dialogForm.Controls.Add(textLabel3);
        dialogForm.Controls.Add(combo);
        dialogForm.AcceptButton = confirmation;
        string[] chosen = new string[3];
        bool correct = false;
        dialogForm.ControlBox = false;
        while (correct == false)
        {
            if (dialogForm.ShowDialog() == DialogResult.OK)
            {
                if (textBox.Text == "" || combo.SelectedItem
== null)
                {
                    MessageBox.Show("Не все поля заполнены",
"Возникла ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
                else
                {
                    chosen[0] = textBox.Text;
                    switch ((string) combo.SelectedItem)
                    {
                        case "Целое число":
                            chosen[1] = "INTEGER";
                            break;
                        case "Строка":
                            chosen[1] = "VARCHAR";
                            break;
                        case "Логическое":
                            chosen[1] = "BIT";
                            break;
                        case "Число с плавающей запятой":
                            chosen[1] = "FLOAT";
                            break;
                        case "Дата":
                            chosen[1] = "DATETIME";
                            break;
                    }
                }
            }
        }
    }
}

```

```

    }
    if (textBox2.Text != "")
    {
        switch ((string) combo.SelectedItem)
        {
            case "Целое число":
                int x;
                if
(int.TryParse(textBox2.Text, out x))
                {
                    chosen[2] =
textBox2.Text;

                    correct = true;
                    break;
                }
                else

MessageBox.Show("Введенное значение не соответствует выбранному
типу данных", "Возникла ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);

                break;
            case "Строка":
                chosen[2] =
textBox2.Text;

                correct = true;
                break;
            case "Логическое":
                bool x1;
                if
(bool.TryParse(textBox2.Text, out x1))
                {
                    chosen[2] =
textBox2.Text;

                    correct = true;
                    break;
                }
                else

MessageBox.Show("Введенное значение не соответствует выбранному
типу данных", "Возникла ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);

                break;
            case "Число с плавающей
запятой":
                float x2;
                if
(float.TryParse(textBox2.Text, out x2))
                {
                    chosen[2] =
textBox2.Text;

                    correct = true;
                    break;
                }

```

```

else

MessageBox.Show("Введенное значение не соответствует выбранному
типу данных", "Возникла ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);

break;
case "Дата":
DateTime x3;
if
(DateTime.TryParse(textBox2.Text, out x3))
{
chosen[2] =
textBox2.Text;

correct = true;
break;
}
else
MessageBox.Show("Введенное значение не соответствует выбранному
типу данных", "Возникла ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);

break;
}
}
else
correct = true;
}

}

}
return chosen;
}

public static List<string> ChooseColumnsToDelete(string
text, string caption, TabControl tabControl1)
{
Form dialogForm = new Form()
{
Width = 300,
Height = 220,
FormBorderStyle = FormBorderStyle.FixedDialog,
Text = caption,
StartPosition = FormStartPosition.CenterScreen
};
Label textLabel = new Label() { Left = 50, Top = 20,
Text = text };
CheckedListBox columns = new CheckedListBox() { Left
= 50, Top = 50, Text = text };
Button confirmation = new Button() { Text = "Ok",
Left = 200, Width = 70, Top = 150, DialogResult =
DialogResult.OK };
confirmation.Click += (sender, e) => {
dialogForm.Close(); };

```

```

        dialogForm.Controls.Add(confirmation);
        dialogForm.Controls.Add(textLabel);
        dialogForm.AcceptButton = confirmation;
        dialogForm.Controls.Add(columns);

        OleDbConnection dbMySQL = new
OleDbConnection(Program.connectionString);

        dbMySQL.Open();
        DataTable data_table = dbMySQL.GetSchema("Columns",
new string[] { null, null, tabControl1.SelectedTab.Text });

        foreach (DataRow row in data_table.Rows)
        {
            columns.Items.Add(row["COLUMN_NAME"]);
        }

        List<string> chosen = new List<string>();
        if (dialogForm.ShowDialog() == DialogResult.OK)
        {
            for (int i = 0; i <= (columns.Items.Count - 1);
i++)
            {
                if (columns.GetItemChecked(i))
                {
                    chosen.Add(columns.Items[i].ToString());
                }
            }
            dbMySQL.Close();
            return chosen;
        }

        private void AddTable_Click(object sender, EventArgs e)
        {
            string tablename = EnterTableName("Введите название
таблицы", "Введите название таблицы");
            bool exists = false;
            foreach (TabPage tp in tabControl1.TabPages)
            {
                if (string.Equals(tp.Text, tablename,
StringComparison.OrdinalIgnoreCase))
                {
                    exists = true;
                }
            }
            if (!exists)
            {
                if (!String.IsNullOrEmpty(tablename))
                {
                    string sqlcom = "create table [" + tablename
+ "] (id int primary key)";

```

