Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки Программная инженерия

РАБОТА ПРОВЕРЕНА
Рецензент,

_____
«____»_____2017г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
доцент
_____А.А.Замышляева
«____»_____2017 г.

Разработка программного обеспечения для наполнения и редактирования баз данных MySQL

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–09.03.04.2017.67.ПЗ ВКР

Руководитель работы,
доцент, к. т. н.
_____/М.Ю. Катаргин
«____»_____2017 г.
Автор работы
Студент группы ЕТ-484
_____/ А.В. Орлова
«____»_____2017 г.
Нормоконтролер, доцент
_____/Т.Ю. Оленчикова
«____»_____2017 г.

Челябинск 2017

АННОТАЦИЯ

Орлова А. В. Разработка программного обеспечения для наполнения и редактирования баз данных MySQL. – Челябинск: ЮУрГУ, ЕТ-484, 114 с., 34 ил., 3 табл., библиогр. список – 23 наим., 2 прил.

Цель данной работы – разработать универсальный инструмент для добавления, редактирования и удаления информации в базах данных MySQL. В рамках дипломной работы были разработаны и описаны основные алгоритмы решения, спроектированы архитектура и интерфейс приложения. На основании чего была выполнена программная реализация системы, произведены ее отладка и тестирование.

Первый раздел работы посвящен анализу требований к программному обеспечению, описанию предметной области, обзору и сравнительному анализу аналогичных решений. Также здесь приводится обоснование выбора средств для разработки приложения.

Второй раздел содержит описание модели системы, поясняющей, каким образом будет извлекаться информация о структуре баз данных, осуществляться контроль ссылочной целостности и выполняться определение внешних ключей для различных типов таблиц.

В третьем и четвертом разделах приводится описание архитектуры и интерфейса приложения.

Пятый раздел посвящен реализации системы: здесь описываются алгоритмы работы приложения и используемые тесты для его отладки.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы. На сегодняшний день почти все информационные системы имеют в качестве основы базы данных (БД), которые позволяют эффективно хранить и структурировать большие объемы информации. Широкое использование БД различными категориями пользователей приводит к необходимости создания средств, требующих минимального количества времени и навыков на освоение принципов управления хранимыми данными.

Обычно клиентские приложения разрабатываются для конкретной информационной системы: при проектировании учитывается физическая структура задействованной базы данных, а предоставленные элементы интерфейса ориентированы на взаимодействие с определенными таблицами.

Целью данной работы является разработка универсального инструмента для эффективного решения рутинных задач по добавлению, редактированию и удалению информации в БД MySQL. Для достижения поставленной цели необходимо решить следующие *задачи*:

- выполнить анализ требований к программному обеспечению;
- провести обзор существующих решений для работы с данными таблиц MySQL, осуществить сравнительный анализ рассмотренных средств и разрабатываемого приложения;
- выбрать платформу, средства и инструменты для создания программного обеспечения;
- спроектировать архитектуру и интерфейс приложения;
- описать основные алгоритмы работы программы;
- разработать ряд тестов для отладки и тестирования системы.

1 АНАЛИЗ ТРЕБОВАНИЙ. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ. ВЫБОР СРЕДСТВ ДЛЯ СОЗДАНИЯ ПО

1.1 Постановка задачи

Поставлена задача разработать программное обеспечение (ПО) для наполнения и редактирования баз данных MySQL с использованием клиент-серверной технологии. ПО предназначено для повышения производительности работы с данными, хранящимися в базах данных MySQL.

Главной особенностью разрабатываемого ПО является возможность ограничения множества видимых записей. Другими словами, пользователь сможет скрывать ненужные записи таблицы и видеть только те, которые представляют для него интерес в данный момент.

Целями создания ПО являются:

- обеспечение возможности просмотра и редактирования содержимого таблиц и баз данных без непосредственного ввода SQL-команд;
- повышение эффективности работы администраторов, занимающихся сопровождением и обслуживанием БД;
- обеспечение целостности данных и предотвращение ввода некорректной информации при наполнении и редактировании таблиц.

В качестве пользователей данного ПО могут выступать:

- администраторы любых web-сайтов, ядром которых является база данных MySQL. Обычно в ней хранятся все необходимые сведения, участвующие в наполнении страниц сайта, и для того, чтобы отредактировать либо дополнить информационное наполнение, достаточно внести изменения в соответствующие таблицы БД. В этом случае разрабатываемое ПО выступает в роли универсальной CMS (Content Management System, система управления контентом) [1], позволяющей администратору редактировать содержимое сайта.
- преподаватели с целью наполнения данными уже существующих БД MySQL, созданных для обеспечения выполнения студентами лабораторных работ по профильным дисциплинам;
- сотрудники небольших предприятий (ЖЭКи, почтовые отделения, медицинские центры и т.д.), администрирующие БД.

1.2 Описание предметной области

1.2.1 Понятие базы данных. Модели организации данных

База данных представляет собой организованную структуру, предназначенную для хранения информации и отображения взаимосвязей между объектами. Применение баз данных в прикладных программах сокращает избыточность хранимых данных, и повышает эффективность использования информационных технологий.

Основой любой базы данных является модель данных, описывающая структуру данных и соглашения о способах их организации. Иными словами, это представленные в формальном виде объекты предметной области и их взаимосвязи, которые обычно классифицируют следующим образом:

- связь "один к одному". Одна запись объекта может быть связана только с одной записью второго объекта;
- "один ко многим". Одной записи соответствует сразу несколько записей другого объекта. Пример – у одного клиента может быть много номеров, но за номером закреплен лишь один клиент;
- "многие ко многим". Множественным записям одного объекта могут соотвествовать множественные записи другого. Пример – учитель может преподавать в нескольких классах, с другой стороны в одном классе может преподавать несколько учителей.

Использование представленных взаимосвязей определило три классические модели баз данных: иерархическую, сетевую и реляционную (см. рисунок 1.1).

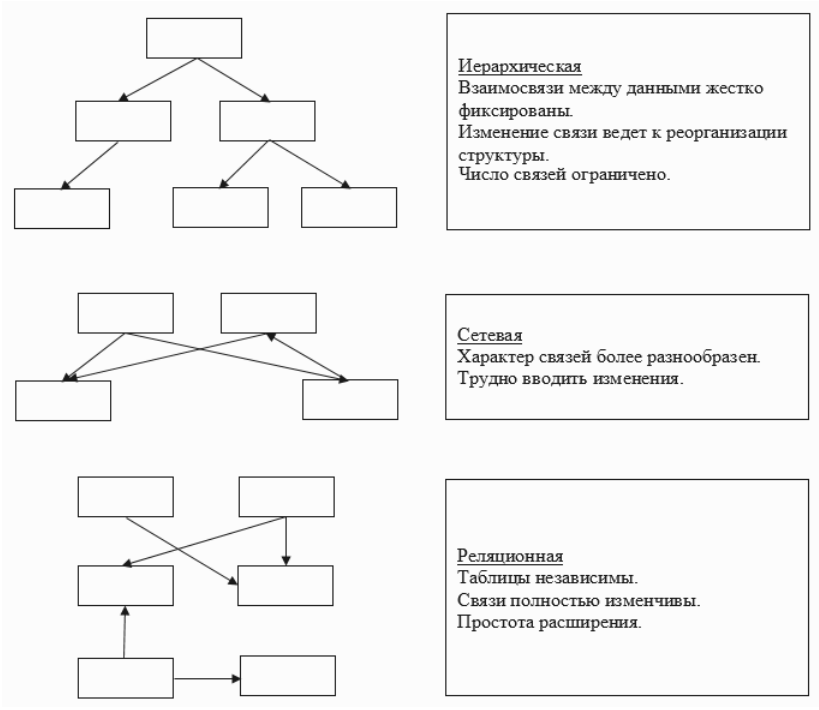

Рисунок 1.1 – Классические типы моделей данных

Иерархическая модель – граф в виде дерева, в котором каждому объекту соответствует только одна связь от объекта уровнем выше. Эту модель можно применять для проектирования баз упорядоченной структуры: например, файлового менеджера или генеалогического древа. Иерархическая модель позволяет легко описывать структуру данных, однако между элементами устанавливается жесткая фиксированная связь, вследствие чего любая ее модификация требует изменения структуры. Таким образом, скорость доступа

достигается за счет информационной гибкости. Здесь для связи объектов используется только вид связи «один ко многим».

Сетевая модель предполагает усовершенствование «дерева»: если в иерархических структурах дочерняя запись имеет только одну родительскую запись, то здесь – произвольное их число. Примером такой структуры может выступать база, которая хранит сведения о детях, посещающие различные секции. При этом возможны занятия одного и того же ребенка в разных секциях и посещение одной секции несколькими детьми. Сетевая модель обладает большей информационной гибкостью в отличие от иерархической, но сохраняет общий с ней недостаток – жесткую структуру. Поэтому при необходимости частого изменения информационной базы применяют более совершенную модель – реляционную.

Несовершенства вышерассмотренных моделей стали причиной возникновения новой реляционной модели, предложенной Коддом в 1970 г, в которой данные представляются в виде таблиц (отношений), разбитых на строки и столбцы. Такая модель предоставляет пользователям привычное и наглядное преподнесение информации. В реляционной модели достигается более высокий уровень абстракции данных, чем в иерархической или сетевой, а представление данных не зависит от способа их физической организации.

Однако несмотря на достоинства реляционной модели, она обладает следующим недостатком: обычно данные об объектах содержатся в нескольких таблицах, а значит, получение информации о нем предполагает выполнение множества операций соединения с помощью первичных и внешних ключей, что заметно замедляет обработку данных.

1.2.2 Реляционные базы данных

Реляционная модель данных (РМД) предметной области – это множество взаимосвязанных отношений, позволяющих хранить информацию об объектах. В свою очередь, реляционная БД служит хранилищем таких отношений, которые описываются в форме двумерных таблиц.

Термины РМД представлены на рисунке 1.2 и в таблице 1.1.

Рисунок 1.2 – Базовые понятия реляционной модели данных

Таблица 1.1

Термины реляционной модели

Термин РМД	**Эквивалентный термин**
Отношение	Таблица (состоит из фиксированного числа столбцов и переменного количества строк)
Кортеж, запись	Строка таблицы
Атрибут, поле	Столбец таблицы
Домен	Множество допустимых значений атрибута
Первичный ключ	Уникальный идентификатор – столбец или некоторое подмножество столбцов, которые единственным образом определяют строки
Кардинальность	Количество строк
Степень	Количество столбцов

Данные в таблицах обязаны отвечать определенным условиям:

- все значения конкретного атрибута принадлежат к одному типу;
- таблица включает в себя только уникальные кортежи;
- каждое поле обладает уникальным именем в пределах таблицы;
- допускается произвольная последовательность полей и записей.

Для выполнения действий над данными в реляционных СУБД применяется специальный язык SQL (structured query language, язык структурированных запросов), который используется для добавления, изменения и удаления данных [2]. Упрощенная схема работы с БД изображена на рисунке 1.3:

Рисунок 1.3 – Схема взаимодействия с базой данных

Пользователь БД посылает SQL-запрос к СУБД, которая обращается к базе и получает определенные данные.

1.2.3 Ключи и ссылочная целостность

В реляционных таблицах есть взаимосвязи и ссылки между ними, поэтому в большинстве СУБД существует возможность указывать так называемые первичные и внешние ключи и использовать механизмы поддержания ссылочной целостности.

1.2.3.1 Первичный ключ

Он однозначно идентифицирует запись и гарантирует, что в таблице отсутствуют одинаковые строки. Поля первичного ключа не могут содержать неопределенные значения. В таблице допускается всего один первичный ключ, им может являться как один столбец, так и группа столбцов.

1.2.3.2 Уникальный ключ

Это столбец или группа столбцов, в которую запрещается вносить повторяющиеся значения. Отличается от первичного тем, что:

- для таблицы может быть задано несколько уникальных ключей;
- они могут содержать NULL-значения.

1.2.3.3 Внешний ключ

Применяется для организации между таблицами связи «один ко многим», и поддержки ссылочной целостности. Представляет собой один или несколько столбцов, ссылающихся на столбцы другой (или этой же) таблицы.

Таблица и ее столбцы, на которые ссылается внешний ключ, называются родительской таблицей и родительским ключом соответственно. Родительский ключ обязан быть первичным или уникальным; значения внешнего ключа могут дублироваться.

1.2.3.4 Ограничения целостности реляционной модели

Ограничения можно поделить на несколько групп:

- ограничения целостности сущностей. Они требуют наличие только уникальных записей таблицы (всякое отношение должно обладать первичным ключом). Отсюда следуют два условия: в таблице отсутствуют записи-дубликаты; столбцы, входящие в первичный ключ, не содержат NULL-значений. Отображением сущности может являться не всякая таблица, однако для них также можно потребовать уникальности;
- ограничения ссылочной целостности. Значения внешнего ключа должны соответствовать какому-нибудь значению родительского ключа или быть неопределенными, если это допускается;
- отдельно следует рассматривать семантические ограничения, накладывающие на поля определенные условия или функциональные зависимости. Для каждой базы/таблицы такие ограничения индивидуальны, поэтому в данной квалификационной работе они рассматриваться не будут.

1.2.3.5 Ссылочная целостность

Она гарантирует, что ссылки между таблицами являются допустимыми. Требование ссылочной целостности уже было указано раннее: недопустимо

появление строк, внешний ключ которых не равен какому-либо значению родительского ключа.

В стандарте SQL-92 существуют четыре механизма, поддерживающие ссылочную целостность при удалении или обновлении ключа в родительских таблицах:

- CASCADE. При удалении или обновлении родительского ключа обеспечивается удаление или обновление внешнего ключа в дочерних таблицах;
- SET NULL. При удалении или обновлении родительского ключа в соответствующие внешние ключи записывается NULL-значение;
- SET DEFAULT. В соответствующие внешние ключи записываются значения по умолчанию;
- NO ACTION. Удаление или изменение ключа запрещается, если для него имеются зависимые записи.

1.2.3.6 Ключи и ссылочная целостность в MySQL

Ссылочная целостность и все вышеуказанные механизмы ее поддержания присутствуют в MySQL 5.0 и выше (при использовании таблиц типа InnoDB) [3].

1.2.4 Системы хранения данных в MySQL

MySQL позволяет работать с несколькими типами (форматами) таблиц, отличающихся способом хранения информации, скоростью работы и функционалом [4]. Их можно разделить на две группы: транзакционные типы и типы, которые не поддерживают транзакции.

Транзакционные таблицы (transaction-safe tables, TST) умеют восстанавливать данные при возникновении системных или аппаратных неполадок (для этих целей используются резервные копии и журналы транзакций) и отменять внесенные изменения. Кроме того, они лучше обеспечивают параллелизм при одновременных запросах к таблице. В свою очередь, не транзакционные таблицы (non-transaction-safe tables, NTST) менее надежны, но выигрывают в производительности и в объемах занимаемого дискового пространства. Далее приводится краткая характеристика каждого типа.

1.2.4.1 Формат InnoDB

Отличается высокой производительностью и надежностью хранения данных. Все таблицы содержатся в едином файле, который может быть поделен на несколько частей и распределен по разным дискам или хостам. Это единственный тип в MySQL, который поддерживает каскадное удаление/обновление и систему внешних ключей.

1.2.4.2 Формат BDB (BerkeleyDB)

В MySQL применяется для работы с таблицами только в режиме транзакций, так как не поддерживает язык запросов. Транзакции реализуются посредством журнальных файлов, где хранятся сведения о внесенных изменениях. Если транзакция отменяется, библиотечные BDB-функции обращаются к журнальным файлам для выполнения "обратных" исправлений.

В BDB-таблицах требуется наличие первичного ключа, и, если таковые отсутствуют, MySQL самостоятельно создаст внутренний первичный ключ. Табличные данные хранятся в виде бинарного дерева: это увеличивает занимаемое ею место на диске и замедляет выборку всех записей, однако поиск отдельных значений ускоряется.

1.2.4.3 Формат MyISAM

MyISAM-таблицы – кроссплатформенные, а значит табличные файлы можно перемещать между разными компьютерами без дополнительных преобразований. MySQL хранит счетчик подключений к MyISAM-таблице, который сбрасывается в ноль при ее закрытии и остается положительным числом в случае аварийного завершения работы. В последнем случае сервер сможет обнаружить проблему и при необходимости восстановить поврежденную информацию.

1.2.4.4 Формат MERGE

Представляет собой множество идентичных MyISAM-таблиц (с одинаковой структурой: столбцами, индексами и т.д.), с которыми можно взаимодействовать как с одной единой таблицей. Применяется главным образом для снятия ограничения на объем: например, если операционная или файловая системы не позволяют создавать таблицы больше 4 Гб. Они не поддерживают операцию вставки, поскольку MySQL не сможет определить, в какую из исходных таблиц должны быть помещены новые записи; извлечение данных осуществляется медленнее, чем из таблиц других типов.

1.2.4.5 Формат MEMORY (HEAP)

Хранится в оперативной памяти, а не в файловой системе, поэтому доступ к ней осуществляется очень быстро. Проблемой является потеря всех данных в случае сбоя работы сервера, поэтому в таблицах такого типа обычно содержится временная информация, которую можно без проблем восстановить. MEMORY не обладает многими возможностями обычных таблиц, в частности не может иметь столбцы типов BLOB и TEXT.

1.2.5 Функции СУБД

На уровне СУБД решаются следующие важные задачи:

- обрабатываются данные, содержащиеся во внешней памяти или в буфере оперативной памяти;
- выполняются транзакции;
- обеспечивается журнализация изменений, восстановление после сбоев, целостность и безопасность данных;
- поддерживаются полномочия и языки управления БД.

1.2.5.1 Управление данными во внешней памяти

СУБД предоставляет необходимые структуры внешней памяти для служебных целей и хранения информации, содержащейся в базе. Работа может производиться на уровне функционирования устройств внешней памяти или с помощью файловых систем. В любом случае пользователям нет необходимости знать используемую структуру или тип организации файлов.

1.2.5.2 Управление транзакциями

Транзакции применяются для обеспечения целостности хранимых данных. Под термином «транзакция» подразумевается набор инструкций, который контролируется СУБД от начала и до конца. Если по каким-либо причинам (при возникновении сбоев, отказов или ошибок в приложении) транзакция не завершается, то она полностью отменяется. Для нее характерны 3 ключевых свойства:
- атомарность (выполняются все инструкции, содержащиеся в транзакции);
- изолированность (транзакции, обрабатывающиеся одновременно, не оказывают влияния друг на друга);
- долговечность (если транзакция зафиксирована, то результат ее выполнения сохраняется даже в случае сбоев в системе).

1.2.5.3 Ведение журнала

Благодаря журнализации изменений обеспечивается надежность хранения данных: в случае аппаратного или программного сбоя СУБД сможет вернуться к последнему исправному состоянию.

1.2.5.4 Обеспечение целостности и безопасности

Принцип целостности гарантирует исправное функционирование базы и подразумевает, что в ней находится непротиворечивая и полная информация, которая верно описывает предметную область. Это поддерживается посредством отслеживания ограничений целостности, которым должны отвечать хранимые данные.

Безопасность в СУБД достигается благодаря шифрованию, паролям и ограничению доступа к базе и ее компонентам (таблицам, столбцам и др.).

1.2.5.5 Поддержка языков управления

Для взаимодействия с базами данных применяются язык описания структуры данных (SDL, Specification and Description Language) и язык манипулирования данными (DML, Data Manipulation Language). Большая часть реляционных СУБД поддерживают единый язык, включающий все необходимые средства для работы с БД (язык SQL).

1.3 Анализ требований к программе

1.3.1 Общие требования к программе

Поставлена задача разработать ПО для наполнения и редактирования баз данных MySQL. Целями создания ПО являются:

- обеспечение просмотра и редактирования содержимого таблиц и баз данных без непосредственного ввода SQL-команд;
- повышение эффективности работы администраторов, занимающихся сопровождением и обслуживанием БД;
- обеспечение целостности данных и предотвращение ввода некорректной информации при наполнении и редактировании таблиц; обеспечение возможности обработки внешних ключей.

1.3.1.1 Требования к функциональным характеристикам

Разрабатываемое программное обеспечение должно обладать следующими функциями:

- работать под управлением ОС Windows;
- использовать для соединения и обмена данными протокол TCP/IP (Transmission Control Protocol);
- при запуске приложения обеспечивать авторизацию пользователя: запрашивать сервер, логин и пароль;
- главная форма приложения должна отображать перечень всех баз данных, хранящихся на сервере. При выборе из списка необходимой базы, пользователю должен предоставляться список ее таблиц;
- предоставлять возможность просмотра хранящейся в базе информации в табличной форме;
- иметь возможность удаления, редактирования и добавления записей таблицы;
- позволять настраивать удобное расположение таблиц: перемещать, изменять размеры, закрывать/открывать.
- обеспечивать корректную обработку содержимого полей типа BLOB (Binary Large Object, двоичный большой объект), хранящих различные бинарные данные;
- осуществлять возможность определения связанных таблиц родитель-ребенок (master-detail). Эту связь, в случае, если обе таблицы открыты, визуально отображать в виде стрелки от таблицы-родителя к дочерней таблице;

- обеспечивать фильтрацию данных в дочерних таблицах. Если открыты таблицы, связанные между собой связью master-detail, то в detail-таблице отображать только те записи, которые являются подчиненными по отношению к текущей записи родительской таблицы. В противном случае, если таблица не связана ни с какой другой (родительские таблицы не открыты), то отображать все хранящиеся в ней данные.

1.3.1.2 Требования к режиму функционирования ПО

Требуется, чтобы программное обеспечение обеспечивало доступ к серверу баз данных с рабочих мест пользователей в удаленном режиме. Основная бизнес-логика (авторизация, обработка ошибок, операции с уже загруженными данными) должна быть реализована на клиенте. Сервер же исполняет запросы на манипулирование данными (чтение, запись, удаление и модификацию).

1.3.1.3 Требования к внешнему оформлению и диалогу с пользователем

Взаимодействие пользователей с приложением должно осуществляться с помощью визуального графического интерфейса (graphical user interface, GUI), предоставляющего доступ ко всем предусмотренным функциональным возможностям. Он не должен быть перегружен графическими элементами и избыточной информацией. Все компоненты интерфейса должны быть приближены к традиционным формам представления с учетом их смыслового значения.

Необходимо обеспечить своевременную обратную связь для действий пользователя посредством визуального подтверждения того, что приложение восприняло и выполнило (или не выполнило) команду. Должны быть предоставлены сведения о состоянии текущего процесса, с возможностью остановить его в случае необходимости.

Система должна предупреждать пользователей, если они могут нанести вред системе или данным. При наличии ошибок должно выводиться сообщение с наименованием ошибки и способами её устранения; все сообщения и названия экранных форм должны быть на русском языке.

Контроль над приложением должен осуществляться с помощью меню, значков и других элементов, рассчитанных на применение мыши; при заполнении и редактировании числовых или текстовых полей должна использоваться клавиатура.

1.3.2 Требования к надежности

1.3.2.1 Требования к диагностированию ПО

Реализовать возможность ведения журнала действий пользователей (логов) с целью выявления несанкционированного входа, ошибок ввода информации и

аварийных ситуаций. В этом случае в журнал должны заноситься данные, идентифицирующие пользователя, и сведения о его действиях.

1.3.2.2 Требования к информационной безопасности

Защита информации от несанкционированного доступа обязана удовлетворять следующим требованиям:

- вход в систему разрешается только зарегистрированным пользователям;
- программные средства защиты не должны реализовываться в ущерб основным функциональным характеристикам ПО, таким как быстродействие и надежность.

1.3.2.3 Обработка отказов программы и аварийных ситуаций

Отказы программы вследствие некорректных действий пользователя при взаимодействии с приложением недопустимы. Должна обеспечиваться обработка ошибок, которые могут быть вызваны вводом неверных значений или неправильным форматом данных. Все аварийные ситуации (из-за недопустимых или непредусмотренных действий пользователей) должны обрабатываться на программном уровне. В этих случаях нужно выдавать соответствующее сообщение, а приложение возвращать в состояние, предшествовавшее ошибке или аварии без потери обрабатываемой информации.

1.3.3 Условия эксплуатации

1.3.3.1 Требования к квалификации персонала

Пользователь, работающий с приложением, должен являться администратором базы данных, иметь доступ ко всем базам и таблицам, а также обладать правами наполнения, редактирования и удаления записей.

Предполагается, что пользователь обладает базовыми знаниями в области администрирования баз данных, имеет опыт работы с windows-приложениями: умеет вводить информацию в экранные формы, работать с контекстным и главным меню и т.д.

1.3.3.2 Требования к составу и параметрам технических средств

Для функционирования приложения требуется:

- операционная система Windows XP и выше;
- Microsoft.NET Framework версии 4.5 и выше;
- доступ к СУБД MySQL не ниже версии 5.0;
- доступ к серверу баз данных MySQL;
- оперативная память не менее 20 Мб;
- свободная память на жестком диске не менее 25 Мб.

1.3.4 Перспективы модернизации и развития

При разработке приложения необходимо учесть возможность дальнейшего совершенствования ПО, а именно обеспечения работы с базами данных различных форматов (не только MySQL).

1.4 Анализ существующих программ

Функционал, подобный разрабатываемому ПО, включающий в себя редактирование, добавление, удаление данных в таблицах БД, имеют различные менеджеры и приложения для администрирования СУБД. Помимо этого, они часто включают в себя визуальный конструктор баз данных; графические и текстовые инструменты для построения запросов; средства импорта и экспорта данных из различных источников; работу с триггерами, событиями, хранимыми процедурами; конструктор диаграмм и прочее [5].

Поэтому, принимая во внимание то, что готовые решения для администрирования СУБД MySQL оснащены различными дополнительными возможностями, будем анализировать ограниченный их функционал, имеющий отношение к теме квалификационной работы.

Проведем подробный разбор четырех продуктов: PHPMyAdmin, Adminer, являющихся WEB-приложениями, и SQLyog, HeidiSQL, представляющими Desktop-приложения, работающие на ОС Windows. Сравним их между собой, обозначим преимущества и недостатки.

1.4.1 PHPMyAdmin

PHPMyAdmin – бесплатное web-приложение для администрирования СУБД MySQL. Работать с ним, в том числе просматривать и править содержимое таблиц, можно непосредственно в браузере.

Администратору доступны создание и настройка как баз данных, так и аккаунтов для пользователей с различными привилегиями. PHPMyAdmin обладает встроенными средствами для операций с базами и таблицами: при наличии соответствующих полномочий пользователь вправе менять последовательность полей в таблицах, переименовывать их или перемещать, менять тип, добавлять примечания и т.д. Помимо стандартных функций поддерживается резервирование и восстановление данных из архивов, экспорт информации в различных форматах, т.е. предоставлен полный контроль над сервером.

Интерфейс программы представляет собой окно, поделенное на две части: в первой из них указан либо перечень всех доступных баз, либо список таблиц конкретной базы данных, если пользователь не является администратором. При выборе необходимой таблицы в главной рабочей части отображаются ее параметры и настройки. Наверху расположены вкладки для управления таблицей, другие возможности представлены в виде пиктограмм-ссылок (см. рисунок 1.4).

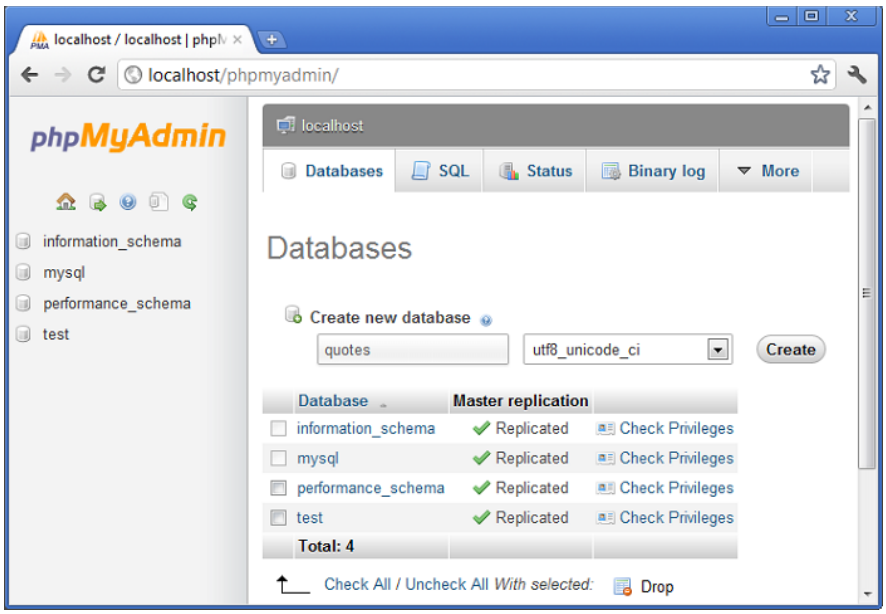

Рисунок 1.4 – Интерфейс web-приложения «PHPMyAdmin»

Программа позволяет осуществлять SQL-запросы, каждый из которых снабжается подробной информацией о времени обработки, количестве затронутых рядов, использованных полей при построении (может пригодиться при отладке и ручной оптимизации).

На основании вышеизложенного обозначим основные преимущества и недостатки продукта. Преимущества:

- динамичное развитие продукта, с учетом всех нововведений СУБД MySQL;
- управление СУБД MySQL напрямую без ввода SQL-команд;
- наличие лицензии GNU (General Public License, лицензия на свободное программное обеспечение), которая допускает интегрирование PHPMyAdmin в самостоятельные проекты. Так, PHPMyAdmin включен в популярные пакеты серверных программ, таких как Apache и Denwer, которые применяют web-разработчики для отладки сайтов;
- реализация на PHP позволяет использовать решение практически где угодно: не требуется дополнительное ПО, достаточно лишь наличие браузера;
- предоставляется разграничение прав. Один и тот же скрипт могут использовать и обычные пользователи, и администраторы. Отличие между ними заключается в том, что первые не обладают некоторыми полномочиями, поэтому им недоступна часть функционала (например, выбор сервера, доступ к служебным базам);
- реализована возможность запускать приложение непосредственно на сервере, что удобно, если хостинг-провайдер запрещает удаленный доступ к базе.

Недостатки:

- в таблицах БД доступно только отображение данных, а работа с каждой записью выполняется на отдельной странице, что осложняет их наполнение и редактирование;

- от версии к версии он дополняется все новым кодом, следовательно, увеличивается в размерах. Многоязычность, поддержка всевозможных кодировок, продуманность операций требуют дополнительных ресурсов. При необходимости срочного редактирования нескольких строк в таблице, имея в наличии лишь ftp-доступ (File Transfer Protocol, протокол передачи файлов) к сайту, распаковка и установка PHPMyAdmin на сервере весьма утомительна;
- в более «старых» версиях PHPMyAdmin (вплоть до версий 3.3.x) отсутствует поддержка AJAX, позволяющая просматривать и редактировать данные в БД без перезагрузки страницы.