```

serverConnect(Encoding.UTF8.GetBytes(sqlcom));
                System.Threading.Thread.Sleep(1000);
                ShowTables(Program.connectionString);
                tabControl1.SelectTab(tablename);
            }
        }
        else
            MessageBox.Show("Таблица с таким именем уже
существует", "Возникла ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    }

    private void AddColumn_Click(object sender, EventArgs e)
    {
        string[] col = EnterColName("Введите название
столбца", "Введите название столбца");
        bool exists = false;
        int i = tabControl1.SelectedIndex;
        if (!String.IsNullOrEmpty(col[0]) &&
!String.IsNullOrEmpty(col[1]))
        {
            if (grid[i].Columns.Contains(col[0]))
            {
                exists = true;
            }
            if (!exists)
            {
                string sqlcom = "alter table [" +
tabControl1.SelectedTab.Text + "] add [" + col[0] + "] " +
col[1];

                if (col[2] != null)
                {
                    if
(Program.connectionString.Contains("Jet"))
                    {
                        if (col[1] == "BIT")
                        {
                            sqlcom += " default " + col[2];

serverConnect(Encoding.UTF8.GetBytes(sqlcom));
                            sqlcom = "update [" +
tabControl1.SelectedTab.Text + "] set [" + col[0] + "] = " +
col[2];

serverConnect(Encoding.UTF8.GetBytes(sqlcom));
                        }
                        else
                        {
                            sqlcom += " default '\"" + col[2]
+ "\"'";

```

```

serverConnect(Encoding.UTF8.GetBytes(sqlcom));
                                sqlcom = "update [" +
tabControl1.SelectedTab.Text + "]" set [" + col[0] + "] = \' " +
col[2] + "\'";

serverConnect(Encoding.UTF8.GetBytes(sqlcom));
                                }
                                if
(Program.connectionString.Contains("SQLOLEDB"))
                                {

serverConnect(Encoding.UTF8.GetBytes(sqlcom));
                                if (col[1] == "BIT")
                                {
                                        sqlcom = "alter table [" +
tabControl1.SelectedTab.Text + "]" add constraint col_" + col[0]
+ "_def default " + col[2] + " for " + col[0];

serverConnect(Encoding.UTF8.GetBytes(sqlcom));
                                sqlcom = "update [" +
tabControl1.SelectedTab.Text + "]" set [" + col[0] + "] = " +
col[2];

serverConnect(Encoding.UTF8.GetBytes(sqlcom));
                                }
                                else
                                {
                                        sqlcom = "alter table [" +
tabControl1.SelectedTab.Text + "]" add constraint col_" + col[0]
+ "_def default \' " + col[2] + "\' for " + col[0];

serverConnect(Encoding.UTF8.GetBytes(sqlcom));
                                sqlcom = "update [" +
tabControl1.SelectedTab.Text + "]" set [" + col[0] + "] = \' " +
col[2] + "\'";

serverConnect(Encoding.UTF8.GetBytes(sqlcom));
                                }
                                }
                                else if
(Program.connectionString.Contains("Jet"))

serverConnect(Encoding.UTF8.GetBytes(sqlcom));
                                System.Threading.Thread.Sleep(1000);
                                RefreshTable(tabControl1.SelectedIndex);
                                }
                                else

```

```

        MessageBox.Show("Столбец с таким именем
уже существует", "Возникла ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    }
}

private void DeleteTable_Click(object sender, EventArgs
e)
{
    string sqlcom = "drop table [" +
tabControl1.SelectedTab.Text + "]";
    serverConnect(Encoding.UTF8.GetBytes(sqlcom));
    System.Threading.Thread.Sleep(1000);
    ShowTables(Program.connectionString);
    if (tabControl1.TabPages.Count>0)
        tabControl1.SelectTab(0);
}

private void DeleteColumn_Click(object sender, EventArgs
e)
{
    int i = tabControl1.SelectedIndex;
    if (grid[i].ColumnCount == 1)
        MessageBox.Show("Таблица содержит всего один
столбец.\nЕсли хотите удалить таблицу, нажмите кнопку \"Удалить
таблицу\".", "Возникла ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    else
    {
        List<string> cols =
ChooseColumnsToDelete("Выберите столбцы для удаления", "Выберите
столбцы", tabControl1);
        if (cols.Count == grid[i].ColumnCount)
            MessageBox.Show("Вы выбрали все столбцы
таблицы.\nЕсли хотите удалить таблицу, нажмите кнопку \"Удалить
таблицу\".", "Возникла ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        else
        {
            foreach (string item in cols)
            {
                string sqlcom = "";
                try
                {
                    sqlcom = "alter table [" +
tabControl1.SelectedTab.Text + "] drop constraint col_" + item +
"_def";

serverConnect(Encoding.UTF8.GetBytes(sqlcom));
                    sqlcom = "alter table [" +
tabControl1.SelectedTab.Text + "] drop column [" + item + "]";

serverConnect(Encoding.UTF8.GetBytes(sqlcom));
                }
            }
        }
    }
}

```