1.4.2 Adminer

Adminer (бывший PHPMinAdmin) – это бесплатный инструмент для администрирования ряда СУБД: MongoDB, MySQL, PostgreSQL, SQLite, MS SQL и других (см. рисунок 1.5).

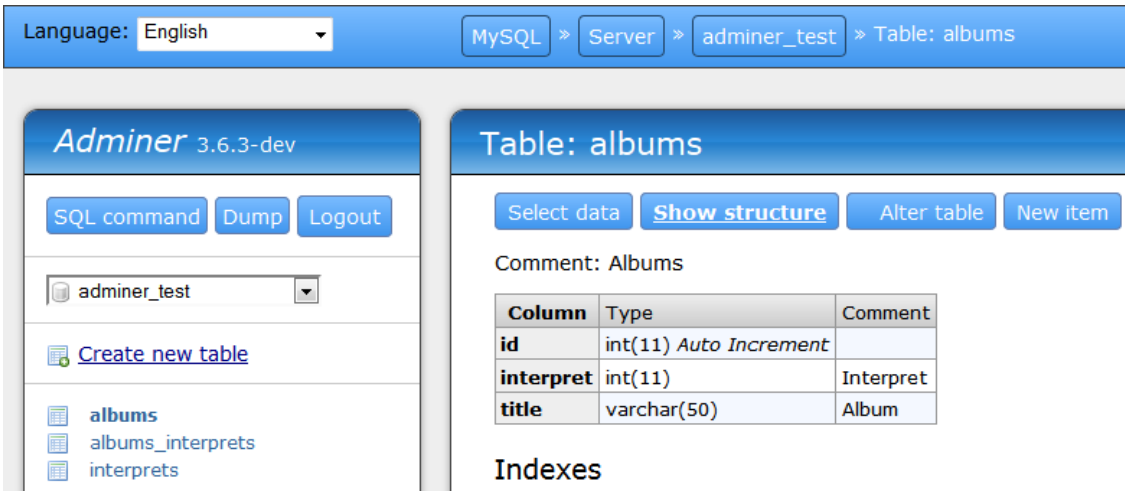

Рисунок 1.5 – Интерфейс web-приложения «Adminer»

Он представлен в виде единственного php-файла, который не требует установки и поддерживает большую часть возможностей phpMyAdmin и других подобных средств, а именно:

- выбор базы, таблицы, просмотр и правка содержимого;
- сортировку и поиск информации;
- работу с правами доступа, представлениями, триггерами, хранимыми процедурами;
- область для ввода произвольных SQL-выражений и поддержку истории команд;
- подсветку SQL-синтаксиса;
- экспорт баз данных и их таблиц;
- визуальный редактор ER-схем;
- защиту от взлома посредством межсайтового скриптинга (XSS, Cross-Site Scripting), подделок запросов (CSRF, Cross Site Request Forgery), внедрения в запрос произвольного SQL-кода (SQL-инъекций), похищений сессий.

Преимущества:

- обладает «встроенной» правкой данных. Многие поля можно отредактировать после двойного щелчка мыши по их содержимому, а изменения в базе вступят в силу после нажатия кнопки «Сохранить»;
- поддерживает такие СУБД, как MySQL, PostgreSQL, SQLite, MS SQL, Oracle, SimpleDB, Elasticsearch, MongoDB;
- имеет небольшой размер (100-500 Кб, в зависимости от версии), быстро устанавливается на любой сервер.

Недостатки:

- уступает PHPMyAdmin в функциональности.

1.4.3 SQLyog

SQLyog – менеджер для работы с MySQL (см. рисунок 1.6), поддерживающий HTTP и SSH-туннелирование в случае, если стандартный порт для соединения с базой заблокирован. Производит всевозможные действия над базами: создание, редактирование, синхронизацию данных, копирование (в том числе и на другой хост).

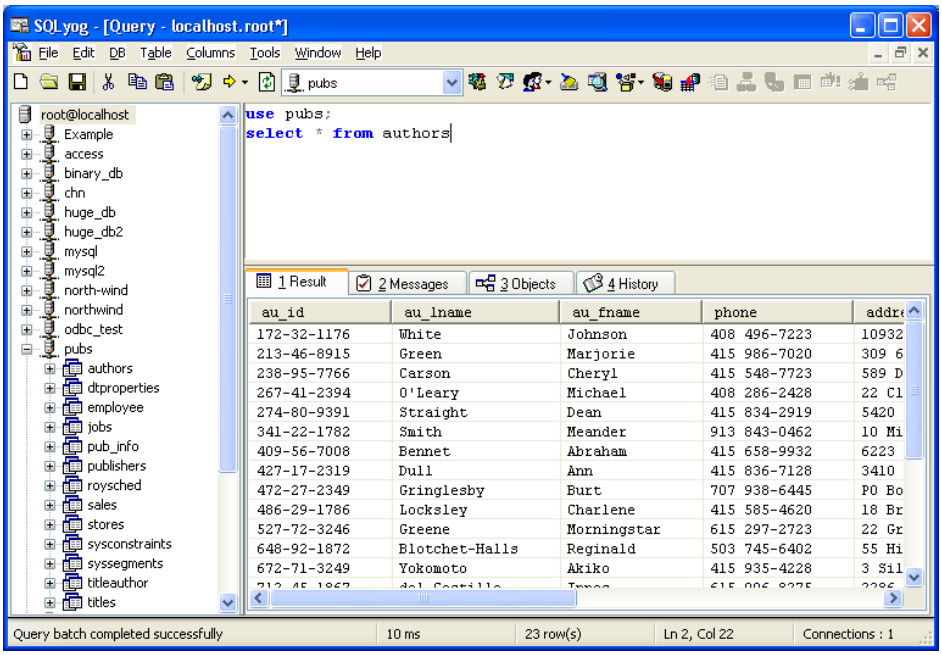

Рисунок 1.6 – Интерфейс менеджера «SQLyog»

SQLyog отличается поддержкой системы Smart Autocomplete, автодополняющей имена баз и таблиц, и встроенным профайлером запросов. Он демонстрирует количество времени на выполнение запроса, анализируя каждую из его составляющих (время передачи, обработки, открытия или блокировки таблиц, и т.д.). Используя профайлер, можно оценить результат внесения изменений в конфигурации сервера, обнаружить дорогие операции и попытаться их модифицировать.

Преимущества:

- в случае, если запрещен удаленный доступ к MySQL или недоступен MySQL-порт, получить доступ к серверу позволит HTTP и SSH-туннелирование;
- данные, полученные в результате запросов, можно отфильтровать при помощи контекстного меню;
- многоформатный редактор BLOB-полей.

Недостатки:

- после 30-дневного ознакомительного периода программу надо зарегистрировать; беспокоит окно, напоминающее о регистрации;
- SQLyog имеет две версии: бесплатную Community и коммерческую Enterprise. Большая часть функционала программы наподобие визуальных редакторов, разграничения прав и автодополнения запросов в коммерческой версии урезаны;
- поддерживается только на ОС Windows.

1.4.4 HeidiSQL

HeidiSQL – это бесплатная графическая оболочка, с помощью которой автоматизируются задачи проектирования, разработки и обслуживания баз данных MySQL, Microsoft SQL Server, PostgreSQL (см. рисунок 1.7).

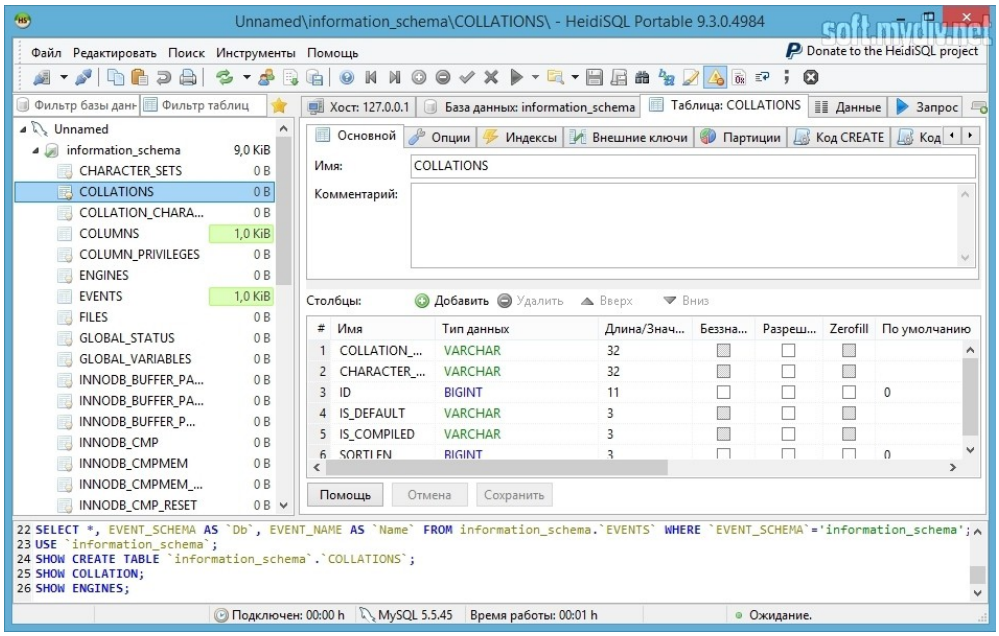

Рисунок 1.7 – Интерфейс приложения «HeidiSQL»

HeidiSQL позволяет экспортировать структуры и данные в SQL-файл, буфер обмена, на другие серверы; управлять таблицами и правами пользователей. Его можно расценивать как инструмент администрирования сервера и разработки небольших баз, а не как среду проектирования БД.

Преимущества:

- соединение с несколькими серверами одновременно; экспортирование данных и их структуры с одного сервера на другой;

- работа с данными реализуется через таблицу, ячейки которой можно редактировать;
- для сортировки данных можно использовать фильтры, отбирая записи в соответствии с заданной маской.

Недостатки:

- в отличие от SQLyog не соединится с сервером, если заблокирован MySQL-порт, так как встроенных механизмов для обхода этого препятствия не содержит;
- поддерживается только на ОС Windows.

1.4.5 Сравнение с разрабатываемым ПО

Помимо вышеперечисленных продуктов, также были рассмотрены следующие:

- MyDB Studio
- EMS SQL Manager
- Workbench
- Navicat
- dbForge Studio
- SQL Buddy

Каждый из них обладает своими плюсами и минусами, однако иных функциональных возможностей, отличных от вышерассмотренных программ, они не содержат, поэтому подробно не рассматриваются.

Проведем краткий анализ готовых решений путем их сравнения с разрабатываемым в квалификационной работе ПО. Будем исследовать следующие возможности:

- обзор всех доступных пользователю баз данных и их таблиц;
- одновременное открытие нескольких таблиц и настройка их удобного расположения;
- визуализация связей между открытыми таблицами;
- просмотр содержимого BLOB-полей;
- осуществление фильтрации данных на основании связи родитель-ребенок;
- поддержка различных манипуляций с данными;
- разграничение прав, поддержка различных СУБД.

В таблице 1.2 приведены результаты анализа.

Таблица 1.2

Сравнение инструментов администрирования для MySQL

Критерии сравнения	ПО			
	PHPMyAdmin	Adminer	HeidiSQL	SQLyog
Обзор баз данных, таблиц	Реализовано			
Открытие двух и более таблиц одновременно	Не реализовано			

Настройка положения и размеров таблиц	Не реализовано			
Визуализация связей родитель-ребенок	Не реализовано			
BLOB-поля	Не реализовано			Реализовано
Фильтрация записей на основании связей родитель-ребенок	Не реализовано			
Манипуляции с данными	Реализовано			
Разграничение прав	Реализовано			
Доступные СУБД	MySQL	MySQL, PostgreSQL, SQLite, MS SQL и Oracle.	MySQL, Microsoft SQL Server, PostgreSQL.	MySQL
ОС	WEB		Windows	

1.4.6 Заключение

- Во всех продуктах отсутствует возможность открытия двух и более таблиц одновременно. Они открываются либо в различных вкладках, либо попеременно в одном окне. Как следствие, отсутствует настройка их положения и размеров. Таким образом, оказывается невозможной одновременная работа с несколькими таблицами на одном экране;
- отсутствует визуализация связей родитель-ребенок между таблицами, а также фильтрация данных дочерних таблиц на основе этих связей;
- просматривать содержимое BLOB-полей можно лишь в программе SQLyog.

1.5 Обоснование выбора платформы, средств и инструментов для создания ПО

При разработке ПО для работы с базами данных MySQL в первую очередь необходимо ответить на вопрос, какое же решение будет наилучшим для поставленной задачи: разработка desktop или web-приложения.

Многие разработчики останавливают свой выбор на web-интерфейсах и логике их взаимодействия с базами данных. Это объясняется тем, что взаимодействовать с такими приложениями можно через браузер, который доступен на любом компьютере. Но закономерное стремление достичь универсальности приводит к усложнению системы, что негативно влияет на ее производительность. В скрипты вводится множество абстракций, появляются избыточные запросы к базам данных, часто код обрастает модулями, ненужными для решения конкретных практических задач. Развитие стандартов и возникновение новых технологий, таких как AJAX, позволяют частично устранить эти проблемы [6].

Часто куда более исчерпывающей и выигрышной альтернативой оказывается проектирование специализированного клиента. Абстракции программной архитектуры и сложные интерфейсы выносятся на уровень клиента, пользователю предоставляется максимум степеней свободы, способы управления функциональностью становятся «прозрачнее» и проще:

- изменения, внесенные в данные, обычно обособлены от серверной базы данных и не оказывают на нее никакого воздействия, пока пользователь не подтвердит результат;
- существуют стандартные элементы управления, позволяющие отображать данные базы в табличном формате, в настраиваемой сетке. К преимуществам представленного подхода можно отнести: легкость реализации таких базовых функций, как правка содержимого ячеек; одновременный просмотр значительных объемов табличных данных;
- благодаря высокому уровню абстракции разрабатываемый проект может быть полностью абстрагирован от языка серверной реализации и от определенной СУБД.

На основании вышеизложенного можно сделать следующие выводы:

- рекомендуется делать выбор в пользу web-приложений, если основной упор делается на доступность и оперативность, а требования к навыкам пользователей минимальны;
- специализированный клиент будет лучшим решением в том случае, если в приоритете гибкость системы, а одной из целей является обеспечение доступа к большим объемам структурированной информации и взаимосвязям между объектами.

Таким образом, выбор был сделан в пользу разработки desktop-приложения. Рассмотрим средства и инструменты для его реализации, проанализируем принцип их работы, раскроем преимущества и недостатки.

1.5.1 MySQL

Необходимость в использовании баз данных часто возникает при разработке программных проектов. Выбор существующих СУБД весьма обширен, однако практически каждый хостинг, предоставляющий стандартные услуги, способен взаимодействовать с MySQL.

MySQL – это реляционная СУБД, хранящая данные в виде логически связанных между собой таблиц, доступ к которым осуществляется с помощью языка запросов SQL. Является одной из самых популярных систем для серверов с высокой нагрузкой, которая задействована в огромном количестве web-приложений и сайтов. По причине широкого распространения для MySQL реализовано большое число плагинов и расширений [7].

СУБД MySQL включает в себя многопоточный SQL-сервер, который поддерживает клиентские программы и библиотеки, средства администрирования и широкий ассортимент программных интерфейсов приложений (API-интерфейсов). Сервер MySQL представлен и в виде встраиваемой многопоточной

библиотеки, которая может быть использована в автономных приложениях, обычно применяющихся в изолированной среде без доступа к сети.

Преимущества MySQL:

- поддерживает обширный SQL-функционал;
- имеет встроенные средства защиты, основанные на паролях и привилегиях. Трафик паролей во время соединения с сервером шифруется;
- позволяет обрабатывать большие объемы данных. Так, компания MySQL AB успешно использует сервер MySQL для администрирования базы, которая содержит пятьдесят миллионов записей;
- игнорирует некоторые SQL-стандарты, благодаря чему возрастает скорость работы и производительность.

Недостатки MySQL:

- не обладает некоторым функционалом, который может быть необходим в требовательных приложениях;
- несмотря на то, что MySQL является продуктом с открытым исходным текстом, его разработка практически остановлена.

Рекомендуется использовать СУБД MySQL при повышенных требованиях к безопасности (в том числе при обеспечении защиты доступа к данным), гибкости и производительности. Однако, если необходимо точно следовать стандартам SQL или использовать специфичный функционал, то применять для этих целей MySQL неразумно.

Некоторые отличия MySQL от других СУБД.

В MySQL вложенные подзапросы в большинстве случаев можно переписать либо в один сложный запрос, либо разбить на несколько запросов, тем самым выигрывая в быстродействии. В СУБД, где допускается применение вложенных подзапросов, программисты часто даже не задумываются о возможности их оптимизации. MySQL поддерживает альтернативный способ хранения таблиц (InnoDB), который не противоречит принципу транзакций, обеспечивая хорошо знакомые механизмы отката неудачных операций.

Если информация имеет специфичный уклон и плохо описывается в базовых терминах вроде чисел, дат и текста, то следует обратить внимание на другие СУБД. Например, если необходимо хранить и обрабатывать географическую информацию, то преимуществом будет обладать PostgreSQL. В ней присутствуют специализированные поля для хранения подобных данных и функции для их обработки. Но, в любом случае, для большинства проектов возможностей MySQL более чем достаточно.

1.5.2 Платформа .NET Framework

Microsoft создала множество средств, предусматривающих абстрагирование (в том числе и .NET Framework). Работая с ними, разработчики могут сконцентрировать внимание на своих задачах и не беспокоиться о технических нюансах, таких как характеристики ОС и аппаратного обеспечения.

Назначение платформы .NET Framework – предоставить пользователям единую модель программирования и API (см. рисунок 1.8). Такой подход упрощает интеграцию, тестирование, обслуживание и развертывание программных продуктов.

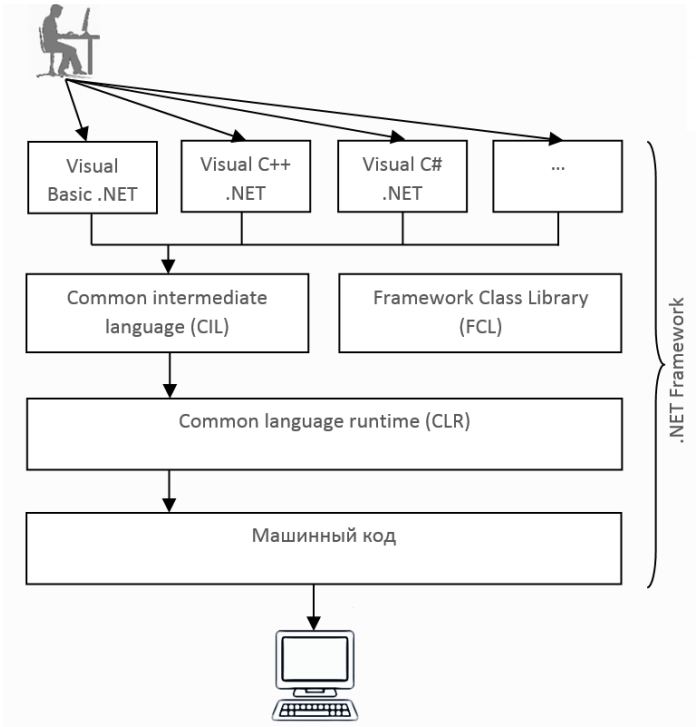

Рисунок 1.8 – Структура .NET Framework

.NET Framework включает в себя общеязыковую исполняющую среду (common language runtime, CLR) и библиотеку классов (Framework Class Library, FCL). CLR – это модель программирования, которая определяет поведение типов и объектов и взаимодействует с собственным загрузчиком файлов, диспетчером памяти, системой защиты доступа к коду, коллекцией потоков и т.д. В свою очередь, FCL является объектноориентированным API, используемым всеми моделями приложений [8]. Она содержит определения типов, позволяющие программистам выполнять ввод/вывод, работать с потоками, манипулировать строками и т. д.

Основные преимущества CLR:

- отсутствует потребность привязки к аппаратной архитектуре устройства. Компиляторы, включенные в Visual Studio, вместо обычного кода из процессорных команд генерируют код на общем промежуточном языке (common intermediate language, CIL). И уже при исполнении CLR преобразует CIL в команды процессора платформы, на которой выполняется приложение;
- CLR способна автоматически управлять памятью благодаря сборщику мусора (Garbage Collector), освобождающему память, которую занимают уже не использующиеся объекты. В ряде языков (например, в C/C++) необходимо самим отдавать команды и на создание, и на удаление объекта. Между тем рекомендуется избавлять программиста от этой работы, так как ошибки при

манипуляции с памятью очень трудно обнаружить, что обычно приводит к неверному выполнению программы в непредвиденный момент. CLR контролирует используемые ресурсы, а также решает другие подобные вопросы, в частности, отслеживает использование нетипизированных указателей;
- CLR обеспечивает корректное обращение к имеющимся типам: например, нельзя передать восьмибайтное значение четырехбайтному параметру. Также гарантируется передача управления только в заранее известные точки, т.е. невозможно принудить программу выполняться с произвольного адреса. Комплекс этих защитных средств предотвращает распространенные программные ошибки;
- предоставляется единый принцип обработки сбоев, что упрощает тестирование и сопровождение программ и избавляет от несогласованного стиля отчетов об ошибках;
- CLR поддерживает отладку мультиязычных приложений, в которых отдельные части программы написаны на различных языках.

Преимущества FCL:
- работая с классами одной библиотеки, различные языки имеют возможность взаимодействовать друг с другом. В классах задаются базовые типы, обеспечивая так называемый «каркас», на который проецируются типы языков программирования: например, целочисленный тип в Visual Basic называется integer, а в C# – int, однако оба отображаются на один и тот же каркас – System32.Int32. Библиотека включает и структурные типы, задающие организацию данных – строки, массивы и перечисления.

1.5.3 Язык программирования C# и среда разработки Visual Studio

Язык программирования C# был создан прежде всего для работы с платформой .NET Framework, поэтому он в полной мере учитывает все его возможности и в настоящее время является одним из самых мощных и востребованных языков в IT-сфере [9]. Отметим некоторые его особенности:
- C# поддерживает объектно-ориентированный подход, который позволяет проектировать масштабируемые, гибкие и расширяемые приложения;
- благодаря взаимодействию с библиотекой каркаса Framework .Net, можно достаточно легко извлекать и хранить информацию из баз данных;
- разрешает, но не поощряет явную работу с памятью (в частности с указателями, адресной арифметикой и т.д.), управление ею производится автоматически.

Visual Studio содержит обширный функционал, характерный для платформы .NET Framework. MVS предоставляет такие базовые компоненты, как редактор исходного текста, конструктор интерфейса, компиляторы, отладчики, компоновщики и т.д.

Одна из его особенностей заключается в том, что помимо включенных в него языков программирования (Visual C++ .Net, Visual C# .Net, J# .Net, Visual Basic

.Net), в среду можно добавлять любые другие, которые совместимы с каркасом Framework .Net. Этот подход сближает ЯП и делает возможным разработку нескольких фрагментов одного приложения на различных языках [10].

Однако несмотря на преимущества, работа с .NET имеет ряд недостатков. В первую очередь, требования к аппаратному обеспечению достаточно высоки: объем ОП должен быть не менее 256 Мб, а свободный объем жесткого диска для Visual Studio – минимум 10 Гб. Вторая проблема заключается в том, что ряд компиляторов для языков программирования представлены сторонними компаниями-разработчиками, результаты деятельности которых довольно сложно контролировать и дорабатывать [11].

1.5.4 ADO.NET и Connector/NET

1.5.4.1 Назначение и структура

Чтобы приложение могло взаимодействовать с СУБД, необходим посредник, с помощью которого осуществляется связь с базой. Одним из таких посредников является ADO.NET – технология работы с данными, основанная на платформе .NET Framework. Она включает в себя классы, выполняющие подключение к БД вне зависимости от ее структуры, места расположения и деталей реализации.

ADO.NET работает с базами посредством ряда объектов: Connection, Command, DataReader, DataSet и DataAdapter.

- Connection отвечает за подключение к источнику данных;
- с помощью Command формируются запросы к базе и запускаются хранимые на сервере процедуры;
- DataReader предназначен для считывания данных;
- DataSet представляет собой контейнер, который хранит полученную в результате запроса информацию и позволяет оперировать ею независимо от базы;
- DataAdapter – промежуточное звено между объектом DataSet и источником данных.

Схематично структуру ADO.NET можно представить следующим образом (см. рисунок 1.9):

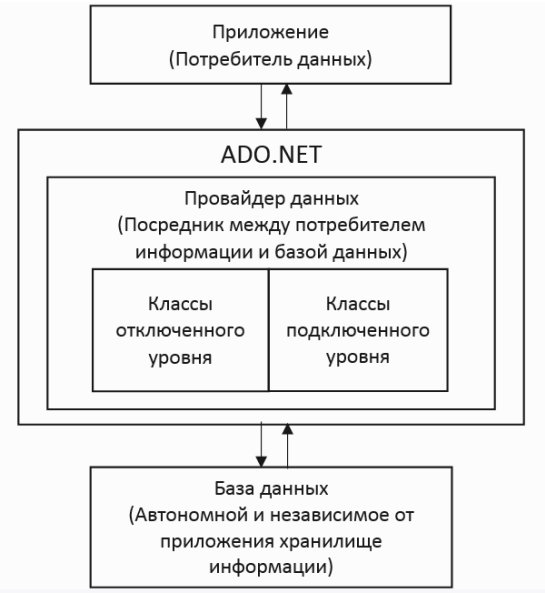

Рисунок 1.9 – Структура ADO.NET

1.5.4.2 Поставщики данных (провайдеры)

Поставщик данных (data provider) представляют собой специальный набор классов, позволяющий взаимодействовать с конкретной СУБД. В ADO.NET для каждого источника данных существует свой провайдер, который и определяет реализацию вышеуказанных классов. Имена основных классов отличаются у различных поставщиков (например, SqlConnection, OracleConnection, OdbcConnection и MySqlConnection), но все они порождены от одного базового класса (см. рисунок 1.10).

Object	SQL Server	OLE DB	ODBC
Connection	SqlConnection	OleDbConnection	OdbcConnection
Command	SqlCommand	OleDbCommand	OdbcCommand
Data reader	SqlDataReader	OleDbDataReader	OdbcDataReader
Data adapter	SqlDataAdapter	OleDbDataAdapter	OdbcDataAdapter

Рисунок 1.10 – Префиксы, которые указывают на провайдера

В ADO.NET встроены следующие поставщики:

- провайдер для MS SQL Server, который работает по протоколу TDS, ориентированный специально на MS SQL Server. С помощью него повышается скорость передачи данных и производительность приложения;
- провайдер для OLE DB. Он предоставляет доступ к ряду СУБД: Access, MySQL, Oracle, DB2. Однако универсальность обеспечивается за счет производительности, поэтому рекомендуется использовать специализированные поставщики, если это возможно;

- провайдер для ODBC предназначен для тех источников, у которых отсутствует собственные поставщики;
- провайдеры для Oracle, EntityClient, SQL Server Compact.

Помимо вышеуказанных провайдеров есть множество других, в том числе и для СУБД MySQL.

1.5.4.3 Доступ к отсоединенным данным

До недавних пор, чтобы обеспечивать доступ к данным требовалось постоянное соединение с источником. Минусы такого подхода стали очевидны после возникновения приложений, ориентированных на интернет. Непрерывное соединение требует использования системных ресурсов, что становится проблемой в том случае, если большое количество клиентов работает с базой, расположенной на сервере.

Поэтому ADO.NET предлагает иную модель доступа. Соединение устанавливается на тот период времени, который нужен для выполнения операции над базой. Таким образом, ADO.NET делится на два уровня: подключенный, который устанавливает соединение и управляет базой со стороны клиента, и отключенный уровень, обеспечивающий хранение полученной информации (см. рисунок 1.11).

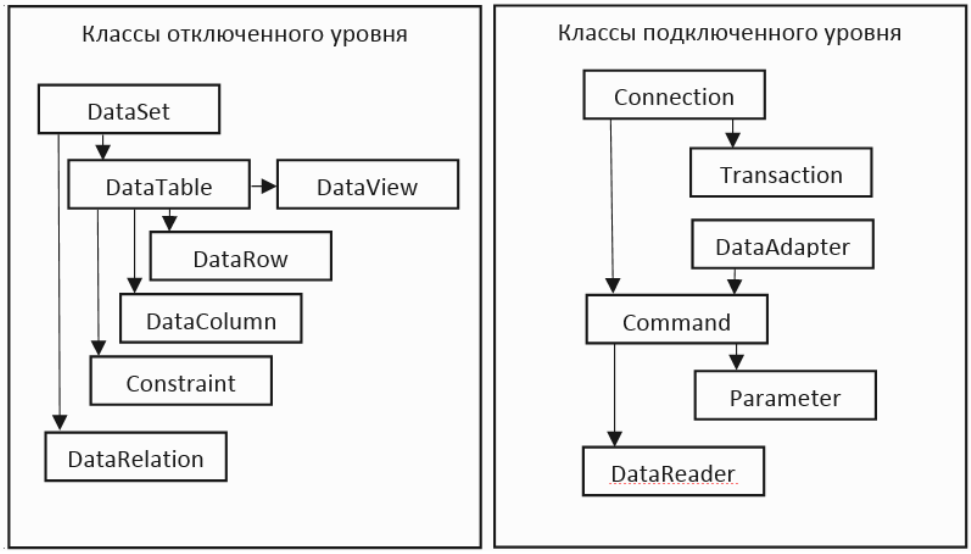

Рисунок 1.11 – Уровни ADO.NET

1.5.4.4 Достоинства ADO.NET

- ADO.NET позволяет создавать обобщенный код для взаимодействия с источником данных, так как каждый провайдер использует одни и те же базовые классы;
- специализированные поставщики для конкретных СУБД гарантируют повышение производительности и скорости доступа.

1.5.4.5 Connector/NET

Есть несколько вариантов организации доступа к MySQL с помощью технологии ADO.NET. Разработчики Windows-приложений часто выбирают между провайдерами Connector/ODBC и Connector/NET (см. рисунок 1.12).

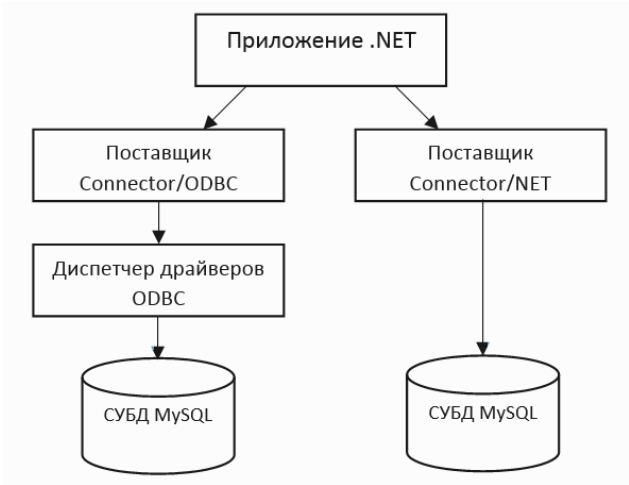

Рисунок 1.12 – Принцип организации доступа с помощью Connector/ODBC и Connector/NET

В Connector/ODBC за взаимодействие с каждой отдельной СУБД отвечает свой ODBC-драйвер. Приложение подключается к диспетчеру драйверов, а он, в свою очередь, предоставляет соответствующий ODBC-драйвер для доступа к источнику. Этот подход позволяет работать практически с любой системой, обеспечивая тем самым универсальность, однако существует ряд недостатков:

- во-первых, увеличивается время обработки запросов;
- во-вторых, на каждом рабочем месте требуется установка и настройка ODBC-драйвера, причем параметры настроек (СУБД, путь к серверу, база данных и прочее) приложение изменить самостоятельно не может.

Основное назначение провайдера ODBC – обеспечивать доступ к тем СУБД, для которых нет «родного» поставщика. Лучше отдать предпочтение поставщику, который специально предназначен для взаимодействия с конкретным источником, так как в этом случае учитываются его особенности. Таким провайдером для MySQL является Connector/NET.

Connector/NET реализуют непосредственное обращение к функциям клиентского API MySQL, что обеспечивает превосходную производительность по сравнению с Connector/ODBC. Connector/NET поддерживается любым языком, который могут использовать поставщики ADO.NET: C#, VB.NET, Delphi и т.д.

1.5.5 Web-сервер

Web-сервер – место хранения страниц вебсайта вместе с его базами данных и программными модулями. При разработке и тестировании ПО для работы с

MySQL возникает потребность в сервере, который организовывал бы доступ к сайту и его базам.

1.5.5.1 Apache

На серверах хост-провайдеров размещаются готовые функционирующие сайты с информационным наполнением. Однако в процессе создания сайта предпочтительней работать с локальным сервером, позволяющим проверять «сырые» проекты и не беспокоиться об оплате хостинга на этапе разработки. Локальный сервер в полном объеме воспроизводит функции сервера хост-провайдера и состоит из тех же компонентов: СУБД MySQL, скриптов для взаимодействия с базами, PHP-парсера и т.д.

Web-сервер Apache – это серверное приложение, которое реализует взаимодействие с ресурсами по HTTP-протоколу и работает на многих ОС, в частности в Windows, Mac и Linux. Apache является довольно распространенным web-сервером, который установлен более чем на половине хостов, чему поспособствовали такие его преимущества, как бесплатная лицензия, кроссплатфоменность, открытость кода и высокий уровень безопасности [12].

1.5.5.2 Сборки XAMPP и Denwer

Чтобы сделать из домашнего ПК полноценный web-сервер и разместить в локальной сети сайты и базы данных, нет нужды в инсталляции и настройке Apache, MySQL и других пакетов. Есть более простой и быстрый способ – установить XAMPP или Denwer. Они представляют собой готовые сборки, включающие в себя нужные компоненты для управления сайтами на локальном ПК. К ним относятся Apache, обработчик языка PHP, СУБД MySQL и др.

На данный момент Denwer – самая популярная сборка для ОС Windows, и, в отличие от XAMPP, она ориентирована на русскоязычную работу с PHP и MySQL. XAMPP в свою очередь имеет большую популярность в связи с кроссплатформенностью (поддерживается на Windows, Linux, Mac и других системах). Отличается простотой установки/удаления, и оперативными выпусками версий по мере обновления составляющих [13]. Есть возможность развернуть полноценный web-сервер, необходимо лишь настроить XAMPP для работы через интернет. К сожалению, Denwer не обладает такими функциями.

Таким образом, в качестве локального виртуального хостинга XAMPP превосходит другие аналогичные средства почти по всем параметрам: он прост и удобен в установке, настройке и управлении, отличается экономичным потреблением ресурсов компьютера, и не требует специализированных знаний у пользователя в области серверных технологий.

1.6 Выводы по разделу

В данном разделе были проанализированы требования к разрабатываемому ПО, в том числе требования к функциональным характеристикам, режиму

функционирования, интерфейсу, надёжности и условиям эксплуатации. Кроме того, были описаны перспективы модернизации и развития продукта.

Был проведен обзор существующих решений для работы с данными таблиц MySQL, описаны их достоинства и недостатки. На основе этого был выполнен сравнительный анализ рассмотренных средств и разрабатываемого ПО и выявлены основные функциональные преимущества этого ПО.

Было приведено обоснование выбора инструментов для разработки приложения, в частности:

- выбор был сделан в пользу разработки desktop-приложения с применением платформы .NET Framework, предлагающей множество способов взаимодействия с базами данных, в том числе собственную технологию доступа к данным – ADO.NET;
- при написании кода будет использоваться язык программирования C#, который поддерживает объектно-ориентированный подход и учитывает все возможности каркаса Framework .NET;
- для реализации взаимодействия с СУБД MySQL будет использоваться провайдер Connector/NET, а для организации доступа к сайту и его базам – web-сервер Apache.

2 МОДЕЛЬ СИСТЕМЫ

2.1 Введение

Основные функциональные возможности, такие как просмотр информации и удаление/редактирование/добавление записей, реализованы для всех типов таблиц MySQL. Фильтрация данных применяется только к таблицам типов MyISAM и InnoDB, которые требуют использование различных подходов для определения внешних ключей.

Необходимо уточнить, что при реализации приложения не применялся стандартный объект DataSet, который является автономной версией схемы базы данных. Обычно его используют для программного представления отношений родитель-ребенок между таблицами, имитирующих ограничения внешнего ключа. Отказ от использования DataSet обоснован следующей причиной. При реализации приложения планируется, что в память будут загружаться только те таблицы, которые открывает пользователь, и, наоборот, при их закрытии память будет очищаться. Но отношения между таблицами и ограничения столбцов можно в полной мере эмулировать с помощью DataSet только в том случае, если в него будут загружены локальные копии сразу всех таблиц [14]. Таким образом, ПО будет самостоятельно контролировать связи между таблицами и выполнять проверку на наличие нарушений ограничений.

Использование ПО будет иметь смысл в том случае, если таблицы приведены как минимум ко второй нормальной форме, а именно:

- таблица должна находиться в первой нормальной форме (все строки в таблице различны, значения внутри ячеек атомарны);
- любое поле таблицы, не входящее в состав первичного ключа, должно функционально полно зависеть от первичного ключа.

2.2 Определение первичных и внешних ключей

Основными типами таблиц в СУБД MySQL являются MyISAM и InnoDB. Основные отличия этих двух типов таблиц приведены в таблице 2.1:

Таблица 2.1

Сравнение таблиц типов MyISAM и InnoDB

Характеристика	MyISAM	InnoDB
Производительность	Быстрое выполнение запросов SELECT и INSERT. С другой стороны, нет поддержки блокировки на уровне строк, из-за чего на таблицу допускается только одна одновременная DELETE- или UPDATE-операция. При	Медленное выполнение INSERT-операций. При преобладании операции чтения работает медленнее, но при смешанной нагрузке (SELECT/UPDATE/INSERT/DELETE) работает быстрее.

Продолжение таблицы 2.1

	больших объемах данных это приводит к серьезным проблемам.	
Блокировка	Блокировка на уровне таблиц	Блокировка на уровне строк
Хранение данных	Каждой таблице отдельный файл	Данные при настройках по умолчанию хранятся в больших совместно используемых файлах
Надежность и восстановление	Отсутствие возможности самовосстановления по журналу при сбоях.	Обеспечивается надежное хранение данных за счет транзакций и блокирования данных на уровне строки. Обеспечивается быстрое самовосстановление после сбоев.
Транзакции и внешние ключи	Не поддерживаются	Поддерживаются
Полнотекстовый поиск	Поддерживается	Не поддерживается (доступен начиная с версии MySQL 5.6.4)
Объем таблиц	Таблицы более компактные	Без сжатия наблюдается увеличение объема таблиц

Из таблицы видно, что MyISAM не поддерживает внешние ключи, поэтому для того, чтобы ПО могло их определить самостоятельно, необходимо пойти на некоторые соглашения: внешним ключом считается поле, которое удовлетворяет следующим условиям:

- оно не является первичным ключом;
- существует таблица, которая содержит поле с таким же именем и типом, являющееся первичным ключом.

Внешние ключи для таблиц InnoDB, а также первичные ключи для таблиц обоих типов определяются с помощью базы данных INFORMATION_SCHEMA, обеспечивающей доступ к метаданным MySQL.

2.3 База данных INFORMATION_SCHEMA

INFORMATION_SCHEMA – это информационная база данных (системный каталог), место, которое сохраняет информацию относительно всех других баз данных, которые поддерживает сервер MySQL [15]. Структура этой базы приведена на рисунке 2.1:

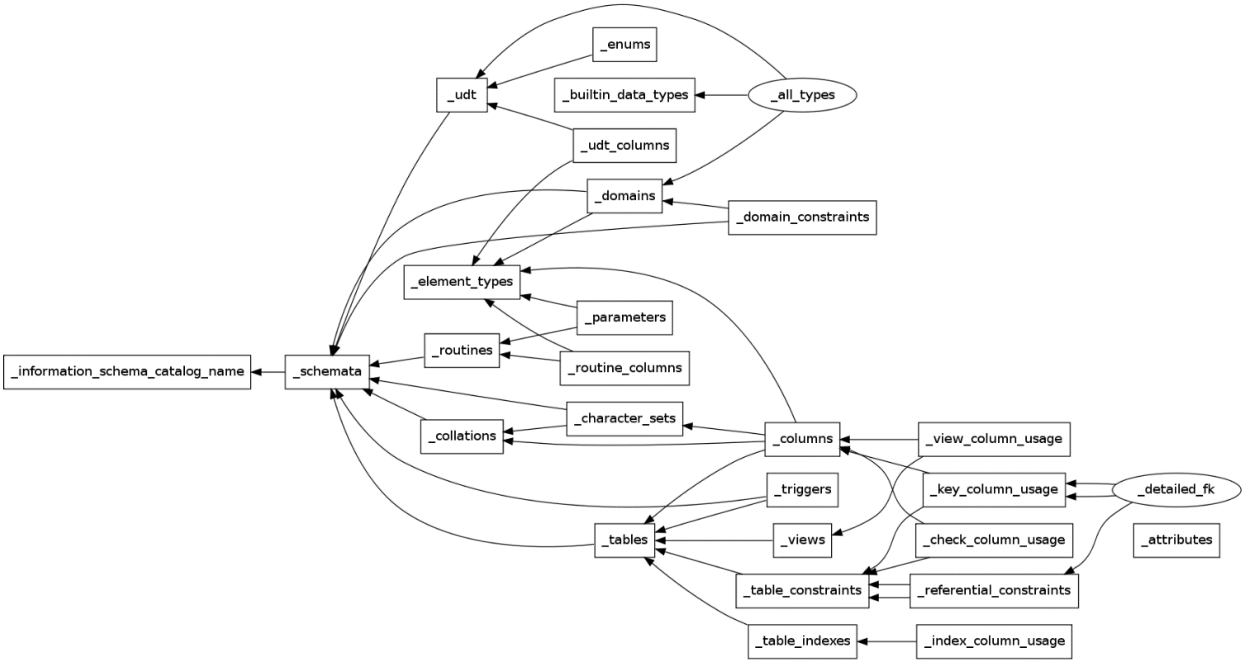

Рисунок 2.1 – Структура системного каталога INFORMATION_SCHEMA

С помощью INFORMATION_SCHEMA разрабатываемое программное обеспечение может получать информацию о базах данных, таблицах, их типах и ключах. Для этого используются следующие инструкции:

- получение имен всех содержащихся на сервере баз данных с помощью таблицы schemata:

```
SELECT schema_name
FROM INFORMATION_SCHEMA.schemata
```

где schema_name – наименование атрибута, содержащего названия баз данных.

- Извлечение названий таблиц конкретной базы данных с помощью запроса к таблице tables:

```
SELECT table_name
FROM INFORMATION_SCHEMA.tables
WHERE table_schema = 'db_name'
```

где table_name – наименование атрибута, содержащего названия таблиц, 'db_name' – название интересующей базы данных.

Также в tables хранятся данные о типах таблиц:

```
SELECT engine
FROM INFORMATION_SCHEMA.tables
WHERE table_schema = 'db_name' AND table_name = 'table_name'
```

здесь engine – наименование атрибута, содержащего тип таблицы, 'db_name' и 'table_name' – названия базы и таблицы соответственно.

- Определение внешних ключей таблиц типа InnoDB с помощью таблицы key_column_usage, которая описывает столбцы, имеющие ограничения:

```
SELECT column_name, referenced_table_name, referenced_column_name
FROM INFORMATION_SCHEMA.key_column_usage
WHERE table_schema = 'db_name' AND table_name = 'table_name' AND
constraint_name <> 'primary' AND referenced_table_name is not null
```

здесь column_name – название столбца, содержащего внешний ключ, referenced_table_name – имя родительской таблицы, referenced_column_name – имя столбца родительской таблицы, на который ссылается внешний ключ, constraint_name – название ограничения.

- Получение информации о первичных ключах таблиц посредством таблицы columns:

```
SELECT column_name
FROM INFORMATION_SCHEMA.columns
WHERE table_schema = 'db_name' AND table_name = 'table_name' AND column_key = 'PRI'
```

где column_name – название столбца, содержащего первичный ключ.

2.4 Поддержание ограничений целостности

Как уже было сказано раннее, система должна самостоятельно контролировать, чтобы в процессе редактирования, удаления или вставки новой записи не происходило нарушения ссылочной целостности.

Для таблиц, которые поддерживают транзакции и внешние ключи (в MySQL это таблицы типа InnoDB), такая проверка происходит на уровне сервера баз данных. В других случаях требуется предварительно проверять данные, прежде чем выполнить запрос к базе. Если они удовлетворяют всем ограничениям, то выполняется соединение с базой и вносятся необходимые изменения. В противном случае пользователь получает уведомление о возникшей проблеме и выполненные изменения отменяются.

Перечень основных проверок, которые были реализованы с целью корректного функционирования системы:

- контроль данных при вставке новой строки. Значение внешнего ключа должно соответствовать какому-либо из значений первичного ключа родительской таблицы, или равняться NULL, если это допустимо;
- проверка данных при модификации строки. В случае изменения внешнего ключа, его новое значение должно совпадать с каким-либо из значений первичного ключа родительской таблицы, или равняться NULL, если это допустимо. В случае изменения первичного ключа и успешного выполнения запроса к БД необходимо заменить внешние ключи дочерних таблиц на соответствующие новые значения;
- выяснение правомерности удаления записи. При удалении строки родителя нужно перебрать дочерние таблицы и выяснить, содержат ли они внешние ключи, совпадающие с первичным ключом удаляемой записи. В этом случае операция не проводится.

Также с помощью стандартного объекта DataTable обеспечиваются основные проверки, такие как:

- проверка на допустимость нулевых значений. Операция вставки или модификации отменяется, если столбец не может содержать NULL-значения;
- проверка типа данных. В случае, если новое значение имеет неверный формат, действие не выполняется;

- проверка на допустимость изменения значения (read only). Некоторые поля доступны только для чтения, в этом случае система не разрешит их изменять;
- проверка на уникальность. Если поле имеет ограничение UNIQUE, то система запретит вставлять значение-дубликат.

2.5 Выводы по разделу

Для того, чтобы разрабатываемое программное обеспечение отвечало всем основным функциональным требованиям, были реализованы:

- взаимодействие с системным каталогом INFORMATION_SCHEMA для извлечения информации о структуре баз данных и их таблиц;
- определение внешних и первичных ключей для таблиц MyISAM и InnoDB с целью установления связи родитель-ребенок между таблицами;
- контроль ограничений и ссылочной целостности при удалении, модификации и добавлении записей.

3 РАЗРАБОТКА АРХИТЕКТУРЫ СИСТЕМЫ

3.1 Диаграмма вариантов использования

Диаграмма использования (use-case diagram) позволяет получить общее представление о функциональном назначении системы [16].

На рисунке 3.1 представлена диаграмма, описывающая основное взаимодействие между пользователем и системой. На рисунке 3.2 приведено уточнение диаграммы для прецедента «Открыть таблицу».

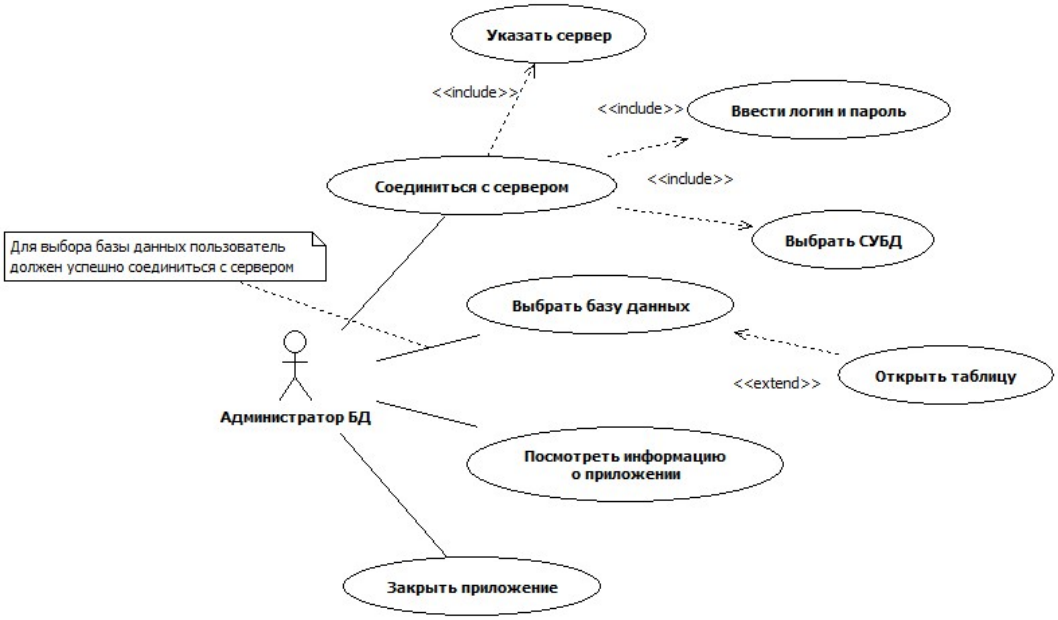

Рисунок 3.1 – Диаграмма вариантов использования системы

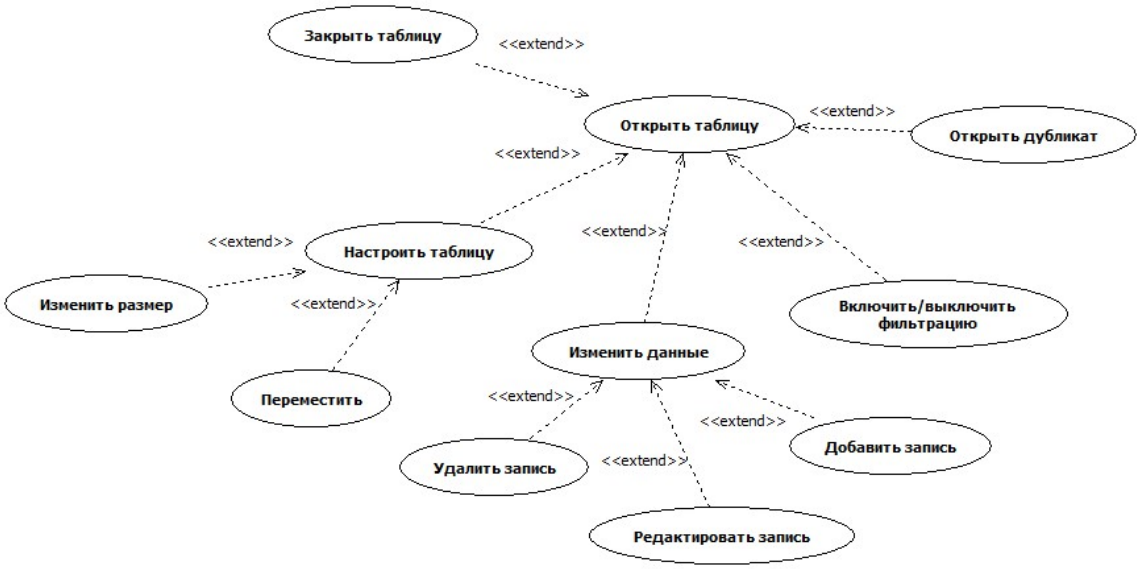

Рисунок 3.2 – Диаграмма вариантов использования прецедента «Открыть таблицу»

Далее приводится словесное описание (сценарий) для каждого прецедента.

3.1.1 Use-case: «Вход в систему/соединение с сервером»

Пользователь (администратор) может получить доступ к хранящимся на сервере базам данных, указав необходимые сведения для установки соединения.

Главная последовательность:

Пользователь вводит сервер, логин и пароль в соответствующие формы, выбирает СУБД из предоставленного списка и подтверждает ввод. Система соединяется с сервером, используя указанные данные, и запускает главное окно программы.

Альтернативная последовательность 1:

Ввод неверных данных. Система выдает сообщение об ошибке доступа и предлагает повторить ввод.

Альтернативная последовательность 2:

Отсутствие соединения с сервером. Система сообщает о невозможности установить связь с сервером и предлагает ввести новые данные.

Альтернативная последовательность 3:

Пользователь закрывает окно авторизации или нажимает кнопку «Отмена». Система не выполняет соединение с сервером и закрывает приложение.

3.1.2 Use-case: «Выбрать базу данных»

После успешного соединения с сервером администратор может выбрать любую базу данных, чтобы впоследствии взаимодействовать с ее таблицами.

Главная последовательность:

Пользователь находит в списке БД нужную ему базу и щелкает по ней левой кнопкой мыши. Система разворачивает подсписок таблиц выбранной базы.

3.1.3 Use-case: «Посмотреть информацию о приложении» и «Закрыть приложение»

В любой момент пользователь может получить информацию о программе или прервать работу с приложением, закрыв его.

3.1.4 Use-case: «Открыть таблицу»

Развернув подсписок таблиц конкретной базы данных, администратор может открыть таблицы для последующего взаимодействия с ними.

Главная последовательность:

Администратор щелкает правой кнопкой мыши по названию таблицы и выбирает пункт контекстного меню «Открыть». Система отправляет запрос к базе и получает все записи открываемой таблицы. Затем система создает структуру, имитирующую табличную форму, и наполняет ее данными.

Альтернативная последовательность 1:

Отсутствие соединения с сервером. Система выдает сообщение об отсутствии связи с сервером; таблица не открывается.

Альтернативная последовательность 2:

Во процессе открытия на сервере была удалена база или открываемая таблица. В этом случае система сообщает об ошибке; таблица не открывается.

3.1.5 Use-case: «Изменить данные»

В режиме редактирования таблицы администратор может удалять, добавлять или править записи. Система контролирует корректность действий пользователя и правильность введенных данных.

Главная последовательность:

Администратор изменяет, добавляет либо удаляет активную запись таблицы. Система отправляет запрос к базе и обновляет данные в форме.

Альтернативная последовательность 1:

Отсутствие соединения с сервером. Система выдает сообщение об отсутствии связи с сервером; данные в форме не обновляются.

Альтернативная последовательность 2:

На сервере была удалена база или таблица, с которой пользователь взаимодействует посредством приложения. Система сообщает об отсутствии на сервере базы/таблицы; данные в форме не обновляются.

Альтернативная последовательность 3:

Администратор вводит неверный формат данных или его действия нарушают целостность хранимых данных. Система предупреждает об ошибке, не посылает запрос к базе и не обновляет форму.

Альтернативная последовательность 4:

База данных не позволяет выполнить запрос. Система сообщает об ошибке и не обновляет данные в форме.

3.1.6 Use-case: «Настроить таблицу»

После того, как администратор успешно открыл таблицу, он сможет ее настроить.

Главная последовательность:

Пользователь указывает удобное местоположение таблицы и оптимальные размеры.

3.1.7 Use-case: «Включить/выключить фильтрацию данных»

В случае, если открытая таблица ссылается на другую таблицу (между ними существует связь «родитель-ребенок»), администратор может включить режим фильтрации данных для дочерней таблицы (или выключить, если он активен).

Главная последовательность 1:

Пользователь щелкает правой кнопкой мыши по заголовку таблицы и выбирает в контекстном меню те родительские таблицы, по которым должна

производиться фильтрация. Система посылает ряд запросов к базе данных и заполняет дочернюю таблицу данными, удовлетворяющими активным записям в выбранных открытых таблицах.

Дочерняя таблица заполняется всеми данными в следующих случаях:

- в контекстном меню ни одна родительская таблица не выбрана;
- ни одна из выбранных родительских таблиц не открыта.

Альтернативная последовательность 1:

Соединение с сервером было потеряно. В этом случае система предупреждает пользователя об отсутствии соединения, и, соответственно, не отправляет запросы и не обновляет таблицу.

Альтернативная последовательность 2:

На сервере была удалена база данных, с которой работает пользователь, или таблица, в которой производится фильтрация. Система оповещает пользователя об этой проблеме и не обновляет таблицу.

Альтернативная последовательность 3:

На сервере были удалены родительские таблицы, по которым производится фильтрация. Система оповещает пользователя об этой проблеме и пытается выполнить фильтрацию только по тем таблицам, которые не были удалены.

3.1.8 Use-case: «Открыть дубликат»

Если таблица ссылается на себя же (связь типа «петля»), то для нее можно открыть таблицу-дубликат. Она копирует структуру и данные оригинала, но:

- дубликат имеет только одну дочернюю таблицу – таблицу-оригинал;
- для дубликата нельзя открыть еще один дубликат.

Предназначение такой таблицы заключается в обеспечении фильтрации для оригинала. Она выступает в роли родительской таблицы, по записям которой будет выполняться фильтрация данных в таблице-оригинале.

Главная последовательность:

Администратор щелкает правой кнопкой мыши по заголовку таблицы и выбирает пункт контекстного меню «Открыть дубликат». Система создает структуру, имитирующую табличную форму, и наполняет ее данными оригинала.

Альтернативная последовательность 1:

Отсутствие соединения с сервером. Система выдает сообщение об отсутствии связи с сервером; дубликат не открывается.

Альтернативная последовательность 2:

Во процессе открытия на сервере была удалена база или открываемая таблица. В этом случае система сообщает об ошибке; дубликат не открывается.

3.1.9 Use-case: «Закрыть таблицу»

В любой момент пользователь может закрыть таблицу.

Главная последовательность:

Пользователь щелкает правой кнопкой мыши по заголовку таблицы и выбирает в контекстном меню пункт «Закрыть». Система уничтожает форму и очищает память от хранимых данных.

3.2 Диаграмма классов

На рисунке 3.4 представлена логическая модель системы в виде диаграммы классов. Диаграмма отражает взаимосвязи между основными классами системы, а также описывает их внутреннюю структуру и типы отношений.

В данной работе обеспечивается взаимодействие с MySQL, однако благодаря некоторым архитектурным решениям существует возможность расширения ПО для работы с другими форматами баз данных. Для этих целей код, который отвечает за взаимодействие с базами данных, был размещен в отдельном специальном модуле. Это позволяет обособить основную логику приложения от типа используемой СУБД [17].

В общем интерфейсе были объявлены методы, необходимые для корректной работы приложения (перечень основных методов приведен ниже). Разработчик может определить производные классы, отвечающие за взаимодействие с конкретными СУБД, которые бы учитывали все нюансы при построении запросов и работали со своими поставщиками данных (см. рисунок 3.3).

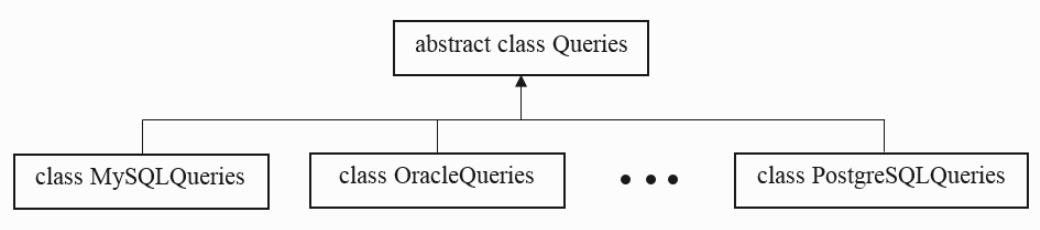

Рисунок 3.3 – Интерфейс и производные классы

Интерфейс содержит следующие методы:

- установка соединения с сервером баз данных;
- поиск баз данных и их таблиц;
- заполнение таблицы всеми данными;
- заполнение таблицы данными, удовлетворяющими заданному фильтру;
- определение первичных и внешних ключей;
- операции обновления, вставки и удаления записи.

Разработчики могут определять и дополнительные методы в производном классе. В частности, подкласс, реализующий взаимодействие с MySQL, содержит два вспомогательных метода, каждый из которых определяет внешние ключи для своего типа таблиц (InnoDB и MyISAM).