```

        }
        catch
        {

        }
    }
    System.Threading.Thread.Sleep(1000);
    RefreshTable(tabControl1.SelectedIndex);
}
}

public void DeleteRow(string id)
{
    int i = tabControl1.SelectedIndex;
    string sqlcom = "delete from [" +
tabControl1.SelectedTab.Text + "] where [" +
grid[i].Columns[0].Name + "] = " + id;
    serverConnect(Encoding.UTF8.GetBytes(sqlcom));
}

public void SQL_Click(object sender, EventArgs e)
{
    Form dialogForm = new Form()
    {
        Width = 500,
        Height = 550,
        FormBorderStyle = FormBorderStyle.FixedDialog,
        Text = "SQL",
        StartPosition = FormStartPosition.CenterScreen
    };

    RichTextBox textBox1 = new RichTextBox() { Left =
50, Top = 50, Width = 400, Height = 150, Name = "textBox1" };
    RichTextBox textBox2 = new RichTextBox() { Left =
50, Top = 250, Width = 400, Height = 150, Name = "textBox2" };
    Button confirmation = new Button() { Text =
"Выполнить", Left = 350, Width = 100, Top = 210 };

    dialogForm.Controls.Add(textBox1);
    dialogForm.Controls.Add(textBox2);
    dialogForm.Controls.Add(confirmation);

    textBox1.TextChanged += Command_TextChanged;
    confirmation.Click += Query;
    dialogForm.ShowDialog();
}

public void Query(object sender, EventArgs e)
{
    Button bt = (Button)sender;
    Form f = (Form)bt.Parent;

```

```

        RichTextBox query = f.Controls.Find("textBox1",
true).FirstOrDefault() as RichTextBox;
        RichTextBox result = f.Controls.Find("textBox2",
true).FirstOrDefault() as RichTextBox;
        result.Text = GetFromDB(query.Text);
    }

    private void New_Click(object sender, EventArgs e)
    {

    }

    private void About_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Данная программа предназначена для
работы с файлами баз данных:\nMS Access\nMS SQL Server\nMySQL
Server", "О программе", MessageBoxButtons.OK,
MessageBoxIcon.Information);
    }

    private void AddRow_Click(object sender, EventArgs e)
    {
        Form dialogForm = new Form()
        {
            Width = 380,
            Height = 300,
            FormBorderStyle = FormBorderStyle.FixedDialog,
            Text = "Добавить строку",
            StartPosition = FormStartPosition.CenterScreen
        };
        Panel panel = new Panel();
        panel.AutoSize = true;
        dialogForm.Controls.Add(panel);
        panel.Padding = new Padding(10,10,10,10);
        int i = tabControl1.SelectedIndex;
        int k = grid[i].ColumnCount;
        Label[] labels = new Label[k];
        TextBox[] textboxes = new TextBox[k];
        int top = 20;
        int j = 0;
        for (j = 0; j < k; j++)
        {
            labels[j] = new Label() { Left = 50, Top = top,
Text = grid[i].Columns[j].Name, Name = "label" + j };
            top += 25;
            panel.Controls.Add(labels[j]);
            textboxes[j] = new TextBox { Left = 50, Top =
top, Width = 270, Name = "textbox"+j};
            top += 30;
            panel.Controls.Add(textboxes[j]);
        }
        Button confirmation = new Button() { Text =
"Добавить", Left = 270, Width = 70, Top = top};
    }

```

```

        confirmation.Click += TryToAddRow;
        panel.Controls.Add(confirmation);
        dialogForm.AutoScroll = true;
        dialogForm.ShowDialog();
    }

    public void TryToAddRow(object sender, EventArgs e)
    {
        Button bt = (Button)sender;
        Panel p = (Panel)bt.Parent;
        Form f = (Form)p.Parent;
        int n = (p.Controls.Count - 1) / 2;
        string[] labels = new string[n];
        string[] textboxes = new string[n];
        int i = 0;
        for (i = 0; i < n; i++)
        {
            Label l = p.Controls.Find("label" + i,
true).FirstOrDefault() as Label;
            labels[i] = l.Text;
            TextBox temp = p.Controls.Find("textBox" + i,
true).FirstOrDefault() as TextBox;
            textboxes[i] = temp.Text;
        }
        string sqlcom = "";
        try
        {
            sqlcom = "insert into [" +
tabControl1.SelectedTab.Text + "] ";
            sqlcom += "(";
            for (i = 0; i < n; i++)
            {
                if (!string.IsNullOrEmpty(textboxes[i]))
                {
                    if (i > 0)
                        sqlcom += ",";
                    sqlcom += "[" + labels[i] + "]";
                }
            }
            sqlcom += ")";
            sqlcom += " values (";
            for (i = 0; i < n; i++)
            {
                if (!string.IsNullOrEmpty(textboxes[i]))
                {
                    if (i > 0)
                        sqlcom += ",";
                    sqlcom += "\"" + textboxes[i] + "\"";
                }
            }
            sqlcom += ")";
        }
    }

```

```

        serverConnect (Encoding.UTF8.GetBytes (sqlcom) );

        f.Close ();
    }
    catch
    {
        MessageBox.Show ("Ошибка заполнения", "Возникла
ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    System.Threading.Thread.Sleep (1000);
    RefreshTable (tabControl1.SelectedIndex);
}

private void convertToAccess_Click (object sender,
EventArgs e)
{
    string dbName = enterDBName ("Введите название базы",
"Введите название базы");

    if (dbName != "")
    {
        dbName += ".mdb";
        string tempConnectionString =
Program.connectionString;
        Program.connectionString =
Program.connectionString =
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + dbName;
        try
        {
            ipHost = Dns.Resolve (Program.IP);
            ipAddr = ipHost.AddressList [0];
            ipEndPoint = new IPEndPoint (ipAddr,
Convert.ToInt32 (Program.port));

            sSender = new
Socket (AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);

            sSender.Connect (ipEndPoint);
            sSender.Send (BitConverter.GetBytes (1));

            sSender.Send (BitConverter.GetBytes (dbName.Length));

            sSender.Send (Encoding.UTF8.GetBytes (dbName));
        }
        catch (Exception ex)
        {
        }

        convertation ();
        Program.connectionString = tempConnectionString;
    }
}

```

```

        public static string enterDBName(string text, string
caption)
        {
            Form dialogForm = new Form()
            {
                Width = 500,
                Height = 150,
                FormBorderStyle = FormBorderStyle.FixedDialog,
                Text = caption,
                StartPosition = FormStartPosition.CenterScreen
            };
            Label textLabel = new Label() { Left = 50, Top = 20,
Text = text };
            TextBox textBox = new TextBox() { Left = 50, Top =
50, Width = 400 };
            Button confirmation = new Button() { Text = "Ok",
Left = 350, Width = 100, Top = 80, DialogResult =
DialogResult.OK };
            confirmation.Click += (sender, e) => {
dialogForm.Close(); };
            dialogForm.Controls.Add(textBox);
            dialogForm.Controls.Add(confirmation);
            dialogForm.Controls.Add(textLabel);
            dialogForm.AcceptButton = confirmation;

            return dialogForm.ShowDialog() == DialogResult.OK ?
textBox.Text : "";
        }

        private void convertToSQL_Click(object sender, EventArgs
e)
        {
            string dbName = enterDBName("Введите название базы",
"Введите название базы");
            string serverName = enterDBName("Введите имя
сервера", "Введите имя сервера");

            if (dbName != "")
            {
                string tempConnectionString =
Program.connectionString;
                Program.connectionString =
"Provider=SQLOLEDB;Data Source=" + serverName + ";Initial
Catalog=master" + ";Integrated Security=SSPI;";
                serverConnect(Encoding.UTF8.GetBytes("create
database " + dbName));
                Program.connectionString =
"Provider=SQLOLEDB;Data Source=" + serverName + ";Initial
Catalog=" + dbName + ";Integrated Security=SSPI;";
                convertation();
                Program.connectionString = tempConnectionString;
            }
        }
    }

```

```

    }

    public void convertation()
    {
        string sqlcom = "";
        int i = 0, j = 0;