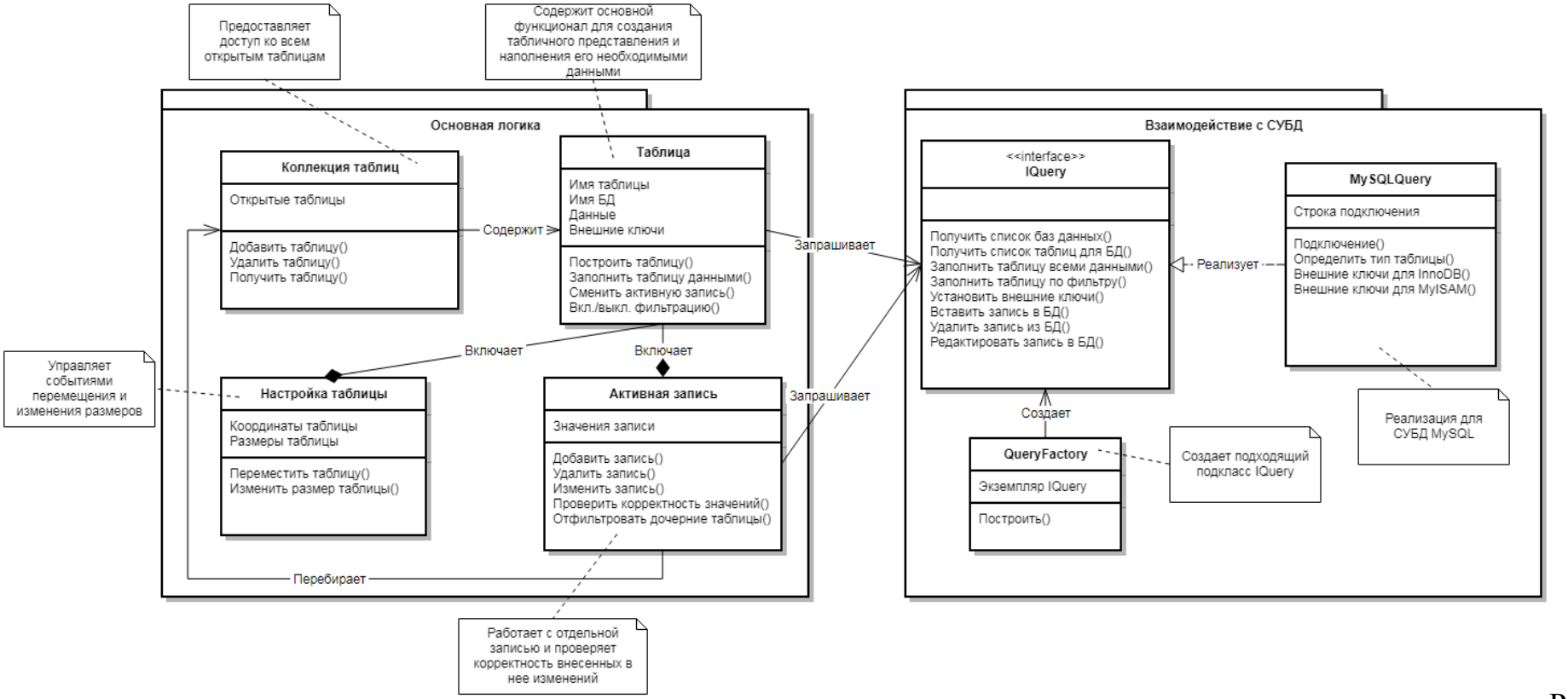

Рисунок 3.4 – Диаграмма классов

Для того, чтобы система оставалась независимой от типов порождаемых объектов (в данном случае – от СУБД), был реализован паттерн «Простая фабрика». Это класс, содержащий метод с большим условным оператором, создающим подходящий экземпляр. Методу передается некоторый параметр, который однозначно указывает, какой из экземпляров необходимо создать [18].

3.3 Выводы по разделу

В данном разделе были описаны:

- диаграмма использования, предоставляющая информацию о функциональном назначении системы; для каждого прецедента были приведены подробные сценарии;
- диаграмма классов, иллюстрирующая взаимосвязи между классами системы и описывающая их внутреннюю структуру.

4 РАЗРАБОТКА ИНТЕРФЕЙСА СИСТЕМЫ

Интерфейс разрабатываемого ПО представлен на рисунках 4.1, 4.3 и 4.5. На рисунках 4.2 и 4.4 указаны основные параметры, участвующие в вычислениях.

4.1 Главное окно приложения

Рассмотрим подробно основные элементы главного окна (см. рисунок 4.1).

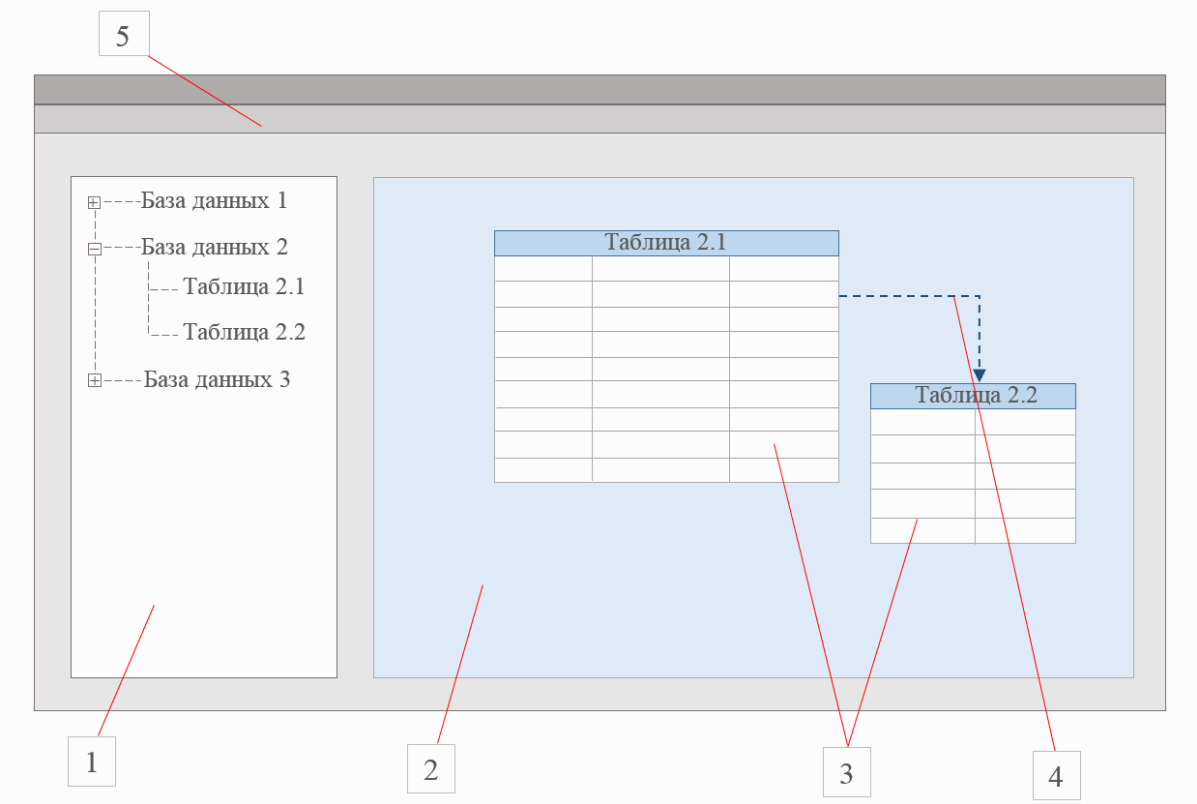

Рисунок 4.1 – Интерфейс главного окна

1. Панель, содержащая древовидный список доступных баз данных и таблиц. При щелчке правой кнопкой мыши по названию таблицы вызывается контекстное меню, позволяющее открыть элемент 3. Средствами Visual Studio осуществляется с помощью компонента TreeView.
2. Панель для открытых таблиц. К данному элементу привязано контекстное меню, с помощью которого можно закрыть все открытые таблицы. В среде Visual Studio для создания подобных элементов используют контейнер Panel.
3. Открытые таблицы, отображающие хранимые в них данные. Таблицу можно закрывать, перемещать в пределах панели, изменять ее размеры, а записи – добавлять, редактировать и удалять. Такие объекты были программно реализованы с помощью комбинации компонентов Panel, Label и DataGridView.
4. Отображение связи между таблицами.
5. Главное меню программы.

Размеры главного окна программы зависят от разрешения экрана монитора, на котором было запущено приложение. Предполагается, что наименьшее разрешение – 800x600 пикселей. Параметры основных компонентов (см. рисунок 4.2) рассчитываются по нижеприведенным формулам.

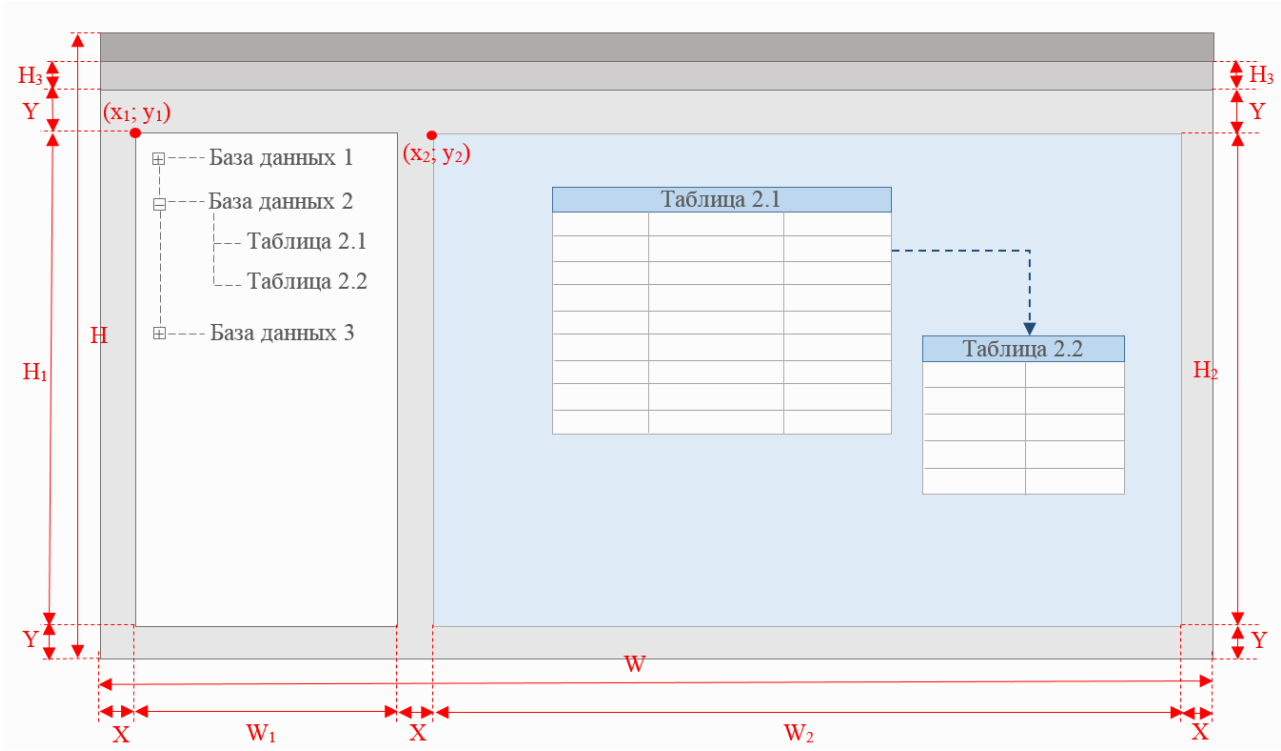

Рисунок 4.2 – Параметры компонентов главного окна

Пусть W и H – ширина и высота окна приложения соответственно. W_1, H_1 – параметры элемента 1 (список баз данных), W_2, H_2 – параметры элемента 2 (панель для таблиц), а H_3 – параметры элемента 5 (главное меню).

$$\begin{cases} W_1 - \text{фиксированная ширина} \\ H_1 = H - (H_3 + 2 * Y) \end{cases}, \begin{cases} W_2 = W - (W_1 + 3 * X) \\ H_2 = H - (H_3 + 2 * Y) \end{cases}.$$

Пусть $(\overset{x}{¿¿1}; y_1)$ – координаты верхнего левого угла первого элемента, позиция которого зафиксирована и не зависит от размеров окна. Тогда $(\overset{x}{¿¿2}; y_2)$ – координаты расположения второго элемента, которые будут определяться так:

$$(\overset{x}{¿¿2}; y_2) = (W - W_2 - x_1; (H - H_3 - H_2)/2).$$

4.2 Таблица для отображения данных

Таблица состоит из 4 элементов (см. рисунок 4.3): Panel, двух Label и DataGridView.

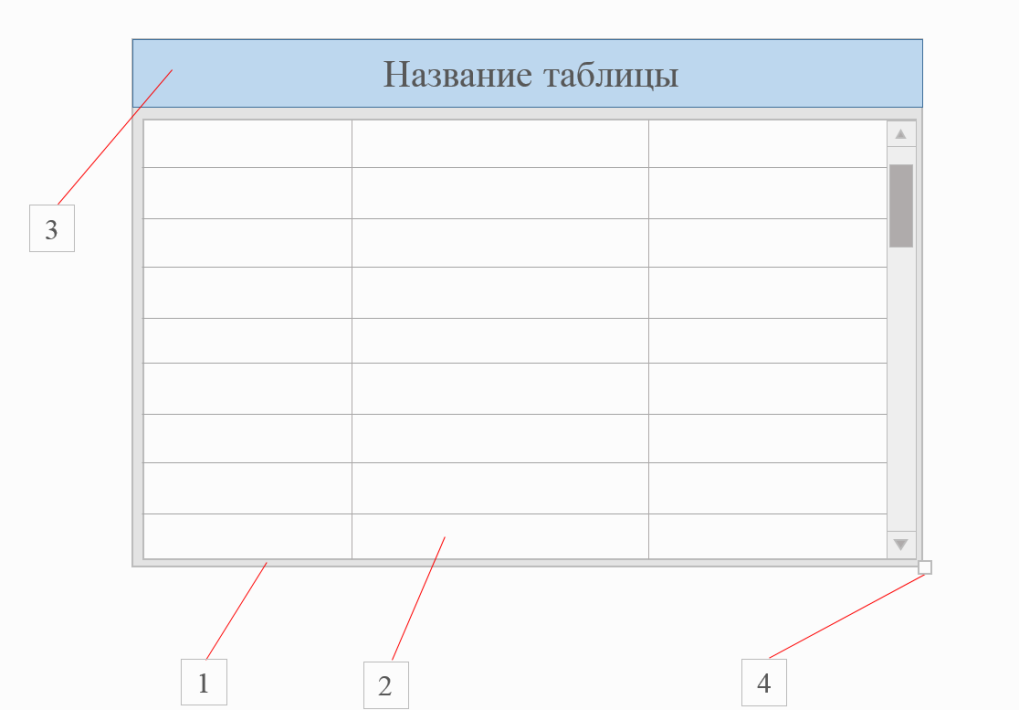

Рисунок 4.3 – Интерфейс таблицы

1. Роль Panel заключается в группировке остальных компонентов с целью формирования единого цельного элемента.
2. DataGridView представляет собой сетку, отображающую данные.
3. В первом Label хранится название таблицы. К нему привязано событие перемещения и контекстное меню, с помощью которого можно закрыть таблицу или включить фильтрацию.
4. Второй Label отвечает за изменение размера.

Максимальный размер панели-таблицы не превышает половины от высоты и ширины панели для отображения таблиц ($W_{панели}$, $H_{панели}$):

$$\begin{cases} 0 \le W \le W_{панели}/2 \\ 0 \le H \le H_{панели}/2 \end{cases} \qquad (1)$$

Пусть $W_{табл}$ и $H_{табл}$ – предполагаемая ширина и высота dataGridView. Это вспомогательные переменные, которые будут участвовать в расчетах:

$$\begin{cases} W_{табл} = \sum_{i=0}^{i<N} w_i \\ H_{табл} = \sum_{i=0}^{i<M} h_i + h_{заголовка} \end{cases};$$

здесь N и M – количество столбцов и строк соответственно. Тогда параметры панели-таблицы и других компонентов (см. рисунок 4.4) рассчитываются следующим образом (с учетом ограничения (1)):

$$\begin{cases} W = W_{табл}, если\ W_{табл} \le W_{панели}/2 \\ W = \frac{W_{панели}}{2}, если\ W_{табл} > W_{панели}/2 \end{cases},$$

$$\begin{cases} H = H_{табл} + H_1, если\ H_{табл} \le H_{панели}/2 + H_1 \\ H = \frac{H_{панели}}{2}, если\ H_{табл} > H_{панели}/2 + H_1 \end{cases},$$

$$\begin{cases} W_2 = W \\ H_2 = H - H_1 \end{cases},$$

$$\begin{cases} W_3 = W \\ H_3 - фиксированная\ высота \end{cases}.$$

Рисунок 4.4 – Параметры таблицы

4.3 Связи между таблицами

Отображение связи между таблицами выполнено в виде стрелки, направленной от родительской таблицы к дочерней (см. рисунок 4.5).

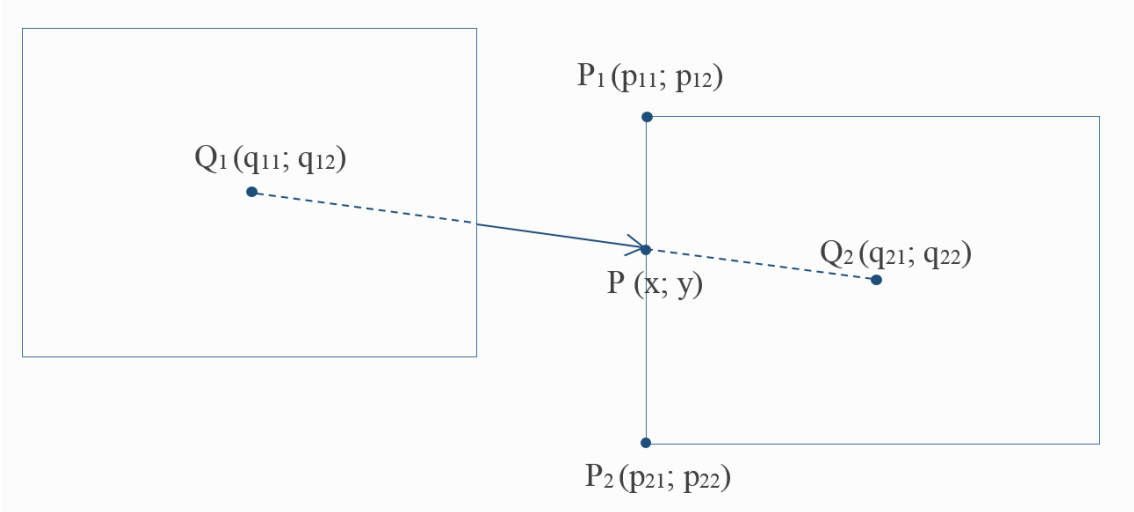

Рисунок 4.5 – Связь между родительской и дочерней таблицами

Пусть H_i и W_i – высота и ширина i-ой таблицы соответственно, $(¿¿i^{x}_{¿}; y_i)$ – координаты ее верхнего левого угла. Тогда координаты центра i-ой таблицы вычисляются следующим образом:

$$(q_{i1}; q_{i2}) = (x_i + \frac{W_i}{2}; y_i + \frac{H_i}{2}) \quad .$$

Пусть Q_1Q_2 – отрезок, соединяющий центры родительской и дочерней таблиц и пересекающий одно из ребер P_1P_2 дочерней таблицы в некоторой точке P. Ребро P_1P_2 определяется одной из нижеприведенных пар координат:

$$\begin{cases} (p_{11}; p_{12}) = (x_2; y_2); (p_{21}; p_{22}) = (x_2; y_2 + H_2) \\ (p_{11}; p_{12}) = (x_2; y_2); (p_{21}; p_{22}) = (x_2 + W_2; y_2) \\ (p_{11}; p_{12}) = (x_2 + W_2; y_2); (p_{21}; p_{22}) = (x_2 + W_2; y_2 + H_2) \\ (p_{11}; p_{12}) = (x_2 + W_2; y_2 + H_2); (p_{21}; p_{22}) = (x_2; y_2 + H_2) \end{cases}$$

Тогда координаты точки P будут решением системы

$$\begin{cases} A_1 * x + B_1 * y + C_1 = 0 \\ A_2 * x + B_2 * y + C_2 = 0 \end{cases},$$

где

$$A_1 = q_{22} - q_{12}, B_1 = q_{11} - q_{21}, C_1 = q_{12} * q_{21} - q_{11} * q_{22} \ ;$$
$$A_2 = p_{22} - p_{12}, B_2 = p_{11} - p_{21}, C_2 = p_{12} * p_{21} - p_{11} * p_{22} \ ,$$

откуда

$$P(x, y) = \left(\frac{C_2 * B_1 - C_1 * B_2}{A_1 * B_2 - A_2 * B_1}; \frac{A_2 * C_1 - A_1 * C_2}{A_1 * B_2 - A_2 * B_1} \right).$$

$\overrightarrow{n_{Q_1 P}}$ и $\overrightarrow{a_{Q_1 P}}$ – нормальный и направленный векторы отрезка Q_1P (см. рисунок 4.6).

Рисунок 4.6 – Параметры стрелки

$$x - q_{11}$$
$$¿$$
$$¿ 2 - (y - q_{12})^2$$
$$¿$$
$$x - q_{11}$$
$$¿$$
$$¿ 2 - (y - q_{12})^2$$
$$(¿¿¿);$$
$$¿$$
$$\sqrt{¿}$$
$$\frac{x - q_{11}}{¿}$$
$$\overrightarrow{a_{Q_1 P}}(a_x, a_y) = ¿$$

$$x-q_{11}$$
¿
¿
$$¿2-\left(y-q_{12}\right)^2$$
¿
$$x-q_{11}$$
¿
¿
$$¿2-\left(y-q_{12}\right)^2$$
$$(¿¿¿).$$
¿
$$\sqrt{¿}$$
$$\frac{q_{12}-y}{¿}$$
$$\overrightarrow{n_{Q_1P}}\left(n_x,n_y\right)=¿$$

С их помощью можно найти точки L, O и M:

$$O\left(o_x,o_y\right)=\left(x-w*a_x;y-w*a_y\right),$$
$$L\left(l_x,l_y\right)=\left(o_x+h*n_x;o_y+h*n_y\right),$$
$$M\left(m_x,m_y\right)=\left(o_x-h*n_x;o_y-h*n_y\right).$$

4.4 Окно авторизации

Элементы окна авторизации реализованы с помощью стандартных компонентов управления, таких как Button, TextBox и ComboBox. Рассмотрим подробно основные элементы окна авторизации (см. рисунок 4.7).

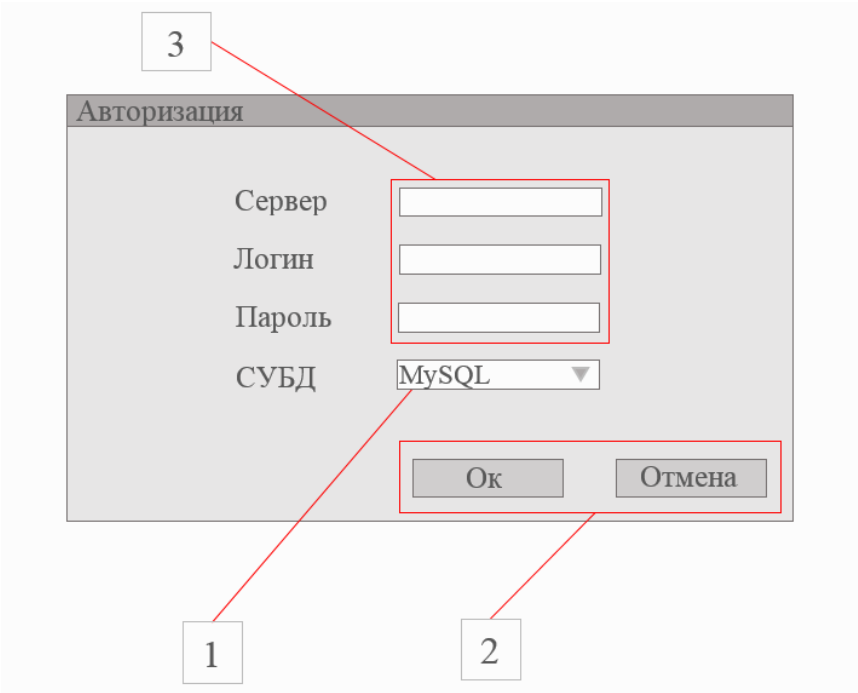

Рисунок 4.7 – Интерфейс окна авторизации

1. Выпадающий список, позволяющий пользователю выбрать СУБД, с которой он планирует работать.
2. Кнопки. С помощью кнопки «Ок» пользователь подтверждает введенные данные, после чего проводится его аутентификация. Кнопка «Отмена» закрывает окно и выходит из приложения.
3. Поля для ввода идентификационных данных пользователя.

4.5 Выводы по разделу

В данном разделе:

- разработан интерфейс программного обеспечения;
- рассмотрены назначение его компонентов и способы их реализации средствами Visual Studio;
- предложена методика расчетов геометрии окон приложения и их элементов.

5 РЕАЛИЗАЦИЯ СИСТЕМЫ

5.1 Разработка алгоритмов

Далее приводится описание алгоритмов работы программы.

5.1.1 Основной алгоритм программы

На рисунке 5.1 приведен основной алгоритм программы, на рисунке 5.2 – алгоритм, уточняющий подпроцесс «Обработка Mes для окна W_i».

Рисунок 5.1 – Основной алгоритм программы

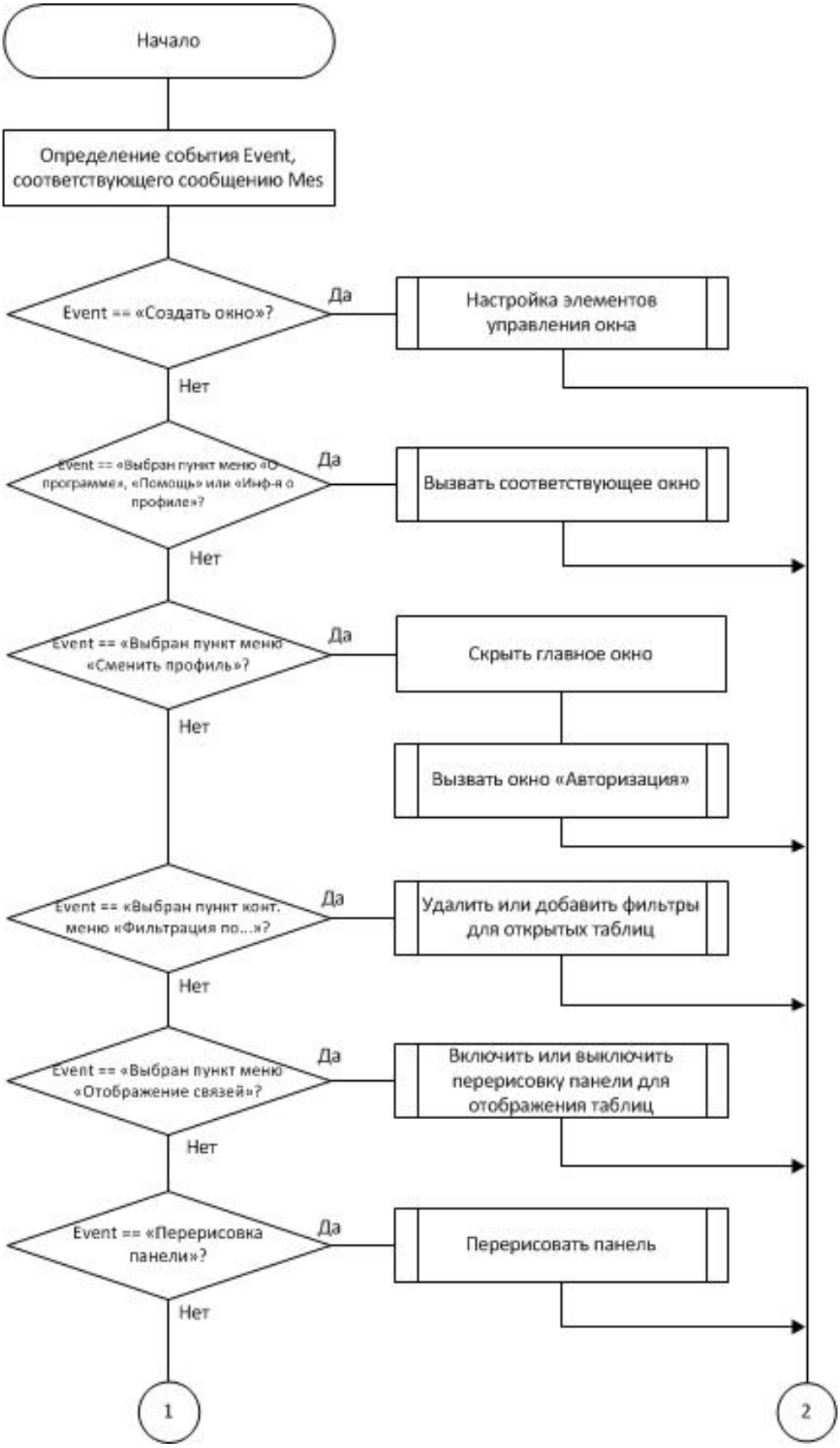

Рисунок 5.2 – Алгоритм обработки Mes для окна W_i

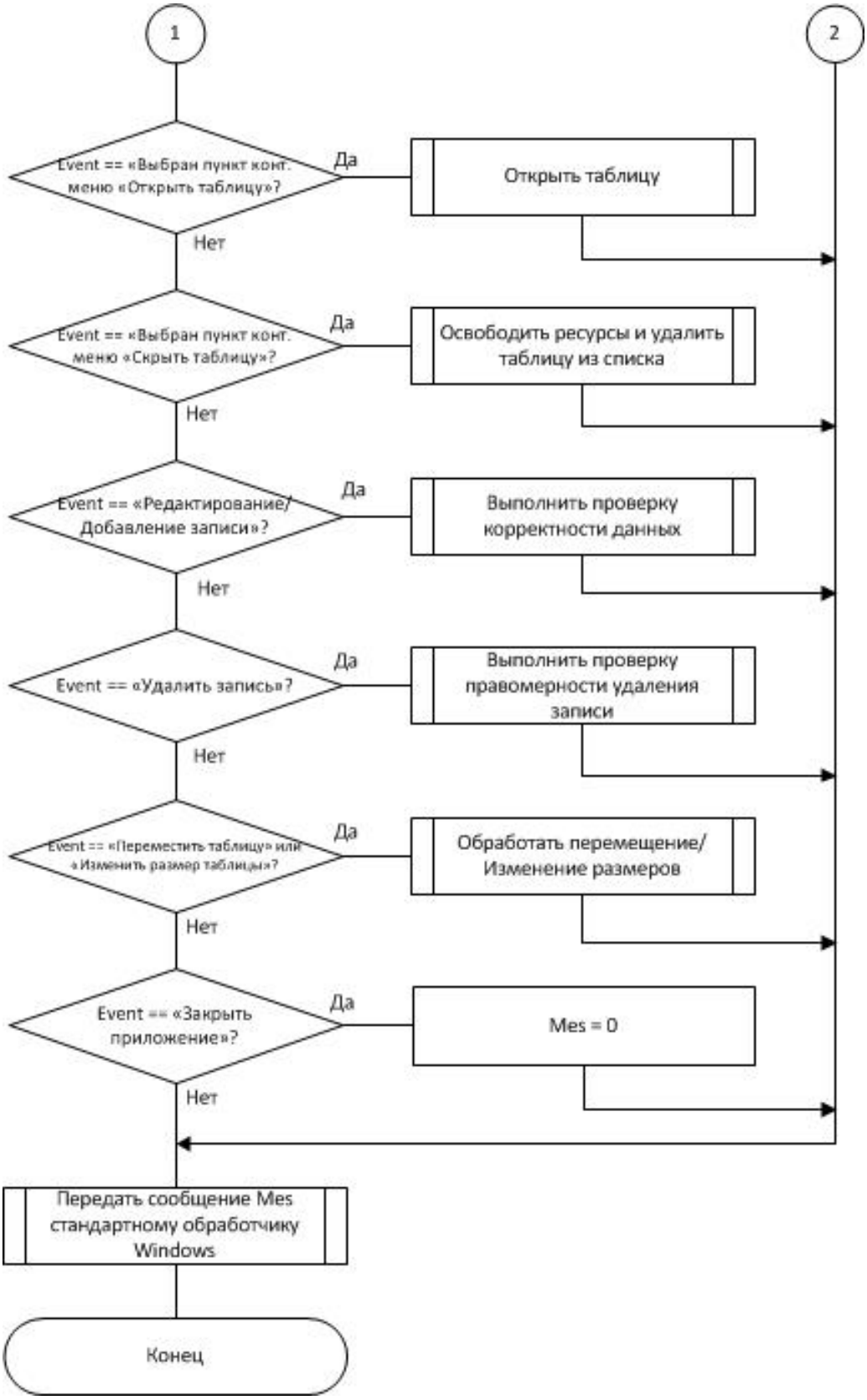

Рисунок 5.2 – Окончание

5.1.2 Алгоритм открытия таблицы

На рисунке 5.3 изображен алгоритм открытия новой таблицы, который начинает выполняться, если пользователь в контекстном меню выбрал пункт «Открыть таблицу».