        List<DataGridView> PKgrids = new
List<DataGridView>();
        List<DataGridView> FKgrids = new
List<DataGridView>();
        List<DataGridView> NKgrids = new
List<DataGridView>(); //no key
        int index = 0;
        foreach (string pkey in PK)
        {
            string[] tableAndColumn = pkey.Split('.');
            index = 0;
            foreach (TabPage page in tabControl1.TabPages)
            {
                if (page.Text.Equals(tableAndColumn[0],
StringComparison.CurrentCultureIgnoreCase))
                {
                    index =
tabControl1.TabPages.IndexOf(page);
                    if (!PKgrids.Contains(grid[index]))
                        PKgrids.Add(grid[index]);
                }
            }
        }

        foreach (string fkey in FK)
        {
            string[] tableAndColumn = fkey.Split('.');
            index = 0;
            foreach (TabPage page in tabControl1.TabPages)
            {
                if (page.Text.Equals(tableAndColumn[0],
StringComparison.CurrentCultureIgnoreCase))
                {
                    index =
tabControl1.TabPages.IndexOf(page);
                    if (!FKgrids.Contains(grid[index]))
                        FKgrids.Add(grid[index]);
                }
            }
        }

        index = 0;
        foreach (TabPage page in tabControl1.TabPages)
        {
            index = tabControl1.TabPages.IndexOf(page);

```

```

        if (!PKgrids.Contains(grid[index]) &&
!FKgrids.Contains(grid[index]))
        {
            NKgrids.Add(grid[index]);
            NK.Add(tabControl1.TabPages[index].Name);
        }

    }

    foreach (DataGridView table in PKgrids)
    {
        sqlcom = "create table [" +
PK.ElementAt(i).Split('.')[0] + "]" (";
        j = 0;
        foreach (DataGridViewTextBoxColumn dc in
table.Columns)
        {

            if (j > 0)
                sqlcom += ",";
            j++;
            sqlcom += "[" + dc.Name + "]" + " ";
            switch (dc.ValueType.Name.ToString())
            {
                case "Int32":
                    sqlcom += "INTEGER";
                    break;
                case "String":
                    sqlcom += "VARCHAR (50)";
                    break;
                case "Boolean":
                    sqlcom += "BIT";
                    break;
                case "Float":
                    sqlcom += "FLOAT";
                    break;
                case "DateTime":
                    sqlcom += "DATETIME";
                    break;
            }
            if
(PK.Contains(PK.ElementAt(i).Split('.')[0] + "." + dc.Name))
                sqlcom += " PRIMARY KEY";
            if
(FK.Contains(PK.ElementAt(i).Split('.')[0] + "." + dc.Name))
            {
                int contains =
FK.IndexOf(PK.ElementAt(i).Split('.')[0] + "." + dc.Name);
                string[] tableAndColumn =
PK.ElementAt(contains).Split('.');
                sqlcom += " FOREIGN KEY REFERENCES " +
tableAndColumn[0] + "(" + tableAndColumn[1] + ")";
            }
        }
    }

```

```

    }
    sqlcom += ")";
    serverConnect(Encoding.UTF8.GetBytes(sqlcom));

    int k = 0;
    foreach (DataGridViewRow dr in table.Rows)
    {
        k = 0;
        j = 0;
        sqlcom = "insert into [" +
PK.ElementAt(i).Split('.')[0] + "]" ";
        sqlcom += "(";
        foreach (DataGridViewTextBoxColumn column in
table.Columns)
        {
            if (dr.Cells[k].Value != null)
            {
                if (j > 0)
                    sqlcom += ",";
                sqlcom += "[" + column.Name + "]"";
                j++;
            }
            k++;
        }
        sqlcom += ")";
        sqlcom += " values (";
        j = 0;
        k = 0;
        foreach (DataGridViewTextBoxColumn column in
table.Columns)
        {
            if (dr.Cells[k].Value != null)
            {
                if (j > 0)
                    sqlcom += ",";
                sqlcom += "\"" +
dr.Cells[k].Value.ToString() + "\"";
                j++;
            }
            k++;
        }
        sqlcom += ")";