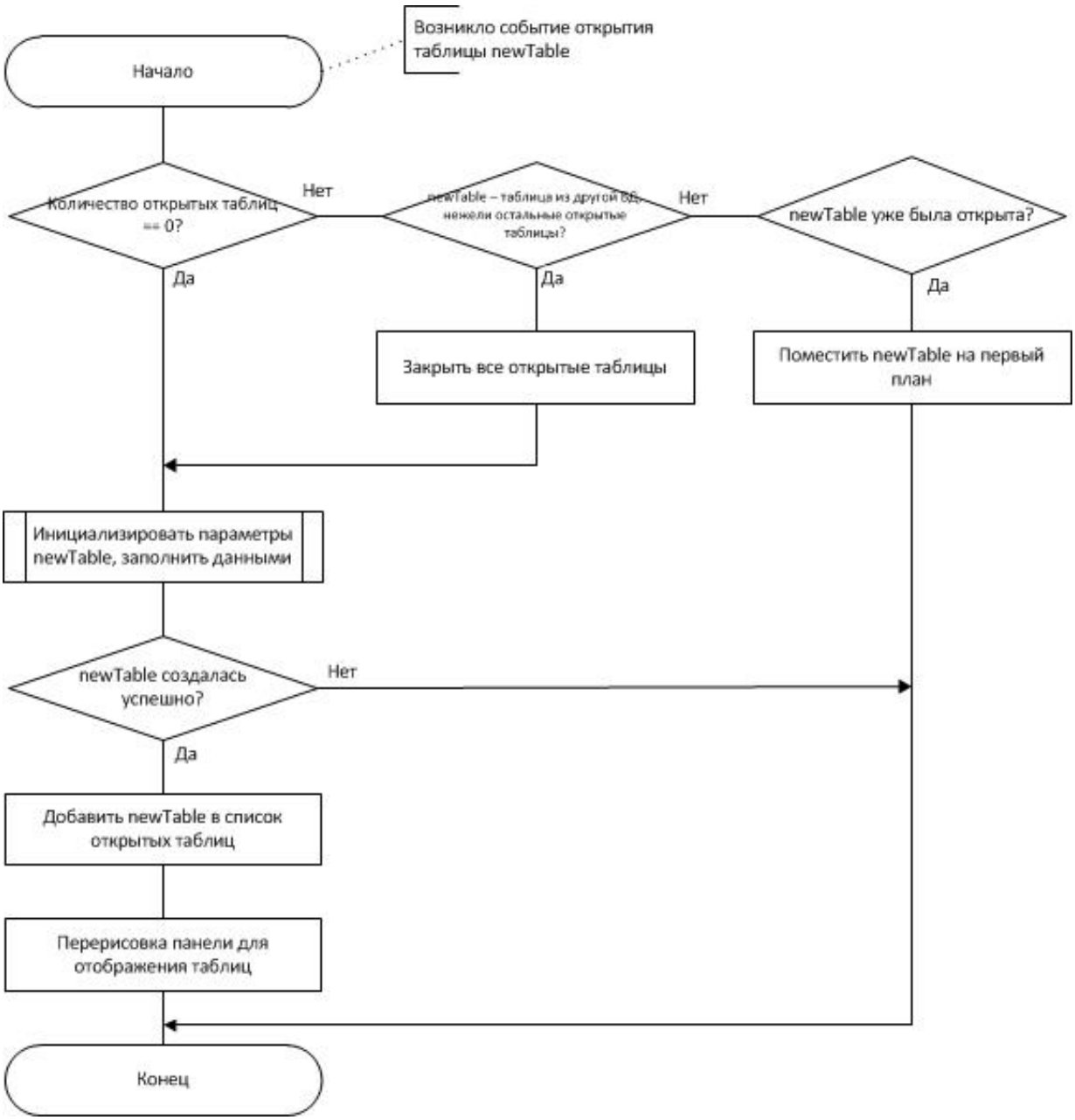

Рисунок 5.3 – Алгоритм открытия таблицы

5.1.3 Алгоритмы определения внешних ключей

На рисунке 5.4 приведен основной алгоритм поиска внешних ключей, на рисунке 5.5 – алгоритм, уточняющий подпроцесс «Поиск внешних ключей для MyISAM».

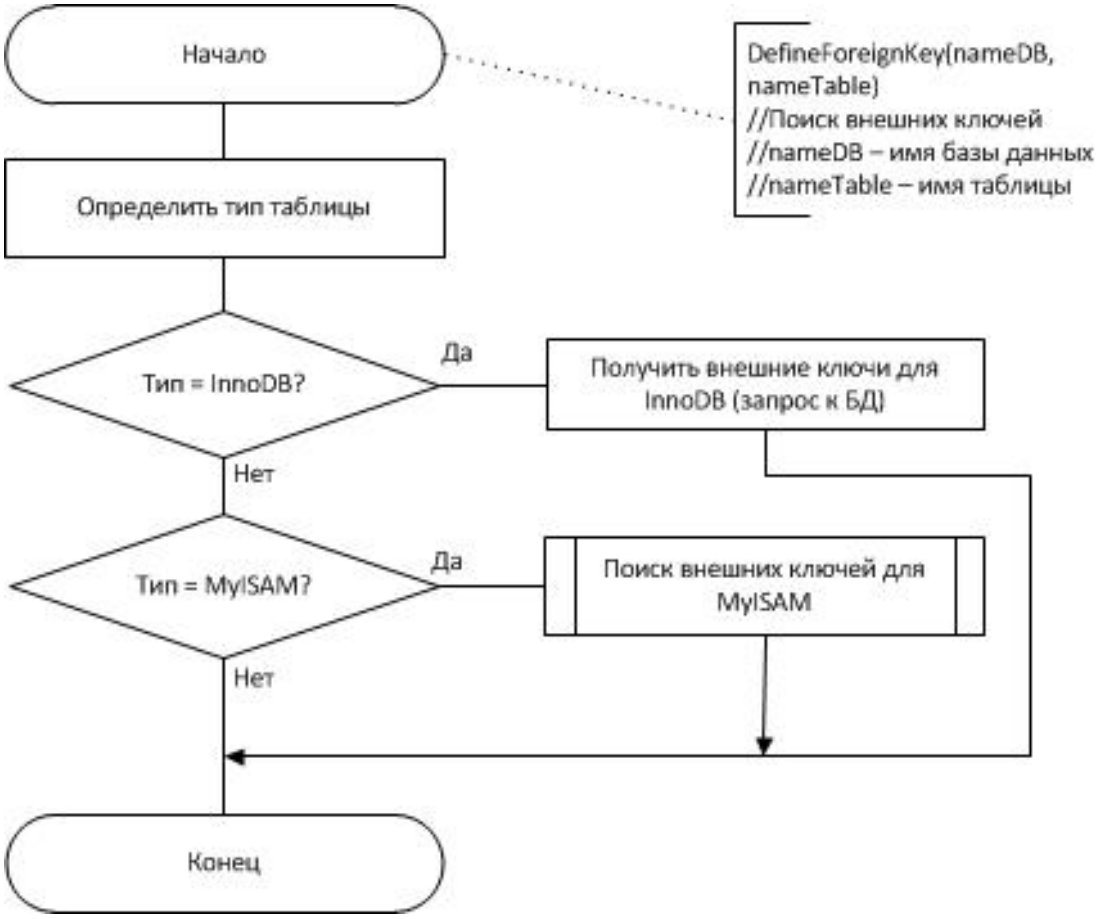

Рисунок 5.4 – Основной алгоритм поиска внешних ключей

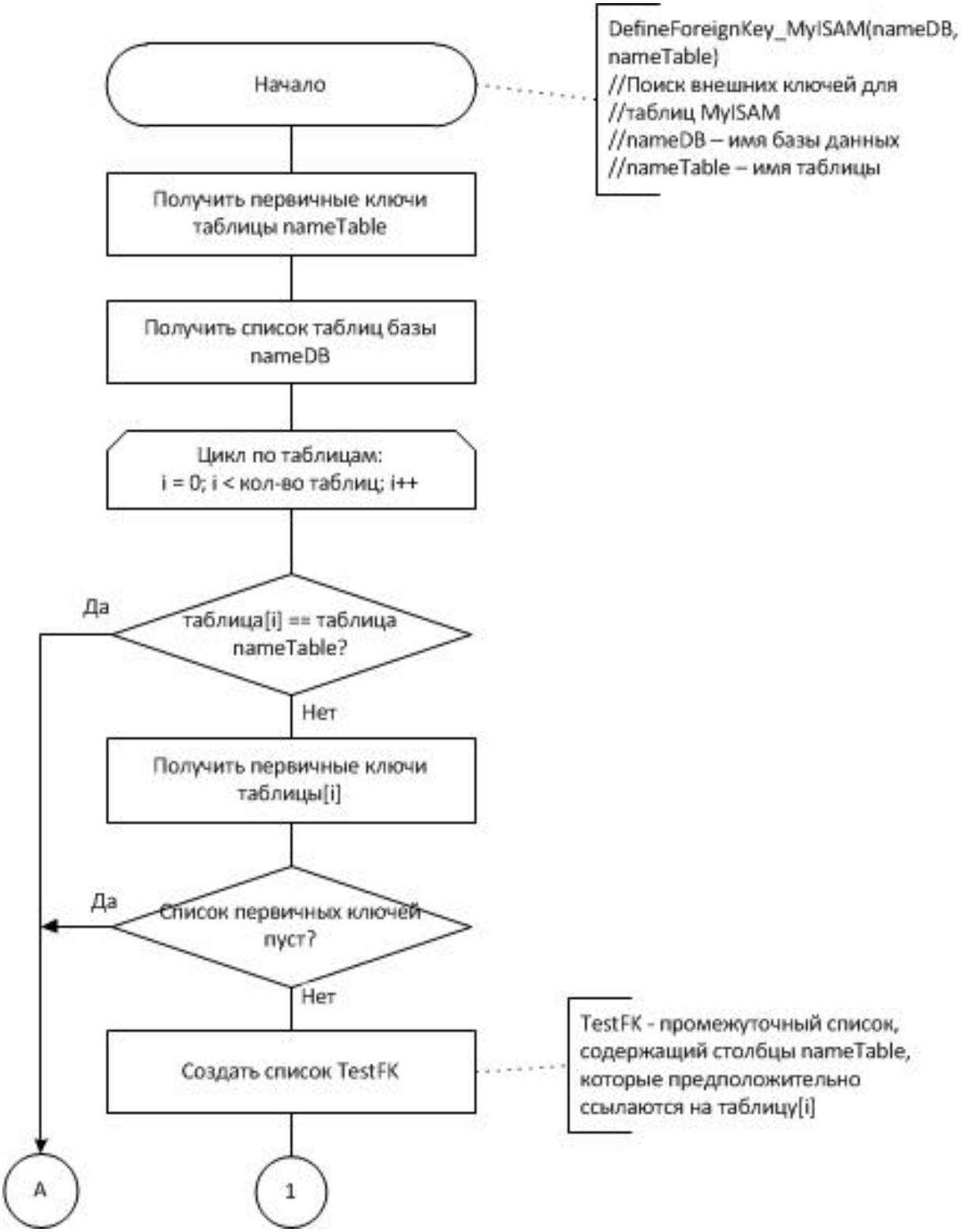

Рисунок 5.5 – Алгоритм поиска внешних ключей для таблиц MyISAM


1
Цикл по первичным ключам таблицы [i]:
j = 0; j < кол-во перв. ключей; j++
Цикл по столбцам таблицы nameTable:
k = 0; k < кол-во столбцов; k++
Имя и тип столбца[k] и ключа [j] совпадают?
Нет
Да
Столбец[k] не является первичным ключом и не состоит в другой связи?
Нет
Да
Добавить столбец[k] в список TestFK
Цикл по столбцам таблицы nameTable
Цикл по первичным ключам таблицы [i]
Кол-во ключей в TestFK == количеству первичных ключей таблицы[i]?
Нет
Да
Создать связь родитель-ребенок между таблицей[i] и nameTable
A
Цикл по таблицам
Конец

Рисунок 5.5 – Окончание

5.1.4 Алгоритмы фильтрации дочерних таблиц

На рисунках 5.6-5.7 предоставлены алгоритмы фильтрации дочерних таблиц.

При открытии родительской таблицы (или смене активной записи в ней) нужно пройтись по всем ее дочерним открытым таблицам (в которых включена фильтрация по данному родителю) и «отфильтровать» их содержание: другими словами, послать запрос в БД на выборку записей, в которых значения внешнего ключа совпадают со значениями первичного ключа родительской таблицы.

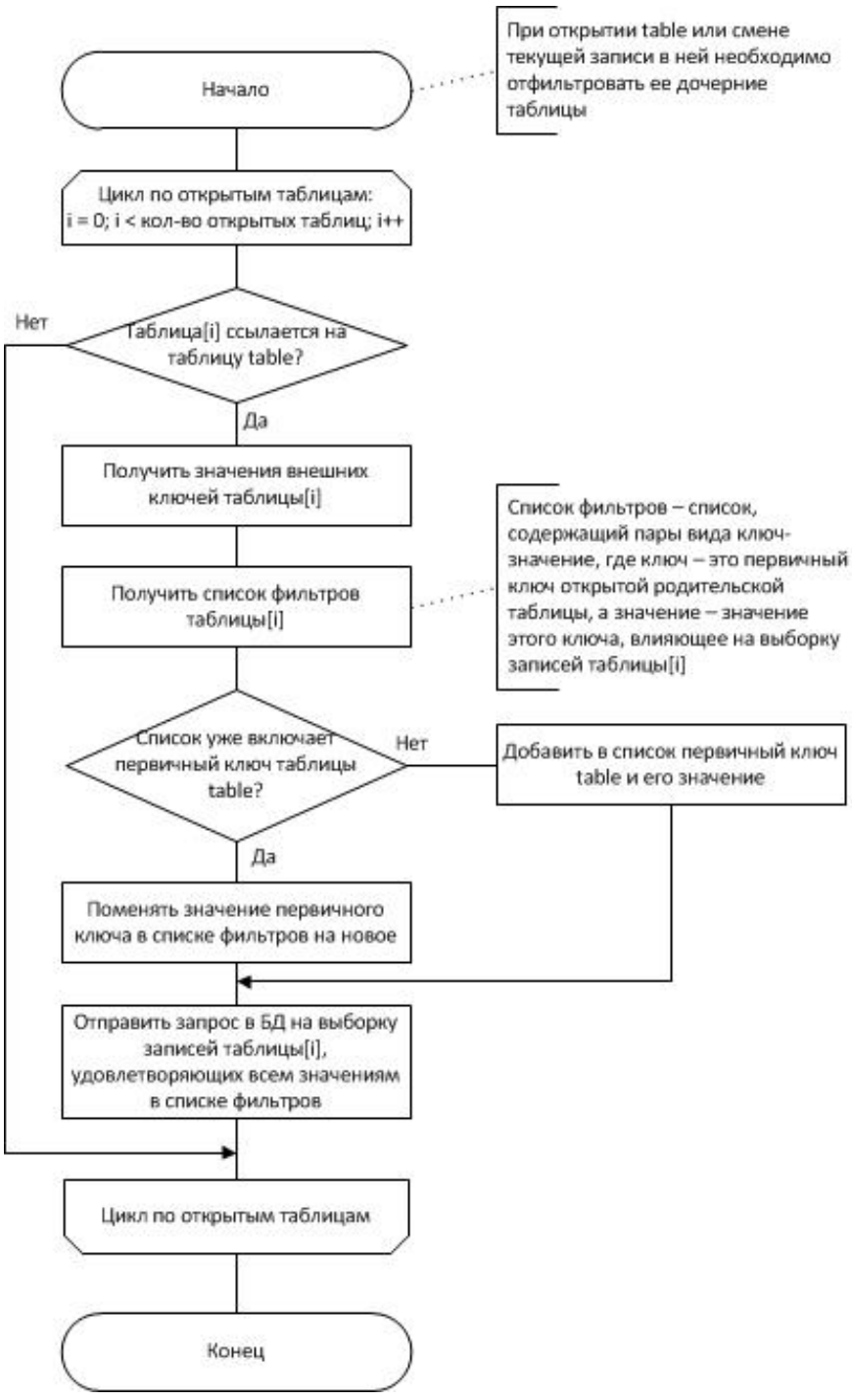

Рисунок 5.6 – Алгоритм фильтрации дочерних таблиц, добавление нового фильтра

При закрытии таблицы необходимо обеспечить, чтобы значения ее первичного ключа больше не оказывали влияния на содержание тех дочерних таблиц, в которых включена фильтрация по данной родительской таблице.

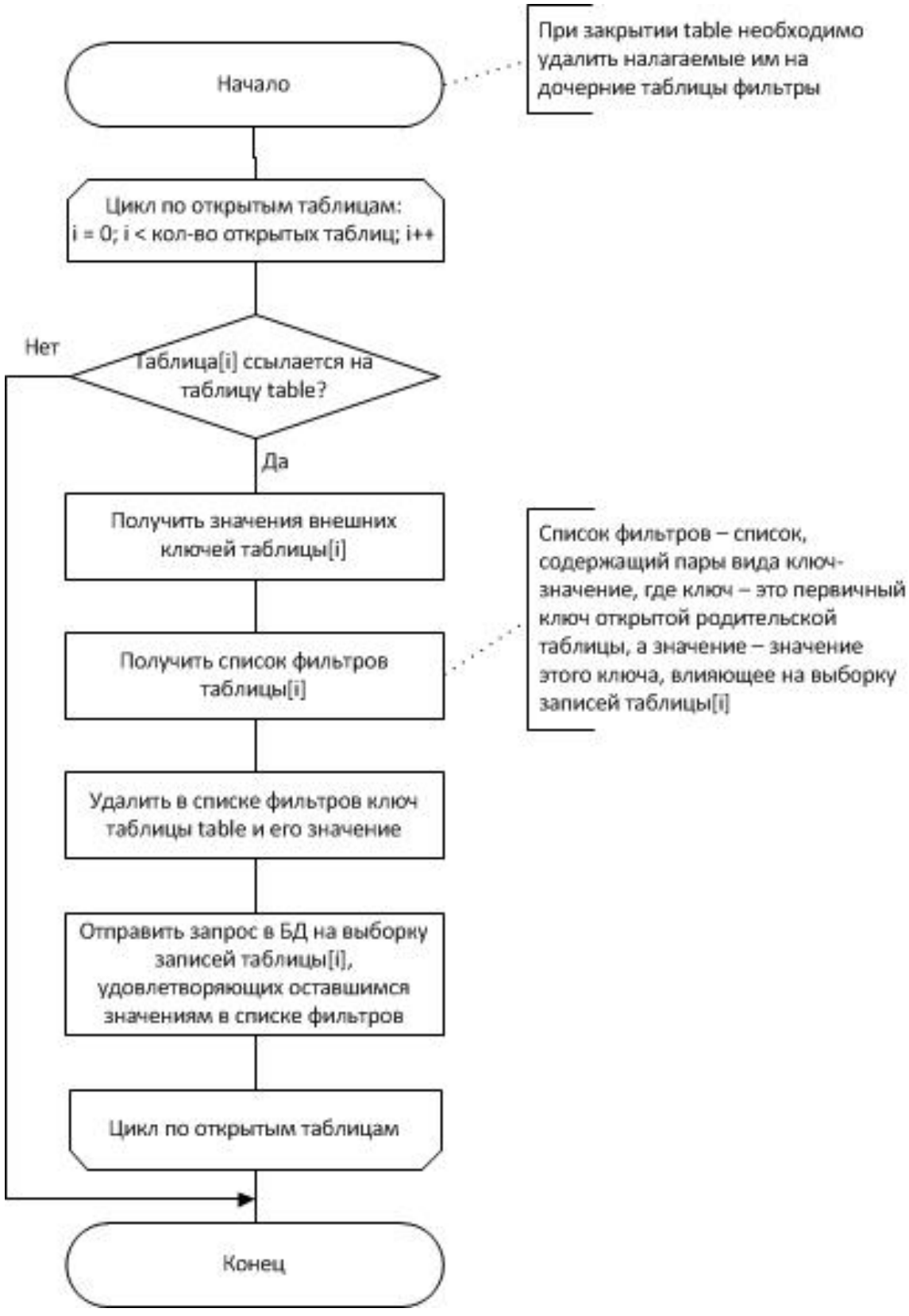

Рисунок 5.7 – Алгоритм фильтрации дочерних таблиц, удаление фильтра

5.1.5 Алгоритмы поддержания ограничений и ссылочной целостности

На рисунках 5.8-5.9 изображены алгоритмы проверок, выполняющихся при модификации/добавлении и удалении записи.

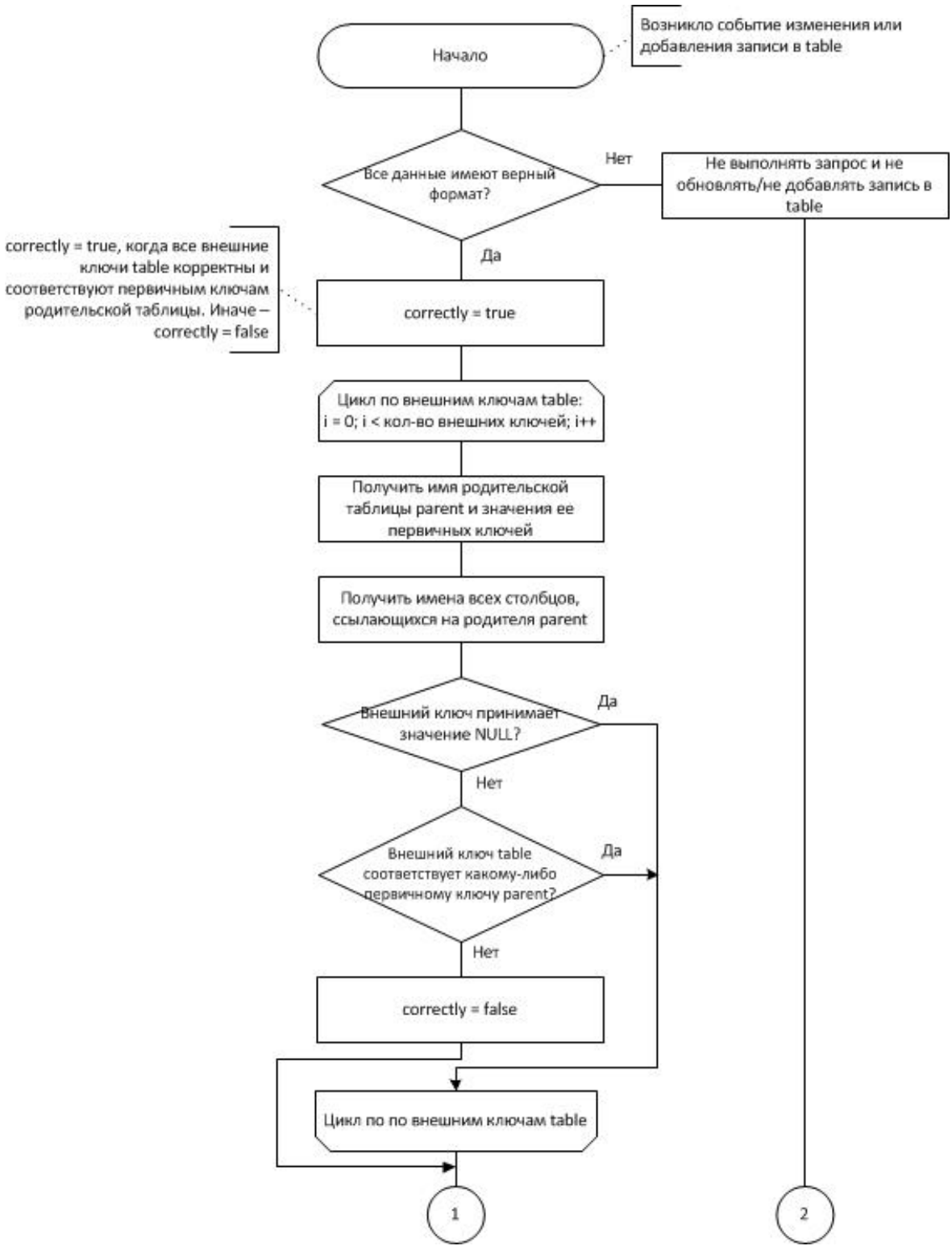

Рисунок 5.8 – Алгоритм проверки корректности данных при редактировании/добавлении записи