        serverConnect(Encoding.UTF8.GetBytes(sqlcom));
    }
    i++;
}

i = 0;
foreach (DataGridView table in FKgrids)
{

```

```

        sqlcom = "create table [" +
FK.ElementAt(i).Split('.')[0] + "]" (";
        j = 0;
foreach (DataGridViewTextBoxColumn dc in
table.Columns)
{
    if (j > 0)
        sqlcom += ",";
    j++;
    sqlcom += "[" + dc.Name + "]" + " ";
    switch (dc.ValueType.Name.ToString())
    {
        case "Int32":
            sqlcom += "INTEGER";
            break;
        case "String":
            sqlcom += "VARCHAR (50)";
            break;
        case "Boolean":
            sqlcom += "BIT";
            break;
        case "Float":
            sqlcom += "FLOAT";
            break;
        case "DateTime":
            sqlcom += "DATETIME";
            break;
    }
    if
(PK.Contains(FK.ElementAt(i).Split('.')[0] + "." + dc.Name))
        sqlcom += " PRIMARY KEY";
    }
    sqlcom += ")";
    serverConnect(Encoding.UTF8.GetBytes(sqlcom));
    int k = 0;
    foreach (DataGridViewRow dr in table.Rows)
    {
        k = 0;
        j = 0;
        sqlcom = "insert into [" +
FK.ElementAt(i).Split('.')[0] + "]" ";
        sqlcom += "(";
        foreach (DataGridViewTextBoxColumn column in
table.Columns)
        {
            if (dr.Cells[k].Value != null)
            {
                if (j > 0)
                    sqlcom += ",";
                sqlcom += "[" + column.Name + "]" + " ";
                j++;
            }
            k++;
        }
    }
}

```

```

        }
        sqlcom += ")";
        sqlcom += " values (";
        j = 0;
        k = 0;
        foreach (DataGridViewTextBoxColumn column in
table.Columns)
        {
            if (dr.Cells[k].Value != null)
            {
                if (j > 0)
                    sqlcom += ",";
                sqlcom += "\"" +
dr.Cells[k].Value.ToString() + "\"";
                j++;
            }
            k++;
        }
        sqlcom += ")";

serverConnect(Encoding.UTF8.GetBytes(sqlcom));

    }

    int p = 0;
    if (true)
    {
        int contains = 0;
        foreach (string fkey in FK)
        {
            if (FK.ElementAt(i).Split('.')[0] ==
fkey.Split('.')[0])
            {
                contains = p;
                string[] tableAndColumn =
PK.ElementAt(contains).Split('.');
                sqlcom = "ALTER TABLE [" +
FK.ElementAt(contains).Split('.')[0] + "] ADD CONSTRAINT FK_" +
FK.ElementAt(contains).Split('.')[1] + " FOREIGN KEY ([" +
FK.ElementAt(contains).Split('.')[1] + "]) REFERENCES [" +
tableAndColumn[0] + "] ([" + tableAndColumn[1] + "])";

serverConnect(Encoding.UTF8.GetBytes(sqlcom));
            }
            p++;
        }

        i++;
    }

    i = 0;
    foreach (DataGridView table in NKgrids)

```

```

        {
            sqlcom = "create table [" +
NK.ElementAt(i).Split('.')[0] + "]" (";
            j = 0;
            foreach (DataGridViewTextBoxColumn dc in
table.Columns)
            {
                if (j > 0)
                    sqlcom += ",";
                j++;
                sqlcom += "[" + dc.Name + "]" + " ";
                switch (dc.ValueType.Name.ToString())
                {
                    case "Int32":
                        sqlcom += "INTEGER";
                        break;
                    case "String":
                        sqlcom += "VARCHAR (50)";
                        break;
                    case "Boolean":
                        sqlcom += "BIT";
                        break;
                    case "Float":
                        sqlcom += "FLOAT";
                        break;
                    case "DateTime":
                        sqlcom += "DATETIME";
                        break;
                }
            }
            sqlcom += ")";
            serverConnect(Encoding.UTF8.GetBytes(sqlcom));

            int k = 0;
            foreach (DataGridViewRow dr in table.Rows)
            {
                k = 0;
                j = 0;
                sqlcom = "insert into [" +
NK.ElementAt(i).Split('.')[0] + "]" ";
                sqlcom += "(";
                foreach (DataGridViewTextBoxColumn column in
table.Columns)
                {
                    if (dr.Cells[k].Value != null)
                    {
                        if (j > 0)
                            sqlcom += ",";
                        sqlcom += "[" + column.Name + "]" + " ";
                        j++;
                    }
                    k++;
                }
            }
        }
    }
}

```

```

        }
        sqlcom += ") ";
        sqlcom += " values (";
        j = 0;
        k = 0;
        foreach (DataGridViewTextBoxColumn column in
table.Columns)
        {
            if (dr.Cells[k].Value != null)
            {
                if (j > 0)
                    sqlcom += ",";
                sqlcom += "\" +
dr.Cells[k].Value.ToString() + "\"";
                j++;
            }
            k++;
        }
        sqlcom += ") ";