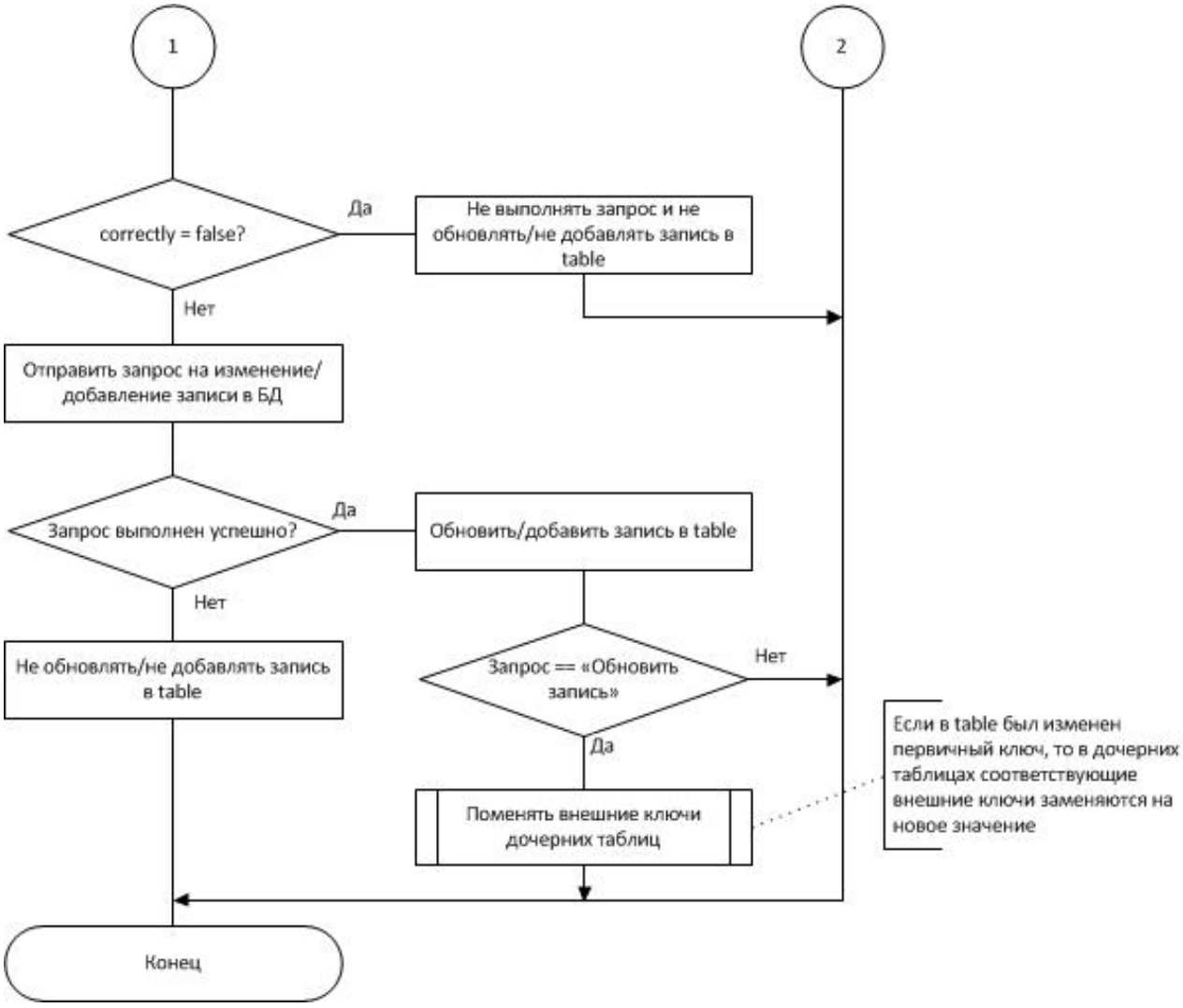

Рисунок 5.8 – Окончание

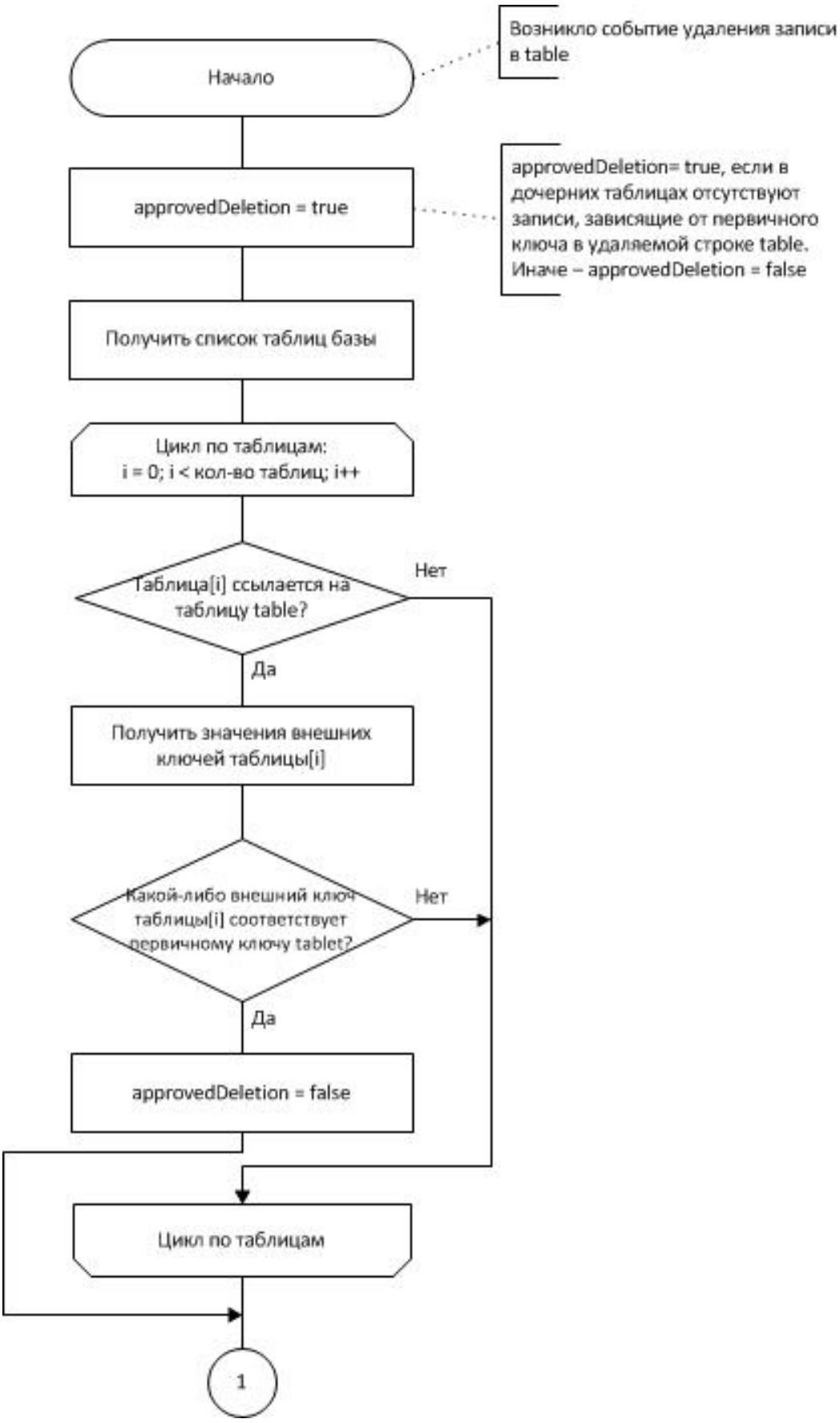

Рисунок 5.9 – Алгоритм проверки при удалении записи

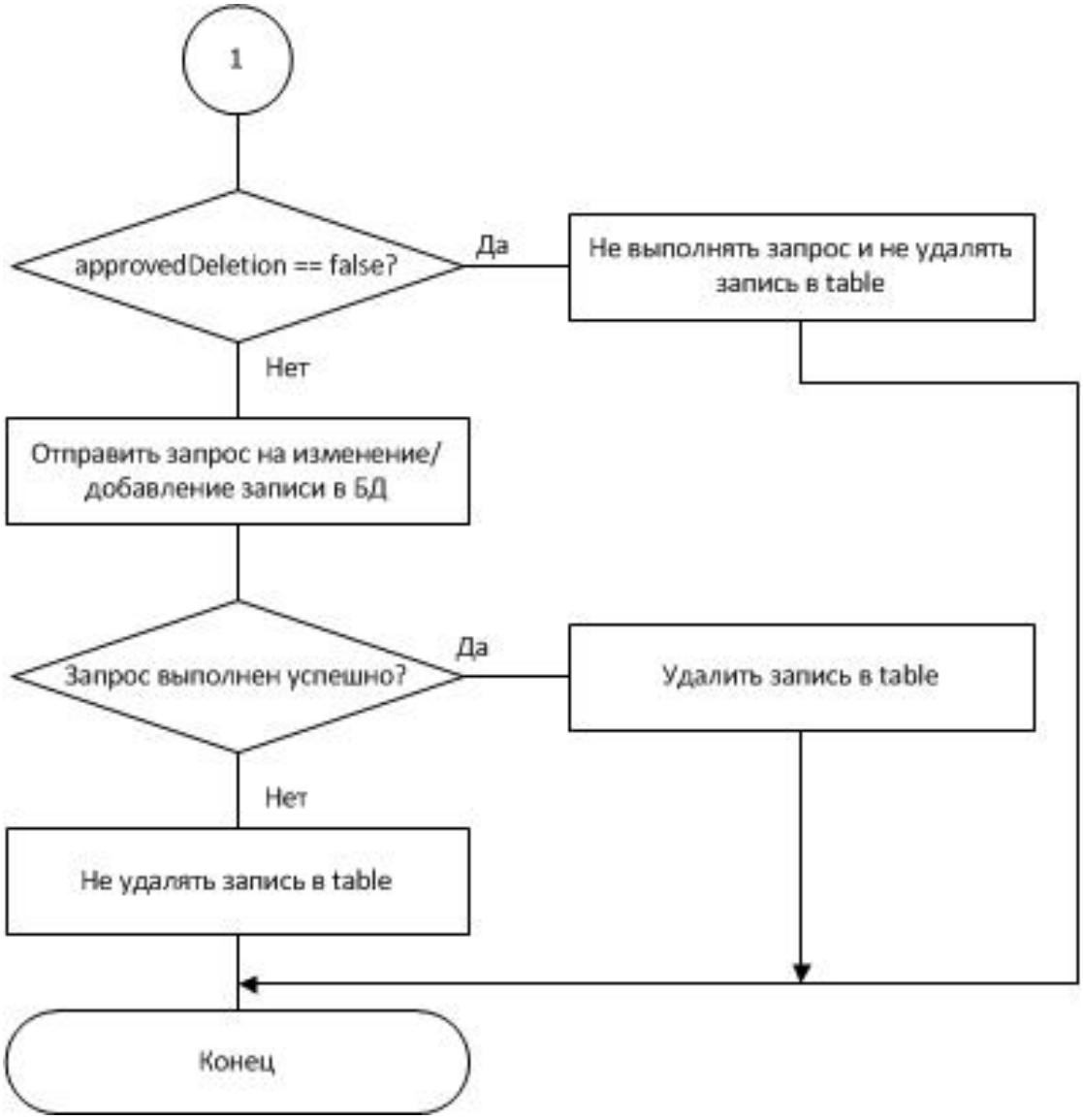

Рисунок 5.9 – Окончание

5.2 Тестирование системы

В процессе тестирования необходимо показать, что система безошибочно определяет внешние ключи таблиц; в режиме фильтрации отображает данные, удовлетворяющие заданным фильтрам; обеспечивает надежность удаления, добавления и редактирования табличных записей.

1. Определение внешних ключей. Для тестирования нужно предоставить системе следующие тестовые данные:

- две таблицы, одна из которых – родитель с единственным столбцом, входящим в состав первичного ключа, другая – ребенок, ссылающийся на родительскую таблицу. Протестировать таблицы типов InnoDB и MyISAM, в последнем случае первичный и внешний ключи должны иметь одинаковые имена.
- Две таблицы, одна из которых – родитель с несколькими столбцами, входящими в состав первичного ключа (составной первичный ключ), другая – ребенок, ссылающийся на родительскую таблицу. Протестировать таблицы типов InnoDB и MyISAM, в последнем случае столбцы первичного и внешнего ключей должны иметь одинаковые имена.
- Дочернюю таблицу, ссылающуюся на несколько родительских таблиц. Проверить таблицы обеих типов (MyISAM и InnoDB).
- Родительскую таблицу, на которую ссылаются несколько дочерних таблиц. Проверить таблицы обеих типов (MyISAM и InnoDB).
- Цепочку таблиц, ссылающихся друг на друга (вида parent1 -> parent2 -> … -> child). Проверить таблицы обеих типов (MyISAM и InnoDB).
- Таблицу со связью вида «петля» (ссылающаяся на себя же). В данном случае система должна не только верно установить внешние ключи такой таблицы, но и обеспечить открытие ее «дубликата» для обеспечения фильтрации, а также своевременное обновление записей как в таблице-оригинале, так и в таблице-дубликате, если она открыта. Проверить таблицу типа InnoDB.
- Родительскую таблицу, на которую ссылается дочерняя таблица со связью «петля». При фильтрации таблица-родитель должна влиять на дочернюю, в том числе и на ее дубликат (если он открыт). Проверить таблицу типа InnoDB.
- Родительскую таблицу со связью «петля», на которую ссылается дочерняя таблица. При фильтрации на дочернюю таблицу должна оказывать влияние запись родителя-оригинала, но не родителя-дубликата. Проверить таблицу типа InnoDB.

2. Фильтрация данных. Нужно обеспечить проверку фильтрации для всех вышеприведенных типов связей. В том числе провести следующие тесты:

- открыть дочернюю таблицу и включить фильтрацию по одной или нескольким закрытым родительским таблицам, после чего открывать по одной таблице-родителю. Система должна переопределять записи, которые должны быть отображены в дочерней таблице;

- включить фильтрацию в открытой дочерней таблице по одной или нескольким открытым родительским таблицам, после чего закрывать по одной таблице-родителю. Система должна переопределять записи, которые должны быть отображены в дочерней таблице;
- выключить всю фильтрацию в открытой дочерней таблице. Система должна наполнить дочернюю таблицу всеми записями.

3. Добавление, удаление и редактирование данных. Тестирование необходимо выполнить в режиме фильтрации и без него.

- Добавить или изменить запись дочерней таблицы; внешний ключ должен принимать значение, отсутствующее в родительской таблице. Система должна запретить данную операцию и выдать соответствующее предупреждение.
- Добавить или изменить запись дочерней таблицы; внешний ключ должен принимать NULL-значение. Система должна разрешить данную операцию и отправить запрос к БД.
- Добавить или изменить запись дочерней таблицы; внешний ключ должен принимать значение, присутствующее в родительской таблице. Система должна разрешить данную операцию и отправить запрос к БД.
- Добавить или изменить запись таблицы; значение одной или нескольких ячеек должно иметь неверный формат. Система должна запретить данную операцию и выдать соответствующее предупреждение.
- Добавить или изменить запись таблицы; в ячейках, не допускающих неопределенные значения, поместить NULL. Система должна запретить данную операцию и выдать соответствующее предупреждение.
- Изменить запись родительской таблицы; первичный ключ должен принимать новое значение. Система должна разрешить данную операцию, отправить запрос к БД и, если он выполняется успешно, заменить во всех дочерних таблицах соответствующие значения их внешних ключей на новое.
- Удалить запись в родительской таблице; значение первичного ключа должно присутствовать в дочерних таблицах. Система должна запретить данную операцию и выдать соответствующее предупреждение.
- Удалить запись в родительской таблице; значение первичного ключа не должно присутствовать в дочерних таблицах. Система должна разрешить данную операцию и отправить запрос к БД.
- Добавить/изменить/удалить запись в таблице со связью «петля». В случае, если операция успешно выполнилась, система должна обеспечить обновление и в таблице-дубликате, и в таблице-оригинале.
- Добавить/изменить/удалить запись в таблице, доступной только для чтения. Система должна запретить данную операцию и выдать соответствующее предупреждение.

Для отладки и тестирования были созданы две базы данных (см. рисунки 5.10-5.11), первая содержит таблицы InnoDB, вторая – MyISAM. Они включают в себя всевозможные виды связей и ячейки различных форматов, в том числе допускающие и не допускающие NULL-значения.

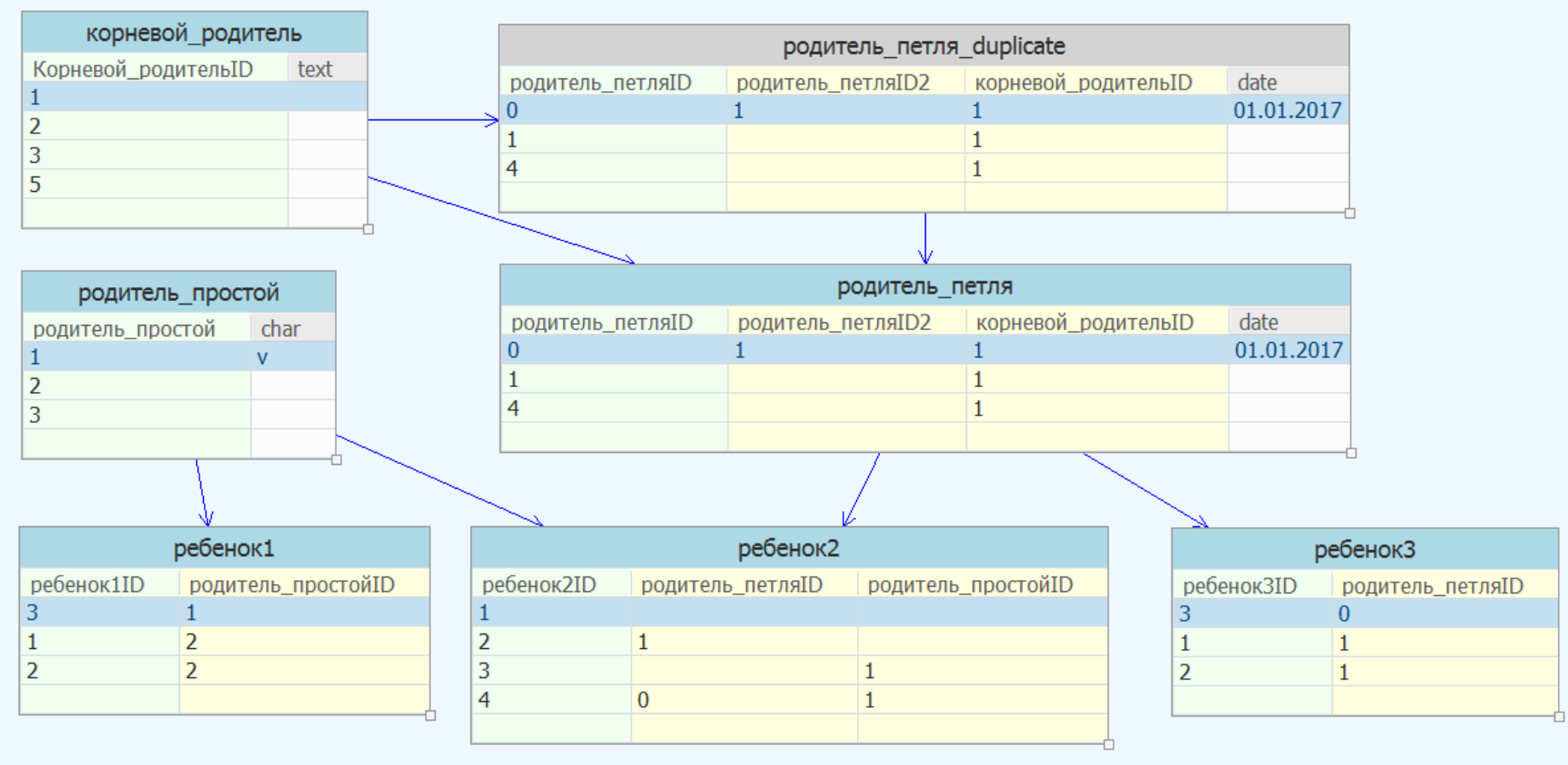

Рисунок 5.10 – База данных, содержащая таблицы InnoDB

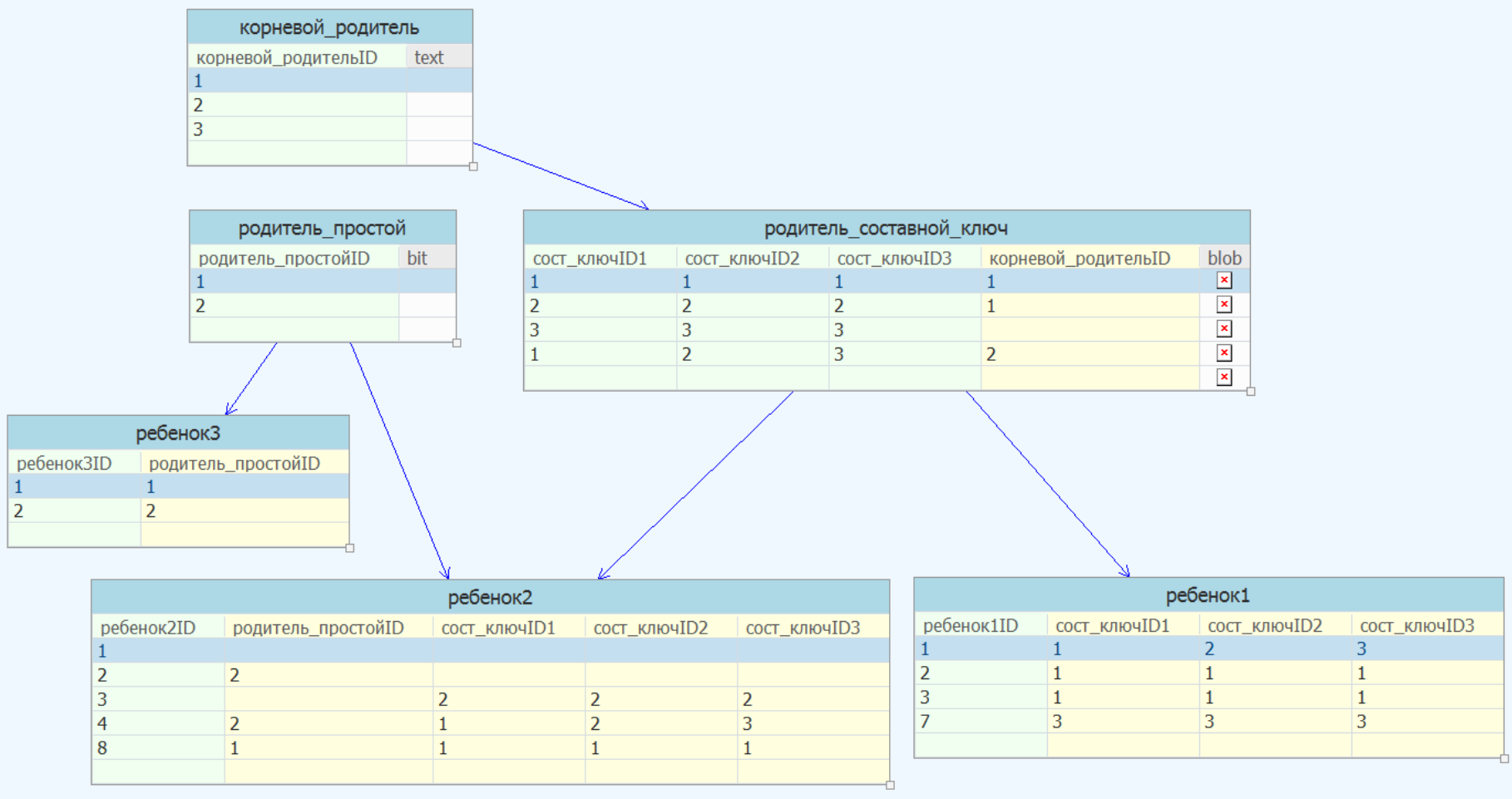

Рисунок 5.11 – База данных, содержащая таблицы MyISAM

5.3 Выводы по разделу

В данном разделе приведено описание разработанных алгоритмов. Также в разделе были описаны:

- тестовые данные, которые требуются для тестирования и отладки системы;
- список используемых тестов. Для каждого теста указан ожидаемый результат работы приложения;
- разработанные базы данных, включающие в себя необходимые тестовые данные и позволяющие протестировать поведение системы.

.

ЗАКЛЮЧЕНИЕ

В данной работе были проанализированы требования к приложению, проведен обзор существующих решений для работы с данными таблиц MySQL, описаны их достоинства и недостатки. Был выполнен сравнительный анализ рассмотренных средств и разработанного приложения, на основе чего были выявлены его основные функциональные преимущества.

Были разработаны архитектура системы и ее интерфейс, приведены схемы основных алгоритмов, обеспечивающих корректное функционирование системы. Для тестирования и отладки были разработаны тесты и две отладочные базы данных, содержащие таблицы с различными видами связей и форматами ячеек.

В результате было разработано программное обеспечение для наполнения и редактирования баз данных MySQL, отвечающее всем указанным требованиям. Таким образом, все поставленные задачи были успешно выполнены.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Понятие и функции системы управления контентом [Электронный ресурс] / Режим доступа: http://dotnetnukeru.com/default.aspx?ArticleId=8&tabid=325, свободный. – Загл. с экрана.
2. Дейт К.Дж. SQL и реляционная теория. Как грамотно писать код на SQL / К.Дж. Дейт. – пер. с англ. А. Слинкина. – СПб.; М.: Сивол-Плюс, 2010. – 480 с.
3. Кузнецов, М.В. Самоучитель MySQL 5 / М.В. Кузнецов, И.В. Симдянов. – СПб.: БХВ-Петербург, 2006. – 524 с.
4. Шварц, Б. MySQL. Оптимизация производительности / Б. Шварц, П. Зайцев, В. Ткаченко, Д. Заводны, А. Ленц и др. – 2-е изд., испр. и доп. – М.: Символ, 2010. – 708 с.
5. Обзор инструментов для работы с MySQL [Электронный ресурс] / Режим доступа: http://www.webmasters.by/articles/review-po/96, свободный. – Загл. с экрана. z
6. Орлов, В.В. Технологии разработки программных продуктов / В.В. Орлов. – СПб.: Питер, 2003. – 437 с.
7. Гольцман, В. MySQL 5.0 / В. Гольцман. – СПб.: Питер, 2010. – 253 с.
8. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C# / Дж. Рихтер. – 3-е изд., испр. и доп. – М.: Русская Редакция; СПб.: Питер, 2013. – 927 с.
9. Стиллмен, Э. Изучаем C# / Э. Стиллмен, Дж. Грин. – 2-е изд., испр. и доп. – СПб.: Питер, 2012. – 689 с.
10. Visual Studio 2013 [Электронный ресурс] / Режим доступа: https://msdn.microsoft.com/ru-ru/library/dd831853(v=vs.120).aspx, свободный. – Загл. с экрана.
11. .NET Framework 4.6 и 4.5 [Электронный ресурс] / Режим доступа: https://msdn.microsoft.com/ru-ru/library/w0x726c2(v=vs.110).aspx, свободный. – Загл. с экрана.
12. Хокинс, С. Администрирование WEB-сервера Apache и руководство по электронной коммерции / С. Хокинс. – СПб.; М.: Вильямс, 2001. – 330 с.
13. XAMPP – готовый комплект Web-сервера [Электронный ресурс] / Режим доступа: http://www.uamedwed.com/blog/soft/xampp-gotovyj-komplekt-web-servera.html, свободный. – Загл. с экрана.
14. Работа с базами данных в .NET Framework [Электронный ресурс] / Режим доступа: https://professorweb.ru/my/ADO_NET/base/level1/ado_net_index.php, свободный. – Загл. с экрана.
15. Использование INFORMATION_SCHEMA в хранимых процедурах [Электронный ресурс] / Режим доступа: http://webew.ru/articles/200.webew, свободный. – Загл. с экрана.
16. Брауде, Э. Технология разработки программного обеспечения / Э. Брауде. – СПб.: Питер, 2004. – 656 с.

17. Абельсон, Х. Структура и интерпретация компьютерных программ / Х. Абельсон, Д. Сассман. – пер. с англ. Г.К. Бронникова. – М.: Добросвет, 2012. – 608 с.
18. Тепляков, С. Паттерны проектирования на платформе .NET / С. Тепляков. – СПб.: Питер, 2015. – 320 с.
19. ГОСТ 19.201 – 78. Техническое задание. Требование к содержанию и оформлению. – М.: Изд-во стандартов, 2004. – 34 с.
20. ГОСТ 19.402 – 78. Описание программы. – М.: Изд-во стандартов, 2000. – 49 с.

ПРИЛОЖЕНИЕ 1 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Назначение системы, условия применения.

Приложение SoftwareForWorkingWithTablesMySQL предназначено для наполнения и редактирования баз данных MySQL.

Для функционирования приложения требуется:

- операционная система Windows XP и выше;
- Microsoft.NET Framework версии 4.5 и выше;
- доступ к СУБД MySQL не ниже версии 5.0;
- доступ к серверу баз данных MySQL;
- оперативная память не менее 20 Мб;
- свободная память на жестком диске не менее 25 Мб.

Пользователь должен иметь доступ ко всем базам и таблицам, обладать правами наполнения, редактирования и удаления записей. Также он должен владеть базовыми знаниями в области администрирования баз данных и иметь опыт работы с windows-приложениями.

Подготовка системы к работе.

Чтобы начать работу, необходимо скопировать папку с программным обеспечением на компьютер и запустить приложение (папка bin –> Debug –> SoftwareForWorkingWithTablesMySQL.exe). Для дальнейшего взаимодействия с системой нужно ввести в соответствующие текстовые поля сервер, логин, пароль, выбрать из выпадающего списка СУБД и нажать кнопку «Ок».

Описание операций.

1. Главное окно программы. После удачной авторизации запускается главное окно приложения, поделенное на две части:

- слева располагается список всех баз данных, доступных пользователю;
- справа размещается панель, которая будет отображать открытые пользователем таблицы.

Чтобы увидеть список таблиц конкретной базы данных, необходимо два раза щелкнуть левой кнопкой мыши по названию базы.

Сверху находится главное меню, с помощью которого можно включить или выключить отображение связей между таблицами, посмотреть информацию о приложении, сменить профиль или получить сведения о текущем профиле пользователя.

2. Работа с таблицами. Таблицу можно открыть:

- с помощью контекстного меню, которое можно вызвать, щелкнув по названию таблицы правой кнопкой мыши;
- щелкнув по названию таблицы два раза левой кнопкой мыши.

Чтобы переместить открытую таблицу в пределах панели для отображения таблиц, нужно потянуть за ее заголовок. Чтобы изменить размер таблицы, нужно потянуть за ее правый нижний угол.

Записи таблицы можно редактировать, добавлять и удалять.

- Для того, чтобы войти в режим редактирования записи, необходимо два раза щелкнуть по ячейке таблицы левой кнопкой мыши; после окончания редактирования нужно нажать Enter.
- Для того, чтобы удалить запись, нужно ее выделить и нажать клавишу Delete.
- Последняя пустая строка таблицы предназначена для добавления новых записей. После внесения значений в ячейки нужно нажать клавишу Enter.

Закрыть таблицу можно с помощью контекстного меню, щелкнув по ее заголовку правой кнопкой мыши. Закрыть сразу все открытые таблицы можно с помощью контекстного меню, щелкнув по пустому пространству рабочей панели.

3. Связи между таблицами. Связи отображаются с помощью стрелок от родительской таблицы к дочерней. Столбцы первичных ключей таблицы окрашены в зеленый цвет, столбцы внешних ключей – в желтый.

Чтобы выключить отображение связи, в главном меню нужно щелкнуть Настройки –> Отображение связи, или нажать сочетание клавиш Alt + L.

4. Фильтрация. При включенном режиме фильтрации активная запись в родительской таблице влияет на то, какие записи будут отображаться в дочерних таблицах.

Программа поддерживает связь типа "Петля" (когда таблица ссылается на саму себя). Чтобы обеспечить фильтрацию в случае такой связи, нужно щелкнуть по заголовку таблицы правой кнопкой мыши и выбрать пункт контекстного меню "Открыть дубликат". Откроется таблица-дубликат (ее заголовок окрашен в серый цвет), которая для таблицы-оригинала является аналогом родительской таблицы.

Чтобы переключить режим фильтрации для конкретной таблицы, нужно щелкнуть по ее заголовку правой кнопкой мыши и выбрать пункт контекстного меню «Фильтрация по…». В появившемся подменю указать те родительские таблицы, по которым будет производиться фильтрация записей.

ПРИЛОЖЕНИЕ 2 ТЕКСТ ПРОГРАММЫ

MainForm.cs – класс главного окна системы, содержащий метод настройки параметров окна и элементов, методы авторизации, построения списков баз и таблиц, обработки событий щелчков по контекстным меню и т.д.

```
namespace SoftwareForWorkingWithTables
{
   public partial class MainForm : Form
   {
      private Links links;
      public MainForm()
      {
         InitializeComponent();
         ConfigureFormSettings();
         Authorization();
         BuildingList();
         links = new Links(PanelForTables);
      }

      //Настройка размеров окна и главных элементов
      private void ConfigureFormSettings()
      {
         Screen screen = Screen.FromRectangle(Bounds);
         this.MinimumSize = new Size(800, 600);
         this.Width = screen.WorkingArea.Width;
         this.Height = screen.WorkingArea.Height;

         int heightWindow = this.ClientSize.Height;
         int widthWindow = this.ClientSize.Width;
         PanelForTables.Height = (Int32)(heightWindow - MainMenu.Height - 20);
         PanelForTables.Width = (Int32)(this.ClientSize.Width - ListOfDatabases.Width -
                        ListOfDatabases.Location.X * 3);
         PanelForTables.Location = new Point((Int32)(this.ClientSize.Width –
                           PanelForTables.Width - ListOfDatabases.Location.X),
                           (Int32)((this.ClientSize.Height - MainMenu.Height –
                           PanelForTables.Height) / 2 + MainMenu.Height));

         ListOfDatabases.Height = PanelForTables.Height;
         ListOfDatabases.Location = new Point(ListOfDatabases.Location.X,
                                    PanelForTables.Location.Y);
      }

      //Авторизовать пользователя
      private void Authorization()
      {
         Authorization authorization = new Authorization();
         authorization.SetOwner(this);
         authorization.ShowDialog();
         if (authorization.DialogResult == System.Windows.Forms.DialogResult.Cancel)
            System.Environment.Exit(0);
      }

      //Построить список баз и таблиц
      private void BuildingList()
      {
         ListOfDatabases.ImageList = IconsForDatabase;
         List<string> dataBases = new List<string>();
         List<string> tablesDB = new List<string>();
```

```
    dataBases = Query.query.FindDatabases();
    if (dataBases == null || dataBases.Count == 0) return;
    foreach (string dataBase in dataBases)
    {
        TreeNode node = new TreeNode { Text = dataBase, ImageIndex = 0,
                                SelectedImageIndex = 0 };
        ListOfDatabases.Nodes.Add(node);
    }

    foreach (TreeNode database in ListOfDatabases.Nodes)
    {
        tablesDB = Query.query.FindTablesForDatabase(database.Text);
        if (tablesDB == null || tablesDB.Count == 0) continue;
        foreach (string table in tablesDB)
        {
            TreeNode node = new TreeNode { Text = table, ImageIndex = 1,
                                    SelectedImageIndex = 1 };
            database.Nodes.Add(node);
        }
    }
}

//Если щелкнули по таблице правой кнопкой мыши - вызвать контекстное меню
private void ListOfDatabases_MouseClick(object sender, MouseEventArgs e)
{
    if (e.Button != MouseButtons.Right ||
        ListOfDatabases.HitTest(e.Location).Node.Level != 1) return;
    ListOfDatabases.SelectedNode = ListOfDatabases.HitTest(e.Location).Node;
    ListOfDatabases.SelectedNode.ContextMenuStrip = ContextMenuForListOfDB;
    ContextMenuForListOfDB.Show(Cursor.Position);
    ListOfDatabases.SelectedNode.ContextMenuStrip = null;
}

//В контекстном меню щелкнули на пункт "Показать таблицу"
private void MenuItem_ShowTable_Click(object sender, EventArgs e)
{
    string nameTable = ListOfDatabases.SelectedNode.Text;
    string nameDB = ListOfDatabases.SelectedNode.Parent.Text;
    Control parent = PanelForTables;
    CollectionTables.AddTable(nameTable, nameDB, parent);
}

//Или по таблице дважды щелкнули левой кнопкой мыши - открыть
private void ListOfDatabases_MouseDoubleClick(object sender, MouseEventArgs e)
{
    if (e.Button != MouseButtons.Left ||
        ListOfDatabases.HitTest(e.Location).Node.Level != 1) return;
    if(ListOfDatabases.SelectedNode.Level != 1) return;
    string nameTable = ListOfDatabases.SelectedNode.Text;
    string nameDB = ListOfDatabases.SelectedNode.Parent.Text;
    Control parent = PanelForTables;
    CollectionTables.AddTable(nameTable, nameDB, parent);
}

//Открыть конт. меню панели, если есть хотя бы одна открытая таблица
private void ContextMenuForPanel_Opening(object sender, CancelEventArgs e)
{
    if (CollectionTables.CountTables() == 0) e.Cancel = true;
}
```

```
        //В контекстном меню щелкнули на пункт "Скрыть все таблицы"
        private void MenuItem_HideTables_Click(object sender, EventArgs e)
        {
            CollectionTables.DeleteAllTables();
        }

        //Скрыть/отобразить связи
        private void MenuItem_ShowLinks_Click(object sender, EventArgs e)
        {
            MenuItem_ShowLinks.Checked = !MenuItem_ShowLinks.Checked;
            links.SetDraw(MenuItem_ShowLinks.Checked);
            PanelForTables.Invalidate();
        }

        //Сменить профиль
        private void MenuItem_ChangeProfile_Click(object sender, EventArgs e)
        {
            DialogResult result = MessageBox.Show("Прервать текущую работу?",
                        "Подтверждение", MessageBoxButtons.OKCancel);
            if (result != System.Windows.Forms.DialogResult.OK) return;

            CollectionTables.DeleteAllTables();
            MenuItem_ShowLinks.Checked = true;
            ListOfDatabases.Nodes.Clear();
            this.Visible = false;
            Authorization();
            this.Visible = true;
            BuildingList();
        }

        private void MenuItem_AboutProfile_Click(object sender,EventArgs e)
        {
            string message = "Сервер: " + Query.server + "\n";
            message += "Логин: " + Query.login + "\n";
            message += "СУБД: " + Query.DBMS;
            MessageBox.Show(message, "Информация о профиле");
        }

        private void MenuItem_AboutProgram_Click(object sender, EventArgs e)
        {
            AboutProgram aboutProgram = new AboutProgram();
            aboutProgram.ShowDialog();
        }

        private void MenuItem_Help_Click(object sender, EventArgs e)
        {
            Help help = new Help();
            help.ShowDialog();
        }

    }

    //Наследник класса Panel, в котором включена двойная буферизация
    public class myPanel : Panel { public myPanel() { DoubleBuffered = true; } }

}
```

Authorization.cs – класс окна авторизации, включающий в себя методы работы с идентификационными данными пользователя.

```
namespace SoftwareForWorkingWithTables
{
   public partial class Authorization : Form
   {
      //manager - работает с *.ini файлом, в котором
      //хранятся сведения о последнем пользователе
      private IniManager manager;
      private Logger logger = LogManager.GetCurrentClassLogger();

      public Authorization()
      {
         InitializeComponent();
         manager = new IniManager(AppDomain.CurrentDomain.BaseDirectory +
                              "\\UserData.ini");
         if (dbMS.Items.Count > 0) dbMS.SelectedItem = dbMS.Items[0];
         GetUserData();
      }

      //Получить данные из *.ini файла
      private void GetUserData(){
         serv.Text = manager.GetPrivateString("common", "SERVER NAME");
         login.Text = manager.GetPrivateString("common", "USER NAME");
      }

      //Сохранить введенные пользователем данные в *.ini файл
      private void SetUserData()
      {
         manager.WritePrivateString("common", "SERVER NAME", serv.Text);
         manager.WritePrivateString("common", "USER NAME", login.Text);
      }

      public void SetOwner(Form owner)
      {
         this.Owner = owner;
      }

      private void buttonOK_Click(object sender, EventArgs e)
      {
         Information.Text = "Проверка соединения";
         timer1.Interval = 500; timer1.Enabled = true;
         logger.Info("Попытка входа: сервер - " + serv.Text + "; логин - " + login.Text
                  + "; СУБД - " + databaseMS() + ".");

         string connectionString = "server=" + serv.Text + ";user=" + login.Text + ";";
         if (psw.Text != "")
            connectionString += "password=" + psw.Text + ";";
         connectionString += "port=3306;charset=utf8";
         Query.Build(serv.Text, login.Text, psw.Text, databaseMS(), connectionString);

         Information.Text = ""; timer1.Enabled = false;
         if (Query.query == null)
         {
            logger.Info("Неудачная попытка входа.");
            return;
         }
         this.DialogResult = System.Windows.Forms.DialogResult.OK;
         SetUserData();
         logger.Info("Вход был выполнен.");
         this.Close();
      }
```

```
        private void buttonCANCEL_Click(object sender, EventArgs e)
        {
            this.DialogResult = System.Windows.Forms.DialogResult.Cancel;
        }

        private string databaseMS()
        {
            if (dbMS.SelectedItem != null) return dbMS.SelectedItem.ToString();
            else return "";
        }

        int t = 0;
        private void timer1_Tick(object sender, EventArgs e)
        {
            t++;
            if (t >= 4) {t = 0; Information.Text = "Проверка соединения";}
            else Information.Text += ".";
        }
    }
}
```

Help.cs – класс справочного окна, в котором приведено описание основных функциональных возможностей системы.

```
namespace SoftwareForWorkingWithTables
{
    public partial class Help : Form
    {
        public Help()
        {
            InitializeComponent();
        }
    }
}
```

AboutProgram.cs – класс окна «О программе».

```
namespace SoftwareForWorkingWithTables
{
    public partial class AboutProgram : Form
    {
        public AboutProgram()
        {
            InitializeComponent();
        }
    }
}
```

IniManager.cs – класс, взаимодействующий с *.ini-файлом, в котором хранятся сведения о настройках пользователя (сервер и логин по умолчанию).

```
namespace SoftwareForWorkingWithTables
{
    public class IniManager
    {
        private const int SIZE = 1024
        private string path = null;
```

```
        public IniManager(string aPath)
        {
            path = aPath;
        }

        //Возвращает значение из INI-файла (по указанным секции и ключу)
        public string GetPrivateString(string aSection, string aKey)
        {
            StringBuilder buffer = new StringBuilder(SIZE);
            GetPrivateString(aSection, aKey, null, buffer, SIZE, path);
            return buffer.ToString();
        }

        //Пишет значение в INI-файл (по указанным секции и ключу)
        public void WritePrivateString(string aSection, string aKey, string aValue)
        {
            WritePrivateString(aSection, aKey, aValue, path);
        }

        //Импорт функции GetPrivateProfileString (для чтения значений) из библиотеки
        //kernel32.dll
        [DllImport("kernel32.dll", EntryPoint = "GetPrivateProfileString")]
        private static extern int GetPrivateString(string section, string key, string def,
                                    StringBuilder buffer, int size, string path);

        //Импорт функции WritePrivateProfileString (для записи значений) из библиотеки
        //kernel32.dll
        [DllImport("kernel32.dll", EntryPoint = "WritePrivateProfileString")]
        private static extern int WritePrivateString(string section, string key, string
                                        str, string path);
    }
}
```

CollectionTables.cs – статический класс, содержащий список открытых таблиц и включающий в себя методы, взаимодействующие с этим списком.

```
namespace SoftwareForWorkingWithTables
{
    public static class CollectionTables
    {
        public static List<Table> tables = new List<Table>();

        //Количество открытых таблиц
        public static int CountTables()
        {
            return tables.Count;
        }

        //Найти таблицу в коллекции
        public static Table FindTable(string name)
        {
            return tables.Find(x => x.GetPseudonymTable() == name);
        }

        //Добавить в коллекцию таблицу
        public static void AddTable(string nameTable, string nameDB, Control parent)
        {
            if (CountTables() != 0 && tables[0].GetNameDB() != nameDB)
```

```
            {
                DeleteAllTables();
                Add(nameTable, nameDB, parent);
            }
            else if (CountTables() != 0 && tables[0].GetNameDB() == nameDB)
            {
                if (FindTable(nameTable) != null)
                    FindTable(nameTable).Show();
                else
                    Add(nameTable, nameDB, parent);
            }
            else if (CountTables() == 0)
                Add(nameTable, nameDB, parent);
        }

        private static void Add(string nameTable, string nameDB, Control parent)
        {
            try
            {
                Table newTable = new Table(nameTable, nameDB, parent);
                if (newTable.data == null || newTable.foreignKey == null)
                    throw new Exception();
                tables.Add(newTable);
                RedrawLinks();
            }
            catch (Exception)
            {
                MessageBox.Show("Не удалось создать объект " + nameTable);
                return;
            }
        }

        //Удалить таблицу
        public static void DeleteTable(string nameTable)
        {
            tables.Remove(FindTable(nameTable));
            RedrawLinks();
        }

        //Удалить все открытые таблицы
        public static void DeleteAllTables()
        {
            while (CountTables() != 0)
                tables[0].Dispose();
        }

        public static void OpenDuplicate(string nameTable, string nameDB, Control parent)
        {
            if (FindTable(nameTable + "_duplicate") != null)
                FindTable(nameTable + "_duplicate").Show();
            else
                Add(nameTable, nameDB, parent);
        }

        public static void RedrawLinks()
        {
            if (CountTables() != 0) tables[0].InvalidatePanel();
        }
    }
}
```

BuildTable.cs – класс для создания и наполнения таблицы.

```
namespace SoftwareForWorkingWithTables
{
   public class Table
   {
      private Label label;
      private Panel panel;
      private ContextMenuStrip contextMenuForTable;
      private DataGridView grid;

      private string nameTable, nameDataBases;

      private DragAndDrop resizeAndMove;
      private ActiveRecord activeRecord;

      public DataTable data, foreignKey;

      public Dictionary<string, object> filterValues;
      private Dictionary<string, string> columnsType;
      public Dictionary<string, bool> parentsNames;
      private bool dispose = false, duplicate = false;

      public Table(string nameTable, string nameDataBases, Control parent)
      {
         this.nameTable = nameTable; this.nameDataBases = nameDataBases;
         grid = new DataGridView();
         columnsType = new Dictionary<string, string>();


         if (!FillTable())
         {
            grid.Dispose();
            data = null; foreignKey = null;
            columnsType = null;
            return;
         }

         label = new Label(); panel = new Panel();
         contextMenuForTable = new ContextMenuStrip();
         filterValues = new Dictionary<string, object>();
         parentsNames = new Dictionary<string, bool>();
         if (CollectionTables.FindTable(nameTable) != null)
            duplicate = true;

         InitialSetting(parent);
         AdditionalSettings();
         Show();
         activeRecord = new ActiveRecord(this);
         resizeAndMove = new DragAndDrop(panel);

         contextMenuForTable.ItemClicked +=
                           new ToolStripItemClickedEventHandler(ItemClicked);
         grid.RowEnter += new DataGridViewCellEventHandler(RowEnter);
         grid.DataError += new DataGridViewDataErrorEventHandler(DataError);

         grid.Rows[0].Selected = true;
         activeRecord.ChangeRecord(grid.Rows[0]);
      }
```

```
//Заполнить таблицу данными, определить ключи и тип столбцов
private bool FillTable()
{
    data = Query.query.FillTable(nameDataBases, nameTable);
    if (data == null) return false;
    grid.DataSource = data;

    foreignKey = Query.query.DefineForeignKey(nameDataBases, nameTable);
    if (foreignKey == null) return false;

    columnsType = Query.query.GetTypeColumns(nameDataBases, nameTable);
    if (columnsType == null) return false;

    return true;
}

//Настроить параметры
private void InitialSetting(Control parent)
{
    panel.BorderStyle = BorderStyle.FixedSingle;
    panel.BackColor = SystemColors.ControlLight;
    panel.AutoScroll = false;
    panel.Parent = parent;

    label.Text = GetPseudonymTable();
    label.TextAlign = ContentAlignment.MiddleCenter;
    label.Height = 30;
    if (!duplicate) label.BackColor = Color.LightBlue;
    else label.BackColor = Color.LightGray;
    label.BorderStyle = BorderStyle.None;
    label.Parent = panel;


    grid.RowHeadersVisible = false;
    grid.EnableHeadersVisualStyles = false;
    grid.ReadOnly = false;
    grid.AllowUserToAddRows = true;
    grid.AllowUserToDeleteRows = true;
    grid.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.AllCells;
    grid.AutoSizeRowsMode = DataGridViewAutoSizeRowsMode.AllCellsExceptHeaders;
    grid.DefaultCellStyle.WrapMode = DataGridViewTriState.True;
    grid.DefaultCellStyle.Alignment = DataGridViewContentAlignment.TopLeft;
    grid.AllowUserToResizeColumns = false;
    grid.BackgroundColor = SystemColors.ControlLight;
    grid.Parent = panel;
    grid.SelectionMode = DataGridViewSelectionMode.FullRowSelect;
    grid.MultiSelect = false;
    grid.BorderStyle = BorderStyle.None;
    grid.ScrollBars = ScrollBars.Both;

    panel.Location = new Point(0, 0);
    label.Location = new Point(panel.Location.X, panel.Location.Y);
    grid.Location = new Point(panel.Location.X, panel.Location.Y + label.Height);

    contextMenuForTable.Items.Add("Скрыть таблицу");
    contextMenuForTable.Items.Add("Открыть дубликат");
    contextMenuForTable.Items.Add("Фильтрация по...");
    if (duplicate || foreignKey == null || foreignKey.Rows.Count == 0
        || !foreignKey.Select("REFERENCED_TABLE_NAME = '"
                        + GetPseudonymTable() + "'").Any())
        contextMenuForTable.Items[1].Enabled = false;
    if (foreignKey.Rows.Count == 0)
```

```
            contextMenuForTable.Items[2].Enabled = false;
        else
        {
            ToolStripMenuItem itemFilter =
                    ((ToolStripMenuItem)contextMenuForTable.Items[2]);
            itemFilter.DropDownItemClicked +=
                    new ToolStripItemClickedEventHandler(DropDownItemClicked);
            foreach (DataRow row in foreignKey.Rows)
            {
                string nameParent = row["REFERENCED_TABLE_NAME"].ToString();
                if (parentsNames.ContainsKey(nameParent)) continue;
                if (nameTable == nameParent && duplicate) continue;
                parentsNames.Add(nameParent, false);
                ToolStripMenuItem itemParent = new ToolStripMenuItem(nameParent);
                itemParent.Checked = false;
                itemFilter.DropDownItems.Add(itemParent);
            }
        }
        label.ContextMenuStrip = contextMenuForTable;
    }

    //Настроить размеры, расположение и цвета столбцов
    private void AdditionalSettings()
    {
        for (int i = foreignKey.Rows.Count - 1; i >= 0; i--)
        {
            string nameColumn = foreignKey.Rows[i]["COLUMN_NAME"].ToString();
            DataGridViewColumn column = grid.Columns[nameColumn];
            column.HeaderCell.Style.BackColor = Color.LightYellow;
            column.DefaultCellStyle.BackColor = Color.LightYellow;
            column.DisplayIndex = 0;
        }
        for (int i = data.PrimaryKey.Count() - 1; i >= 0; i--)
        {
            string nameColumn = data.PrimaryKey.ElementAt(i).Caption;
            DataGridViewColumn column = grid.Columns[nameColumn];
            column.HeaderCell.Style.BackColor = Color.Honeydew;
            column.DefaultCellStyle.BackColor = Color.Honeydew;
            column.DisplayIndex = 0;
        }

        int maxWidth = panel.Parent.Width / 2, maxHeight = panel.Parent.Height / 2;

        int width = 0, height = 0;
        width = grid.Columns.GetColumnsWidth(DataGridViewElementStates.Visible);
        height = grid.Rows.GetRowsHeight(DataGridViewElementStates.Visible) +
                grid.ColumnHeadersHeight;

        if (height > maxHeight) panel.Height = maxHeight;
        else panel.Height = height;
        if (width > maxWidth) panel.Width = maxWidth;
        else panel.Width = width;
        panel.Height += SystemInformation.BorderSize.Height * 2 + label.Height;
        panel.Width += SystemInformation.BorderSize.Width * 2;
        label.Width = panel.Width;
        grid.Height = panel.Height - label.Height; grid.Width = panel.Width;

        if (grid.Controls[0].Visible)
        {
            panel.Height += SystemInformation.HorizontalScrollBarHeight;
            grid.Height += SystemInformation.HorizontalScrollBarHeight;
```

```
    }
    if (grid.Controls[1].Visible)
    {
        panel.Width += SystemInformation.VerticalScrollBarWidth;
        label.Width = panel.Width; grid.Width = panel.Width;
    }
}

//На первый план
public void Show() { panel.BringToFront(); }

//Изменить текущую строку
void RowEnter(object sender, DataGridViewCellEventArgs e)
{
    activeRecord.ChangeRecord(grid.Rows[e.RowIndex]);
}

//Переключить режим фильтрации
public void Filter(bool filter, string childName)
{
    if (filter)
        activeRecord.FilterChildTables(childName);
    else
        activeRecord.DeleteFilterChildTables(childName);
}

//Перерисовка стрелок
public void InvalidatePanel() { panel.Parent.Invalidate(); }

public string GetNameTable() { return nameTable; }

public string GetPseudonymTable()
{
    if (duplicate) return nameTable + "_duplicate";
    else return nameTable;
}

public string GetNameDB() { return nameDataBases; }

public bool GetDuplicate() { return duplicate; }

public List<int> GetPanelSettings()
{
    List<int> panelSettings = new List<int>();
    panelSettings.Add(panel.Width);
    panelSettings.Add(panel.Height);
    panelSettings.Add(panel.Location.X);
    panelSettings.Add(panel.Location.Y);

    return panelSettings;
}

//Установить источник данных
public void SetDataSource(DataTable newSource)
{
    if (newSource == null) return;

    activeRecord.UnsubscribeEventRowChanged(data);
    data = newSource.Copy();
    grid.DataSource = data;
```

```
    activeRecord.SubscribeEventRowChanged(data);

    grid.Rows[0].Selected = true;
    activeRecord.ChangeRecord(grid.Rows[0]);
}

//Щелчок по контекстному меню
private void ItemClicked(object sender, ToolStripItemClickedEventArgs e)
{
    if (e.ClickedItem.ToString() == "Скрыть таблицу") { Dispose(); return; }
    if (e.ClickedItem.ToString() == "Открыть дубликат")
    {
        CollectionTables.OpenDuplicate(nameTable, nameDataBases, panel.Parent);
        return;
    }
}

//Щелчок по подменю контекстного меню
void DropDownItemClicked(object sender, ToolStripItemClickedEventArgs e)
{
    string clickedItem = e.ClickedItem.ToString();
    if (!parentsNames.ContainsKey(clickedItem)) return;

    ((ToolStripMenuItem)e.ClickedItem).Checked =
            !((ToolStripMenuItem)e.ClickedItem).Checked;
    parentsNames[clickedItem] = ((ToolStripMenuItem)e.ClickedItem).Checked;

    Table parent = null;
    if (clickedItem == nameTable)
        parent = CollectionTables.FindTable(clickedItem + "_duplicate");
    else
        parent = CollectionTables.FindTable(clickedItem);
    if (parent != null)
        parent.Filter(parentsNames[clickedItem], GetPseudonymTable());
}

//Оповестить о возникшей ошибке ввода в grid'e
private void DataError(object sender, DataGridViewDataErrorEventArgs e)
{
    Logger logger = LogManager.GetCurrentClassLogger();
    logger.Error(e.Exception.Message);
    logger.Trace("Не удалось внести изменения в таблицу " + nameTable + ".");
    MessageBox.Show(e.Exception.Message, e.Exception.GetType().Name);
}

//Освобождение ресурсов
public bool IsDispose() { return dispose; }

public void Dispose()
{
    dispose = true;
    activeRecord.DeleteFilterChildTables();
    panel.Hide(); contextMenuForTable.Hide();
    CollectionTables.DeleteTable(GetPseudonymTable());
    if (!GetDuplicate() &&
        CollectionTables.FindTable(nameTable + "_duplicate") != null &&
        CollectionTables.FindTable(nameTable + "_duplicate").GetDuplicate())
        CollectionTables.FindTable(nameTable + "_duplicate").Dispose();
    contextMenuForTable.ItemClicked -= new
                    ToolStripItemClickedEventHandler(ItemClicked);
    grid.RowEnter -= new DataGridViewCellEventHandler(RowEnter);
```

```
            grid.DataError -= new DataGridViewDataErrorEventHandler(DataError);
            if (contextMenuForTable.Items[2].Enabled)
            {
                ToolStripMenuItem itemFilter =
                        ((ToolStripMenuItem)contextMenuForTable.Items[2]);
                itemFilter.DropDownItemClicked -=
                        new ToolStripItemClickedEventHandler(DropDownItemClicked);
            }
            resizeAndMove.Dispose(); activeRecord.Dispose();
            label.Dispose(); panel.Dispose();
            contextMenuForTable.Dispose(); grid.Dispose();
            data = null; foreignKey = null;
            resizeAndMove = null; activeRecord = null;
            filterValues = null; columnsType = null;
            parentsNames = null;
        }
    }
}
```

ActiveRecord.cs – класс, работающий с активной строкой таблицы.

```
namespace SoftwareForWorkingWithTables
{
    public class ActiveRecord
    {
        private DataGridViewRow activeRecord;
        private Logger logger;
        private Table table;
        private Dictionary<string, object> oldValues;

        public ActiveRecord(Table table)
        {
            activeRecord = new DataGridViewRow();
            oldValues = new Dictionary<string, object>();
            this.table = table;
            logger = LogManager.GetCurrentClassLogger();
            SubscribeEventRowChanged(table.data);
        }

        //Поменять текущую строку
        public void ChangeRecord(DataGridViewRow activeRecord)
        {
            this.activeRecord = activeRecord;
            oldValues.Clear();
            if (activeRecord != null && activeRecord.Index >= 0)
            {
                activeRecord.Selected = true;
                for (int i = 0; i < activeRecord.Cells.Count; i++)
                    oldValues.Add(activeRecord.Cells[i].OwningColumn.Name,
                            activeRecord.Cells[i].Value);
            }

            FilterChildTables();
        }

        //Отфильтровать значения дочерней таблицы
        //если  child == null - фильтр всех дочерних таблиц
        public void FilterChildTables(string child = null)
        {
            if (child == null)
```

```
        foreach (Table childTable in CollectionTables.tables)
        {
            string childName = childTable.GetPseudonymTable();
            FilterChildTable(childName);
        }
    else
        FilterChildTable(child);
}

private void FilterChildTable(string child)
{
    Table childTable = CollectionTables.FindTable(child);
    if (childTable == null) return;
    if (!childTable.parentsNames.ContainsKey(table.GetNameTable())) return;
    if (!childTable.parentsNames[table.GetNameTable()]) return;
    if (childTable.GetPseudonymTable() == table.GetPseudonymTable()) return;
    if (childTable.GetNameTable() != table.GetNameTable() &&
            table.GetDuplicate()) return;

    DataRow[] relations = childTable.foreignKey.Select("REFERENCED_TABLE_NAME = '"
                                    + table.GetNameTable() + "'");
    foreach (DataRow relation in relations)
    {
        string nameChildColumn = relation["COLUMN_NAME"].ToString();
        string nameColumn = relation["REFERENCED_COLUMN_NAME"].ToString();
        object value;
        if (activeRecord != null)
            value = activeRecord.Cells[nameColumn].Value;
        else value = null;

        if (!childTable.filterValues.ContainsKey(nameChildColumn))
            childTable.filterValues.Add(nameChildColumn, value);
        else
            childTable.filterValues[nameChildColumn] = value;
    }

    DataTable filterTable = Query.query.Filter(table.GetNameDB(),
                    childTable.GetNameTable(), childTable.filterValues);
    if (filterTable == null) return;
    childTable.SetDataSource(filterTable);
}

//Удалить налагаемые таблицей фильтры
public void DeleteFilterChildTables(string child = null)
{
    if (child == null)
        foreach (Table childTable in CollectionTables.tables)
        {
            string childName = childTable.GetPseudonymTable();
            DeleteFilterChildTables(childName);
        }
    else
        DeleteFilterChildTable(child);
}

private void DeleteFilterChildTable(string child)
{
    Table childTable = CollectionTables.FindTable(child);
    if (childTable == null) return;
    if (!childTable.parentsNames.ContainsKey(table.GetNameTable())) return;
    if (childTable.parentsNames[table.GetNameTable()] &&
```

```
        !table.IsDispose()) return;
    if (!childTable.parentsNames[table.GetNameTable()] &&
        table.IsDispose()) return;

    DataRow[] relations = childTable.foreignKey.Select("REFERENCED_TABLE_NAME = '"
                                    + table.GetNameTable() + "'");
    foreach (DataRow relation in relations)
    {
        string nameChildColumn = relation["COLUMN_NAME"].ToString();
        if (!childTable.filterValues.ContainsKey(nameChildColumn)) continue;
        childTable.filterValues.Remove(nameChildColumn);
    }

    DataTable newData = new DataTable();
    if (childTable.filterValues.Count == 0)
        newData = Query.query.FillTable(table.GetNameDB(),
                                    childTable.GetNameTable());
    else
        newData = Query.query.Filter(table.GetNameDB(), childTable.GetNameTable(),
                                    childTable.filterValues);

    if (newData == null) return;
    childTable.SetDataSource(newData);
}

//-----------------------------------------------
//Обработка изменения/добавления/удаления строки
//-----------------------------------------------

bool approvedDeletion = false;
void RowDeleted(object sender, DataRowChangeEventArgs e)
{
    if (!approvedDeletion)
        table.data.RejectChanges();
    else
    {
        table.data.AcceptChanges();
        UpdateDuplicate();
    }
    approvedDeletion = false;
}
void RowDeleting(object sender, DataRowChangeEventArgs e)
{
    approvedDeletion = CheckingPK();
    if (approvedDeletion)
        approvedDeletion = Query.query.Delete(table.GetNameDB(),
                        table.GetNameTable(), oldValues);
    else
        logger.Trace("Запрос на удаление записи таблицы " +
                table.GetNameTable() + " был отклонен.");
}

private bool ignoreValueCheck = false;
private void RowChanged(object sender, DataRowChangeEventArgs e)
{
    if (ignoreValueCheck) return;
    if (e.Action != DataRowAction.Add && e.Action != DataRowAction.Change) return;
    AddChangeRow(e.Row, e.Action);
}
```

```
private void AddChangeRow(DataRow newRow, DataRowAction action)
{
   Dictionary<string, object> newValues = new Dictionary<string, object>();
   for (int i = 0; i < newRow.ItemArray.Count(); i++)
      newValues.Add(table.data.Columns[i].ColumnName, newRow[i]);

   bool correctly = true;
   if (table.foreignKey.Rows.Count != 0)
      foreach (DataRow FK in table.foreignKey.Rows)
      {
         string nameColumn = newValues.First(x => x.Key ==
                     FK["COLUMN_NAME"].ToString()).Key;
         object value = newValues[nameColumn];
         if (action == DataRowAction.Change &&
            oldValues[nameColumn].ToString() == value.ToString()) continue;

         correctly = CheckingFK(nameColumn, newValues);
         if (!correctly) break;
      }

   if(!correctly)
   {
      ignoreValueCheck = true;
      newRow.RejectChanges();
      ignoreValueCheck = false;
      if(action == DataRowAction.Add)
         logger.Trace("Запрос на добавление новой записи в таблицу " +
                  table.GetNameTable() + " был отклонен.");
      else
         logger.Trace("Запрос на обновление записи таблицы " +
                  table.GetNameTable() + " был отклонен.");
      return;
   }

   bool completed = false;
   if (action == DataRowAction.Add)
      completed = Query.query.Insert(table.GetNameDB(), table.GetNameTable(),
                        newValues);
   if (action == DataRowAction.Change)
      completed = Query.query.Update(table.GetNameDB(),
         table.GetNameTable(), newValues, oldValues);

   if (!completed)
   {
      ignoreValueCheck = true;
      newRow.RejectChanges();
      ignoreValueCheck = false;
      return;
   }

   ignoreValueCheck = true;
   newRow.AcceptChanges();
   ignoreValueCheck = false;

   if (action == DataRowAction.Change)
      foreach (DataColumn PK in table.data.PrimaryKey)
         if (oldValues[PK.ColumnName].ToString() !=
            newValues[PK.ColumnName].ToString())
         {
            ChangeFK_ChildTables(newValues);
            break;
```

```
            }

        foreach (KeyValuePair<string, object> newValue in newValues)
            oldValues[newValue.Key] = newValue.Value;


        if (table.filterValues.Count != 0)
            foreach (DataRow FK in table.foreignKey.Rows)
            {
                string nameColumn = FK["COLUMN_NAME"].ToString();

                if (!table.filterValues.ContainsKey(nameColumn) ||
                    Object.Equals(table.filterValues[nameColumn],
                            newValues[nameColumn])) continue;


                DataTable tableNew = Query.query.Filter(table.GetNameDB(),
                    table.GetNameTable(), table.filterValues);
                if (tableNew != null) table.SetDataSource(tableNew);
                break;
            }

        UpdateDuplicate();
    }

    //Если есть связь типа "петля" и открыт дубликат - обеспечить обновление и
    //дубликата, и оригинала
    private void UpdateDuplicate()
    {
        if (table.GetDuplicate())
        {
            Table original = CollectionTables.FindTable(table.GetNameTable());
            DataTable tableNew = null;
            if (original.filterValues.Count != 0)
                tableNew = Query.query.Filter(original.GetNameDB(),
                        original.GetNameTable(), original.filterValues);
            else
                tableNew = Query.query.FillTable(original.GetNameDB(),
                        original.GetNameTable());
            if (tableNew != null)
                original.SetDataSource(tableNew);
            return;
        }
        if (!table.GetDuplicate() && CollectionTables.FindTable(table.GetNameTable() +
                            "_duplicate") != null)
        {
            Table duplicate = CollectionTables.FindTable(table.GetNameTable() +
                            "_duplicate");
            DataTable tableNew = null;
            if (duplicate.filterValues.Count != 0)
                tableNew = Query.query.Filter(duplicate.GetNameDB(),
                        duplicate.GetNameTable(), duplicate.filterValues);
            else
                tableNew = Query.query.FillTable(duplicate.GetNameDB(),
                        duplicate.GetNameTable());
            if (tableNew != null)
                duplicate.SetDataSource(tableNew);
            return;
        }
    }

    //Проверяет корректность устанавливаемых внешних ключей
    private bool CheckingFK(string nameColumn, Dictionary<string, object> newValues)
```

```
{
   string parentName = table.foreignKey.Select("COLUMN_NAME = '" +
               nameColumn + "'").First()["REFERENCED_TABLE_NAME"].ToString();
   DataRow[] relations = table.foreignKey.Select("REFERENCED_TABLE_NAME = '" +
                     parentName + "'");
   Table parentTable = CollectionTables.FindTable(parentName);
   DataTable parentPK = new DataTable();
   if (parentTable != null && parentTable.filterValues.Count == 0)
      parentPK = parentTable.data;
   else
   {
      List<string> nameColumns = new List<string>();
      foreach (DataRow relation in relations)
         nameColumns.Add(relation["REFERENCED_COLUMN_NAME"].ToString());
      parentPK = Query.query.GetColumnsValues(table.GetNameDB(),
               parentName, nameColumns);
   }
   if (parentPK == null) return false;

   bool NULL = true;
   for (int i = 0; i < relations.Count(); i++)
   {
      string childColumnName = relations[i]["COLUMN_NAME"].ToString();
      string parentColumnName =
                     relations[i]["REFERENCED_COLUMN_NAME"].ToString();
      if (newValues[childColumnName].ToString() != "") { NULL = false; break; }
   }
   if (NULL) return true;

   string select = "";
   for (int i = 0; i < relations.Count(); i++)
   {
      string childColumnName = relations[i]["COLUMN_NAME"].ToString();
      string parentColumnName =
                     relations[i]["REFERENCED_COLUMN_NAME"].ToString();
      if (newValues[childColumnName].ToString() == "")
         select += parentColumnName + " is NULL";
      else
         select += parentColumnName + " = '" +
                     newValues[childColumnName].ToString() + "'";
      if (i < relations.Count() - 1) select += " AND ";
   }

   if (!parentPK.Select(select).Any())
   {
      string message = "Внешний ключ '";
      foreach (DataRow relation in relations)
         message += relation["COLUMN_NAME"].ToString() + " ";
      message += "' не может принимать значения,
                  отсутствующие в родительской таблице.";
      logger.Error(message);
      MessageBox.Show(message, "Запрос был отменен");
      return false;
   }
   return true;
}

//При изменении уникального ключа заменить соотв. внешние ключи
//дочерних таблиц на новое значение
private void ChangeFK_ChildTables(Dictionary<string, object> newValues)
{
```

```
    DataColumn[] columnsPK = table.data.PrimaryKey;
    List<string> allTables = Query.query.FindTablesForDatabase(table.GetNameDB());
    if (allTables == null) return;
    foreach (string nameChildTable in allTables)
    {
        if (nameChildTable == table.GetNameTable()) continue;
        Table childTable = CollectionTables.FindTable(nameChildTable);
        DataTable childForeignKey = new DataTable();
        if (childTable != null)
            childForeignKey = childTable.foreignKey;
        else
            childForeignKey = Query.query.DefineForeignKey(table.GetNameDB(),
                        nameChildTable);
        if (childForeignKey == null || childForeignKey.Rows.Count == 0) continue;
        if (!childForeignKey.Select("REFERENCED_TABLE_NAME = '" +
            table.GetNameTable() + "'").Any()) continue;

        DataRow[] relations = childForeignKey.Select("REFERENCED_TABLE_NAME = '" +
                        table.GetNameTable() + "'");
        Dictionary<string, object> definitions = new Dictionary<string, object>();
        Dictionary<string, object> values = new Dictionary<string, object>();
        foreach (DataRow relation in relations)
        {
            string nameChildColumn = relation["COLUMN_NAME"].ToString();
            string nameParentColumn =
                         relation["REFERENCED_COLUMN_NAME"].ToString();
            definitions.Add(nameChildColumn, oldValues[nameParentColumn]);
            values.Add(nameChildColumn, newValues[nameParentColumn]);
        }
        bool completed = Query.query.Update(table.GetNameDB(), nameChildTable,
                    values, definitions);
        if (childTable == null || !completed) continue;

        DataTable newChildData = new DataTable();
        if (childTable.filterValues.Count == 0)
            newChildData = Query.query.FillTable(table.GetNameDB(),
                        nameChildTable);
        else if(childTable.parentsNames[table.GetNameTable()])
        {
            foreach (DataRow relation in relations)
            {
                string nameChildColumn = relation["COLUMN_NAME"].ToString();
                string nameParentColumn =
                         relation["REFERENCED_COLUMN_NAME"].ToString();
                childTable.filterValues[nameChildColumn] =
                         newValues[nameParentColumn];
            }
            newChildData = Query.query.Filter(table.GetNameDB(), nameChildTable,
                        childTable.filterValues);
        }
        else
            newChildData = Query.query.Filter(table.GetNameDB(), nameChildTable,
                        childTable.filterValues);

        childTable.SetDataSource(newChildData);
    }
}

//При удалении строки проверить внешние ключи дочерних таблиц.
//Если PK_родителя = FK_ребенка - запретить удаление строки
private bool CheckingPK()
```

```
{
   DataColumn[] columnsPK = table.data.PrimaryKey;
   List<string> allTables = Query.query.FindTablesForDatabase(table.GetNameDB());
   if (allTables == null) return false;
   foreach (string nameChildTable in allTables)
   {
      Table childTable = CollectionTables.FindTable(nameChildTable);
      DataTable childForeignKey = new DataTable();
      if (childTable != null)
         childForeignKey = childTable.foreignKey;
      else
         childForeignKey = Query.query.DefineForeignKey(table.GetNameDB(),
                  nameChildTable);
      if (childForeignKey == null) return false;
      if (childForeignKey.Rows.Count == 0) continue;
      if (!childForeignKey.Select("REFERENCED_TABLE_NAME = '" +
                  table.GetNameTable() + "'").Any()) continue;

      DataRow[] relations = childForeignKey.Select("REFERENCED_TABLE_NAME = '" +
                  table.GetNameTable() + "'");
      DataTable childFK = new DataTable();
      if (childTable != null && childTable.filterValues.Count == 0)
         childFK = childTable.data;
      else
      {
         List<string> nameColumns = new List<string>();
         foreach (DataRow relation in relations)
            nameColumns.Add(relation["COLUMN_NAME"].ToString());
         childFK = Query.query.GetColumnsValues(table.GetNameDB(),
                  nameChildTable, nameColumns);
      }
      if (childFK == null) return false;

      string select = "";
      for (int i = 0; i < relations.Count(); i++)
      {
         string childColumnName = relations[i]["COLUMN_NAME"].ToString();
         string parentColumnName =
                  relations[i]["REFERENCED_COLUMN_NAME"].ToString();
         select += childColumnName + " = '" +
                  oldValues[parentColumnName].ToString() + "'";
         if (i < relations.Count() - 1) select += " AND ";
      }
      if (childFK.Select(select).Any())
      {
         string message = "Ключ '";
         foreach (DataRow relation in relations)
            message += relation["REFERENCED_COLUMN_NAME"].ToString() + " ";
         message += "' влияет на таблицу " + nameChildTable +
               ". Сначала удалите зависимые строки дочерней таблицы.";
         logger.Error(message);
         MessageBox.Show(message, "Запрос был отменен");
         return false;
      }
   }
   return true;
}

//-------------------------------------------
//Подписаться/отписаться от события изменения строки
//(Необходимо, когда меняется data при фильтрации)
```

```
        public void SubscribeEventRowChanged(DataTable data)
        {
            data.RowChanged += new DataRowChangeEventHandler(RowChanged);
            data.RowDeleting += new DataRowChangeEventHandler(RowDeleting);
            data.RowDeleted += new DataRowChangeEventHandler(RowDeleted);
        }
        public void UnsubscribeEventRowChanged(DataTable data)
        {
            data.RowChanged -= new DataRowChangeEventHandler(RowChanged);
            data.RowDeleting -= new DataRowChangeEventHandler(RowDeleting);
            data.RowDeleted -= new DataRowChangeEventHandler(RowDeleted);
        }

        public void Dispose()
        {
            table.data.RowChanged -= new DataRowChangeEventHandler(RowChanged);
            table.data.RowDeleting -= new DataRowChangeEventHandler(RowDeleting);
            table.data.RowDeleted -= new DataRowChangeEventHandler(RowDeleted);

            table = null;
            activeRecord = null;
        }
    }
}
```