serverConnect (Encoding.UTF8.GetBytes (sqlcom) );
    }
    i++;
}
}
}
}

```

Приложение А.3 Файл server.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.Data.OleDb;
using System.IO;

namespace Server
{
    class Server
    {
        static void Main(string[] args)
        {
            IPHostEntry ipHost = Dns.Resolve("localhost");
            IPAddress ipAddr = ipHost.AddressList[0];
            IPEndPoint ipEndPoint = new IPEndPoint(ipAddr,
8080);

            Socket sListener = new
Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
            while (true)
            {
                try
                {
                    sListener.Bind(ipEndPoint);
                    sListener.Listen(10);

                    Console.WriteLine("Сервер ожидает");
                    while (true)
                    {
                        Socket handler = sListener.Accept();

                        byte[] code = new byte[4]; //размер
строки, содержащей строку подключения
                        handler.Receive(code);

                        if (BitConverter.ToInt32(code, 0) ==
0)

                            {

                                byte[] size = new byte[4];
                                //размер строки, содержащей строку подключения
                                handler.Receive(size);
                                byte[] bConnectionString = new
byte[BitConverter.ToInt32(size, 0)]; //создаем массив под строку
подключения
```

```

handler.Receive(bConnectionString);
                                string connectionString =
Encoding.UTF8.GetString(bConnectionString);

                                handler.Receive(size);
//получаем число
                                byte[] bytes = new
byte[BitConverter.ToInt32(size, 0)]; //создаем массив нужного
размера
                                handler.Receive(bytes);
//получаем sql-команду
                                string sqlcom =
Encoding.UTF8.GetString(bytes); //конвертируем ее в строку

                                connectionString =
connectionString.Trim('\0');
                                sqlcom = sqlcom.Trim('\0');

                                Console.WriteLine(sqlcom);
                                try
                                {
                                    if
(connectionString.Contains("Jet") ||
connectionString.Contains("SQLOLEDB"))
DBConnection(connectionString, sqlcom);
                                else
DBConnectionMySQL(connectionString, sqlcom);
                                }
                                catch (Exception ex)
                                {
                                    Console.WriteLine("Возникла
ошибка!");
                                }
                                }

                                if (BitConverter.ToInt32(code, 0) ==
1)
                                {

                                    byte[] size = new byte[4];
//размер строки, содержащей имя файла
                                handler.Receive(size);
                                byte[] bDBName = new
byte[BitConverter.ToInt32(size, 0)]; //создаем массив под имя
файла
                                handler.Receive(bDBName);
                                string dbName =
Encoding.UTF8.GetString(bDBName);

```

```

        string sourceFile = @"new.mdb";
        System.IO.File.Copy(sourceFile,
dbName, true);
    }
}
}
catch (Exception ex)
{
    Console.WriteLine("Возникла ошибка!");
    Console.ReadKey();
}
}

public static void DBConnection(string connectionString,
string command)
{
    OleDbConnection dbMySQL = new
OleDbConnection(connectionString);
    dbMySQL.Open();
    OleDbCommand sqlCommand1 = dbMySQL.CreateCommand();
    sqlCommand1.CommandText = command;
    OleDbDataReader sqlReader1 =
sqlCommand1.ExecuteReader();
    sqlReader1.Close();
    dbMySQL.Close();
}
}
}

```

ПРИЛОЖЕНИЕ В

ПРЕЗЕНТАЦИЯ

Тема работы: «Клиент-серверное приложение переноса данных между базами данных»

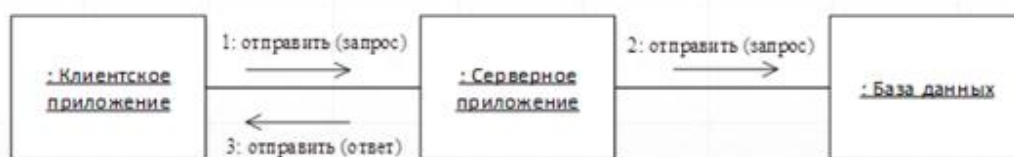
Автор работы: студентка группы 14ВВ1 Нетёсова И.С.
Руководитель: к.т.н., доцент Дорошенко И.Н.

- Цель работы – разработать клиент-серверное приложение, осуществляющее доступ к различным базам данных и работу с ними, в частности, перенос данных из одной базы в другую.
- Актуальность работы заключается в создании приложения, позволяющего открывать базы данных нескольких форматов, работать с ними и конвертировать базы из одного типа в другой.

Наиболее популярные СУБД

Rank			DBMS
Jan 2018	Dec 2017	Jan 2017	
1.	1.	1.	Oracle +
2.	2.	2.	MySQL +
3.	3.	3.	Microsoft SQL Server +
4.	4.	↑ 5.	PostgreSQL +
5.	5.	↓ 4.	MongoDB +
6.	6.	6.	DB2 +
7.	7.	↑ 8.	Microsoft Access
8.	↑ 9.	↓ 7.	Cassandra +
9.	↓ 8.	9.	Redis +
10.	10.	↑ 11.	Elasticsearch +

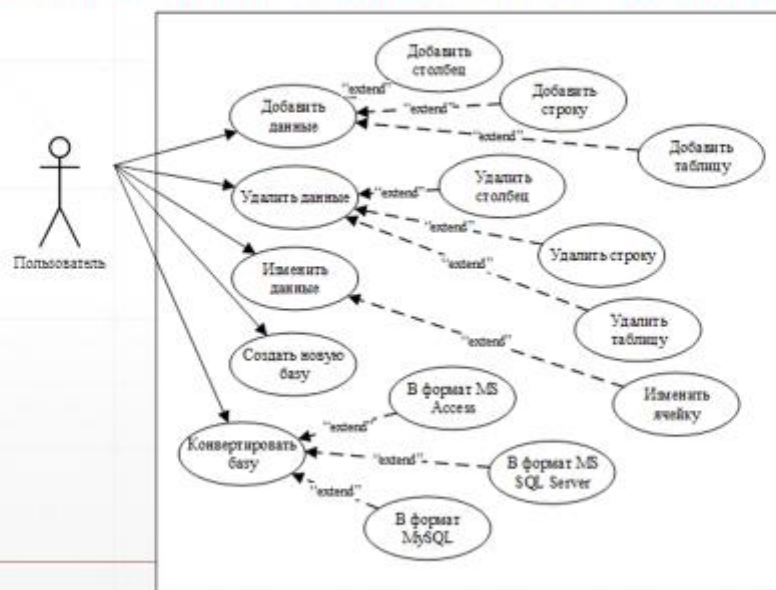
Взаимодействие частей системы



Последовательность действий при работе приложения



Взаимодействие пользователя с базой данных



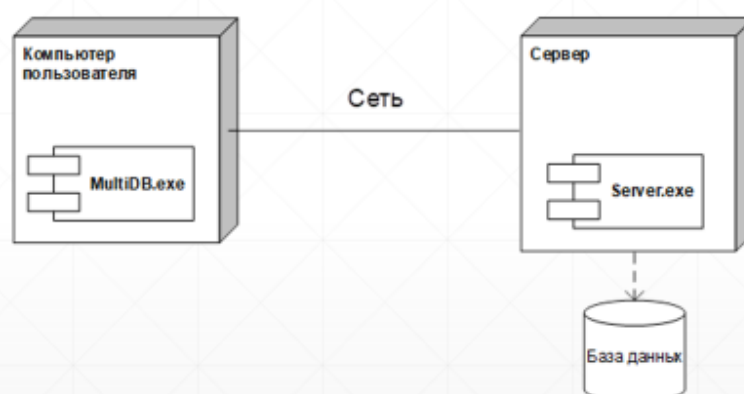
Принимаемые клиентским приложением состояния



Принимаемые серверным приложением состояния



Развертывание компонентов созданного комплекса приложений



Интерфейс окна подключения

Подключение к базе данных

IP-адрес сервера: localhost

Порт: 8080

Тип базы: MS Access

Имя сервера:

Имя базы данных: filmstore.mdb

Пользователь:

Пароль:

Подключиться

Интерфейс основного окна приложения

MultIDB

Файл Редактировать SQL О программе

Country Film FilmDirector

	file_id	film_name	director_id	country_id	time
1	1	Чарли и шоколадная фабрика (2...	4	1	115
2	2	Кошмар перед Рождеством (1993)	4	1	76
3	3	Унесенные призраками (2001)	2	3	125
4	4	Нависшая из долины ветров (19...	2	3	112
5	5	Властелин колец: Братство кол...	3	1	178
6	6	Хоббит: Битва пяти воинов (2014)	3	1	144
7	7	Валерман и город тысяч плаче...	1	2	137
8	8	Леон (1994)	1	2	133
9	9	Властелин колец: Две крепости...	3	1	179
10	10	Властелин колец: Возвращение ...	3	1	201
11	11	Хоббит: Нежданное путешествие...	3	1	169
12	12	Хоббит: Пустошь Смауга (2013)	3	1	161
13	13	Неита (1990)	1	2	117
14	14	Леон (2014)	1	2	89
15	15	Пятый элемент (1997)	1	2	126
17	17	Принцесса Мононоке (1997)	2	3	134
18	18	Небесный замок Латула (1986)	2	3	125
19	19	Холмный замок (2004)	2	3	119