DragAndDrop.cs – класс, в котором обрабатываются события перемещения и изменение размеров таблицы.

```
namespace SoftwareForWorkingWithTables
{
    public class DragAndDrop
    {
        private Control control;
        private Label handle;
        private const int HANDLE_SIZE = 8;
        private const int MIN_SIZE = 50;

        private int xMouse, yMouse;
        private int oldW, oldH;
        private bool dragging;

        public DragAndDrop(Control control)
        {
            this.control = control;

            Control label = control.Controls[0];
            label.Cursor = Cursors.Hand;

            handle = new Label();
            handle.Text = ""; handle.FlatStyle = 0;
            handle.BorderStyle = BorderStyle.FixedSingle;
            handle.BackColor = SystemColors.ControlLight;
            handle.Cursor = Cursors.SizeNWSE;
            control.Parent.Controls.Add(handle);
            MoveHandle(); ShowHandle();

            handle.MouseDown += new MouseEventHandler(this.handle_MouseDown);
            handle.MouseMove += new MouseEventHandler(this.handle_MouseMove);
            handle.MouseUp += new MouseEventHandler(this.handle_MouseUp);
```

```
    label.MouseDown += new MouseEventHandler(control_MouseDown);
    label.MouseMove += new MouseEventHandler(control_MouseMove);
    label.MouseUp += new MouseEventHandler(control_MouseUp);
}

//Показать хендл
private void ShowHandle()
{
    handle.BringToFront();
    handle.Visible = true;
}

//Скрыть хендл
private void HideHandle()
{
    handle.Visible = false;
}

//Установить хендл (в правом нижнем углу контрола)
private void MoveHandle()
{
    int x = control.Left - HANDLE_SIZE, y = control.Top - HANDLE_SIZE;
    int w = control.Width + HANDLE_SIZE, h = control.Height + HANDLE_SIZE;
    int PosX = x + w - HANDLE_SIZE / 2, PosY = y + h - HANDLE_SIZE / 2;
    handle.SetBounds(PosX, PosY, HANDLE_SIZE, HANDLE_SIZE);
}

//Методы для перемещения.
//Щелчок по заголовку фрейма
private void control_MouseDown(object sender, MouseEventArgs e)
{
    control.BringToFront();
    if (e.Button == MouseButtons.Left)
    {
        xMouse = e.X; yMouse = e.Y; dragging = true;
    }
    else
        dragging = false;
}

//Заголовок фрейма отпущен
private void control_MouseUp(object sender, MouseEventArgs e)
{
    if (e.Button != MouseButtons.Left) return;
    dragging = false; MoveHandle(); ShowHandle();
}

//Перемещение фрейма
private void control_MouseMove(object sender, MouseEventArgs e)
{
    if (!dragging) return;

    HideHandle();
    int dy = e.Y - yMouse, dx = e.X - xMouse;
    control.Left = control.Left + dx >= 0 ? control.Left + dx : 0;
    control.Top = control.Top + dy >= 0 ? control.Top + dy : 0;

    CollectionTables.RedrawLinks();
}

//Методы для изменения размера.
```

```
        //Щелчок по хендлу
        private void handle_MouseDown(object sender,MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Left)
            {
                dragging = true; HideHandle();
                oldW = control.Width; oldH = control.Height;
            }
            else
                dragging = false;
        }

        //Перемещение хендла
        private void handle_MouseMove(object sender, MouseEventArgs e)
        {
            if (!dragging) return;

            Control label = control.Controls[0], grid = control.Controls[1];
            control.Width = oldW + e.X > MIN_SIZE ? oldW + e.X : MIN_SIZE;
            control.Height = oldH + e.Y > MIN_SIZE ? oldH + e.Y : MIN_SIZE;
            label.Width = control.Width;
            grid.Width = control.Width; grid.Height = control.Height - label.Height;

            CollectionTables.RedrawLinks();
        }

        //Хендл отпущен
        private void handle_MouseUp(object sender, MouseEventArgs e)
        {
            if (e.Button != MouseButtons.Left) return;
            dragging = false; MoveHandle(); ShowHandle();
        }

        public void Dispose()
        {
            handle.MouseDown -= new MouseEventHandler(this.handle_MouseDown);
            handle.MouseMove -= new MouseEventHandler(this.handle_MouseMove);
            handle.MouseUp -= new MouseEventHandler(this.handle_MouseUp);
            control.Controls[0].MouseDown -= new MouseEventHandler(control_MouseDown);
            control.Controls[0].MouseMove -= new MouseEventHandler(control_MouseMove);
            control.Controls[0].MouseUp -= new MouseEventHandler(control_MouseUp);
            handle.Dispose();
        }
    }
}
```

DrawLinks.cs – класс, отвечающий за отображение связей между таблицами.

```
namespace SoftwareForWorkingWithTables
{
    public class Links
    {
        myPanel panel;
        System.Drawing.Pen myPen;
        private bool draw = true;

        public Links(myPanel panel)
        {
            this.panel = panel;
            panel.Paint += new PaintEventHandler(Paint);
        }
```

```
public void SetDraw(bool draw)
{
   this.draw = draw;
}

private void Paint(object sender, PaintEventArgs e)
{
   if (!draw) return;
   if (CollectionTables.CountTables() == 0) return;
   myPen = new System.Drawing.Pen(System.Drawing.Color.Blue, 1);
   foreach (Table table in CollectionTables.tables)
   {
      float W = table.GetPanelSettings()[0], H = table.GetPanelSettings()[1],
         pX = table.GetPanelSettings()[2], pY = table.GetPanelSettings()[3];

      if (table.foreignKey.Rows.Count != 0 &&
         table.foreignKey.Select("REFERENCED_TABLE_NAME = '" +
         table.GetNameTable() + "'").Any() &&
         CollectionTables.FindTable(table.GetNameTable()
         + "_duplicate") == null)
      {
         int t = 20;
         e.Graphics.DrawLine(myPen,pX + W, pY + H / 4, pX + W + t, pY + H / 4);
         e.Graphics.DrawLine(myPen,pX + W + t,pY + H/4,pX + W + t,pY + H / 2);
         e.Graphics.DrawLine(myPen,pX + W + t, pY + H / 2, pX + W, pY + H / 2);


         DrawArrow(pX + W, pY + H / 2, pX + W + t, pY + H / 2, e);
      }

      foreach (Table childTable in CollectionTables.tables)
      {
         if (childTable.GetPseudonymTable() == table.GetPseudonymTable())
             continue;
         if (childTable.GetNameTable() == table.GetNameTable() &&
            childTable.GetDuplicate()) continue;
         if (childTable.GetNameTable() != table.GetNameTable() &&
            table.GetDuplicate()) continue;
         if (childTable.foreignKey.Rows.Count == 0) continue;
         DataRow[] relations =
                  childTable.foreignKey.Select("REFERENCED_TABLE_NAME = '" +
                  table.GetNameTable() + "'");
         if (relations.Count() == 0) continue;

         float W2 = childTable.GetPanelSettings()[0],
             H2 = childTable.GetPanelSettings()[1],
             pX2 = childTable.GetPanelSettings()[2],
             pY2 = childTable.GetPanelSettings()[3];
         float c11 = pX + W / 2, c12 = pY + H / 2,
             c21 = pX2 + W2 / 2, c22 = pY2 + H2 / 2;
         e.Graphics.DrawLine(myPen, c11, c12, c21, c22);
         List<double> coordinatesPoint = FindPointIntersection(table,
                                 childTable);
         if (coordinatesPoint == null) continue;
         DrawArrow(coordinatesPoint[0], coordinatesPoint[1], c11, c12, e);
      }
   }
   myPen.Dispose();
}

//Ищет ребро дочерней таблицы, с которым пересекается соединяющая прямая
```

```
private List<double> FindPointIntersection(Table t1, Table t2)
{
   double W1 = t1.GetPanelSettings()[0], H1 = t1.GetPanelSettings()[1],
      X1 = t1.GetPanelSettings()[2], Y1 = t1.GetPanelSettings()[3];
   double W2 = t2.GetPanelSettings()[0], H2 = t2.GetPanelSettings()[1],
      X2 = t2.GetPanelSettings()[2], Y2 = t2.GetPanelSettings()[3];

   double c11 = X1 + W1 / 2, c12 = Y1 + H1 / 2;
   double c21 = X2 + W2 / 2, c22 = Y2 + H2 / 2;

   double p11 = X2, p12 = Y2;
   double p21 = X2, p22 = Y2 + H2;

   List<double> coordinatesPoint = ExistencePointIntersection(c11, c12, c21, c22,
                           p11, p12, p21, p22);
   if (coordinatesPoint == null)
   {
      p21 = X2 + W2; p22 = Y2;
      coordinatesPoint = ExistencePointIntersection(c11, c12, c21, c22, p11,
                     p12, p21, p22);
      if (coordinatesPoint == null)
      {
         p11 = X2 + W2; p12 = Y2 + H2;
         coordinatesPoint = ExistencePointIntersection(c11, c12, c21, c22, p11,
                        p12, p21, p22);
         if (coordinatesPoint == null)
         {
            p21 = X2; p22 = Y2 + H2;
            coordinatesPoint = ExistencePointIntersection(c11, c12, c21, c22,
                           p11, p12, p21, p22);
         }
      }
   }
   return coordinatesPoint;
}

//Определяет точку пересечения двух прямых (если таковая есть)
private List<double> ExistencePointIntersection(double x11, double y11,
     double x12, double y12, double x21, double y21, double x22, double y22)
{
   double A1 = y12 - y11, B1 = x11 - x12, C1 = y11 * x12 - x11 * y12;
   double A2 = y22 - y21, B2 = x21 - x22, C2 = y21 * x22 - x21 * y22;

   if (A1 * B2 - A2 * B1 == 0) return null;

   double x = (C2 * B1 - C1 * B2) / (A1 * B2 - A2 * B1);
   double y = (A2 * C1 - A1 * C2) / (A1 * B2 - A2 * B1);

   if (x < x11 && x < x12 || x > x11 && x > x12 ||
      x < x21 && x < x22 || x > x21 && x > x22) return null;
   if (y < y11 && y < y12 || y > y11 && y > y12 ||
      y < y21 && y < y22 || y > y21 && y > y22) return null;

   List<double> coordinatesPoint = new List<double>();
   coordinatesPoint.Add(x); coordinatesPoint.Add(y);
   return coordinatesPoint;
}

//Отрисовка стрелки
private void DrawArrow(double x, double y, double px, double py, PaintEventArgs e)
{
```

```
            double vx = x - px, vy = y - py;
            double lengthLine = Math.Sqrt((x - px) * (x - px) + (y - py) * (y - py));
            if (lengthLine == 0) return;
            double lx = vx / lengthLine, ly = vy / lengthLine;
            double nx = (py - y) / lengthLine, ny = (x - px) / lengthLine;
            int h = 10, w = 5;
            double Ox = x - h * lx, Oy = y - h * ly;
            double Ax = Ox + nx * w, Ay = Oy + ny * w;
            double Bx = Ox - nx * w, By = Oy - ny * w;

            e.Graphics.DrawLine(myPen, (float)x, (float)y, (float)Ax, (float)Ay);
            e.Graphics.DrawLine(myPen, (float)x, (float)y, (float)Bx, (float)By);
        }
    }
}
```

QueryInterface.cs – здесь содержится общий интерфейс для реализации взаимодействия с различными СУБД, и класс-наследник для СУБД MySQL.

```
namespace InteractionDBMS
{
    public interface IQuery
    {
        DataTable FillTable(string nameDataBases, string nameTable);
        DataTable DefineForeignKey(string nameDataBases, string nameTable);
        List<string> FindDatabases();
        List<string> FindTablesForDatabase(string nameDataBases);
        DataTable Filter(string nameDataBases, string nameTable,
                    Dictionary<string, object> filterValues);
        bool Update(string nameDataBases, string nameTable,
                Dictionary<string, object> values,
                Dictionary<string, object> definitions = null);
        bool Insert(string nameDataBases, string nameTable,
                Dictionary<string, object> values);
        bool Delete(string nameDataBases, string nameTable,
                Dictionary<string, object> definitions);
        DataTable GetColumnsValues(string nameDataBases, string nameTable,
                            List<string> nameColumns);
        Dictionary<string, string> GetTypeColumns(string nameDataBases, string nameTable);
    }

    //Реализация интерфейса для работы с СУБД MySQL
    public class MySqlQuery : IQuery
    {
        private MySqlConnection connection;
        private string connectionString;
        private Logger logger;

        private MySqlQuery(string connectionString)
        {
            logger = LogManager.GetCurrentClassLogger();
            this.connectionString = connectionString;
            connection = new MySqlConnection(connectionString);
        }

        public static MySqlQuery CreateInstance(string connectionString)
        {
            try
            {
                using (MySqlConnection connection = new MySqlConnection(connectionString))
```

```
        {
            connection.Open();
            if (connection.State != ConnectionState.Open)
                throw new Exception();
        }
        return new MySqlQuery(connectionString);
    }
    catch (ArgumentException error)
    {
        Logger logger = LogManager.GetCurrentClassLogger();
        logger.Error("Некорректная строка подключения.");
        MessageBox.Show(error.Message, error.GetType().ToString());
        return null;
    }
    catch (Exception error)
    {
        Logger logger = LogManager.GetCurrentClassLogger();
        logger.Error("Не удалось подключиться к серверу. " + error.Message);
        MessageBox.Show(error.Message, "Не удалось подключиться к серверу");
        return null;
    }
}

//columns - список полей, в которых необходимо проводить выборку;
//definitions - список типа 'столбец-значение',
//использующийся при формировании выражения where
//Если columns не задается, то выборка по всем полям
private DataTable MySqlSELECT(string nameDB, string nameTable,
                        List<string> columns = null,
                        Dictionary<string, object> definitions = null)
{
    DataTable data = new DataTable();
    using (connection)
    {
        try
        {
            connection.Open();
            connection.ChangeDatabase(nameDB);
            MySqlCommand command = connection.CreateCommand();

            command.CommandText = "SELECT ";
            if (columns != null && columns.Count != 0)
                for (int i = 0; i < columns.Count; i++)
                {
                    command.CommandText += "`" + columns.ElementAt(i).ToString()
                                + "`";
                    if (i < columns.Count - 1) command.CommandText += " , ";
                }
            else
                command.CommandText += "*";

            command.CommandText += " FROM `" + nameTable + "`";


            if (definitions != null && definitions.Count != 0)
                for (int i = 0; i < definitions.Count; i++)
                {
                    if (i == 0)
                        command.CommandText += " WHERE ";
                    string key = definitions.ElementAt(i).Key;
                    object value = definitions.ElementAt(i).Value;
```

```
                if (value.ToString() == "")
                    command.CommandText += "(`" + key + "` is NULL OR `" + key
                                + "` = '')";
                else
                {
                    command.CommandText += "`" + key + "` = @" + key;
                    command.Parameters.AddWithValue("@" + key, value);
                }
                if (i < definitions.Count - 1) command.CommandText += " AND ";
            }

            MySqlDataAdapter adapter = new MySqlDataAdapter(command);
            adapter.Fill(data);
            adapter.FillSchema(data, SchemaType.Source);
        }
        catch (MySqlException error)
        {
            logger.Error(error.GetType().Name + ": " + error.Message);
            MessageBox.Show("Сообщение: " + error.Message, error.GetType().Name);
            return null;
        }
        finally
        {
            connection.Close();
        }
    }
    return data;
}

private DataTable MySqlSELECT(string nameDB, string commandText)
{
    DataTable data = new DataTable();

    using (connection)
    {
        try
        {
            connection.Open();
            connection.ChangeDatabase(nameDB);
            MySqlCommand command = connection.CreateCommand();
            command.CommandText = commandText;
            MySqlDataAdapter adapter = new MySqlDataAdapter(command);
            adapter.Fill(data);
            adapter.FillSchema(data, SchemaType.Source);
        }
        catch (MySqlException error)
        {
            logger.Error(error.GetType().Name + ": " + error.Message);
            MessageBox.Show("Сообщение: " + error.Message, error.GetType().Name);
            return null;
        }
        finally
        {
            connection.Close();
        }
    }
    return data;
}

//Найти все базы данных на сервере
public List<string> FindDatabases()
```

```
{
    DataTable dataBases = new DataTable();
    List<string> dataBasesList = new List<string>();

    List<string> columns = new List<string>();
    columns.Add("SCHEMA_NAME");
    dataBases = MySqlSELECT("information_schema", "SCHEMATA", columns);

    if (dataBases == null)
    {
        logger.Trace("Не удалось получить список баз данных.");
        return null;
    }
    foreach (DataRow dataBase in dataBases.Rows)
        dataBasesList.Add(dataBase[0].ToString());
    return dataBasesList;
}

//Найти все таблицы для базы
public List<string> FindTablesForDatabase(string nameDataBases)
{
    List<string> columns = new List<string>();
    columns.Add("TABLE_NAME");
    Dictionary<string, object> definitions = new Dictionary<string, object>();
    definitions.Add("TABLE_SCHEMA", nameDataBases);
    DataTable tables = MySqlSELECT("information_schema", "TABLES", columns,
                                definitions);

    if (tables == null)
    {
        logger.Trace("Не удалось получить список таблиц базы " +
                    nameDataBases + ".");
        return null;
    }
    List<string> tablesList = new List<string>();
    foreach (DataRow table in tables.Rows)
        tablesList.Add(table[0].ToString());
    return tablesList;
}

//Определение типа таблицы
private string EngineDefinition(string nameDataBases, string nameTable)
{
    List<string> columns = new List<string>();
    columns.Add("ENGINE");
    Dictionary<string, object> definitions = new Dictionary<string, object>();
    definitions.Add("TABLE_SCHEMA", nameDataBases);
    definitions.Add("TABLE_NAME", nameTable);
    DataTable nameEngine = MySqlSELECT("information_schema", "TABLES", columns,
                                    definitions);

    if (nameEngine == null)
    {
        logger.Trace("Не удалось получить тип таблицы " + nameTable + ".");
        return null;
    }
    if (nameEngine.Rows.Count != 0) return nameEngine.Rows[0][0].ToString();
    else return null;
}

//Заполнение таблицы всеми данными
```

```
public DataTable FillTable(string nameDataBases, string nameTable)
{
   DataTable fillTable = MySqlSELECT(nameDataBases, nameTable, null, null);

   if (fillTable == null)
   {
      logger.Trace("Не удалось получить записи таблицы " + nameTable + ".");
      return null; ;
   }
   fillTable.TableName = nameTable;
   return fillTable;
}

//Определение внешних ключей
public DataTable DefineForeignKey(string nameDataBases, string nameTable)
{
   DataTable foreignKey = new DataTable();
   string engine = EngineDefinition(nameDataBases, nameTable);

   if (engine == null)
   {
      logger.Trace("Не удалось определить внешние ключи таблицы " +
               nameTable + ".");
      return null;
   }

   if (engine == "InnoDB")
      foreignKey = DefineForeignKey_InnoDB(nameDataBases, nameTable);
   else if (engine == "MyISAM")
      foreignKey = DefineForeignKey_MyISAM(nameDataBases, nameTable);

   if(foreignKey == null)
      logger.Trace("Не удалось определить внешние ключи таблицы " +
               nameTable + ".");
   return foreignKey;
}

//Определение внешних ключей (InnoDB)
private DataTable DefineForeignKey_InnoDB(string nameDataBases, string nameTable)
{
   string commandText = "SELECT `COLUMN_NAME`, `REFERENCED_TABLE_NAME`, " +
      "`REFERENCED_COLUMN_NAME` FROM `KEY_COLUMN_USAGE` WHERE " +
      "`TABLE_SCHEMA` = " + "'" + nameDataBases + "' AND `TABLE_NAME` = " +
      "'" + nameTable + "' AND " +
      "`CONSTRAINT_NAME` <> 'PRIMARY' AND `REFERENCED_TABLE_NAME` is not null";

   DataTable foreignKey = MySqlSELECT("information_schema", commandText);
   return foreignKey;
}

//Определение внешних ключей (MyISAM)
private DataTable DefineForeignKey_MyISAM(string nameDataBases, string nameTable)
{
   DataTable foreignKey = new DataTable();
   foreignKey.Columns.Add("COLUMN_NAME");
   foreignKey.Columns.Add("REFERENCED_TABLE_NAME");
   foreignKey.Columns.Add("REFERENCED_COLUMN_NAME");

   string commandText = "SELECT * FROM `" + nameTable + "` LIMIT 0 ";
   DataTable columns = MySqlSELECT(nameDataBases, commandText);
   if (columns == null) return null;
```

```
    DataColumn[] primaryColumns = columns.PrimaryKey;

    List<string> tablesDB = FindTablesForDatabase(nameDataBases);
    if (tablesDB == null) return null;

    foreach (string regularTable in tablesDB)
    {
        string nameRegularTable = regularTable;
        if (nameTable == nameRegularTable) continue;
        commandText = "SELECT * FROM `" + nameRegularTable + "` LIMIT 0 ";
        DataTable columnsRT = MySqlSELECT(nameDataBases, commandText);
        if (columnsRT == null) return null;
        DataColumn[] primaryColumnsRT = columnsRT.PrimaryKey;
        if (primaryColumnsRT.Count() == 0) continue;

        DataTable foreignKeyTest = foreignKey.Clone();
            foreach (DataColumn primaryColumnRT in primaryColumnsRT)
            foreach (DataColumn column in columns.Columns)
            {
                if (column.ColumnName != primaryColumnRT.ColumnName ||
                    column.DataType != primaryColumnRT.DataType) continue;
                if (primaryColumns.Any(x => x.Caption == column.ColumnName))
                    continue;
                if (foreignKey.Select("COLUMN_NAME = '" + column.ColumnName +
                    "'").Any()) continue;
                foreignKeyTest.Rows.Add(column.ColumnName, nameRegularTable,
                                primaryColumnRT.ColumnName);
                break;
            }
        if (foreignKeyTest.Rows.Count == primaryColumnsRT.Count())
            foreignKey.Merge(foreignKeyTest);
    }
    return foreignKey;
}

//Выборка данных согласно установленным фильтрам
public DataTable Filter(string nameDataBases, string nameTable,
                    Dictionary<string, object> filterValues)
{
    DataTable filterTable = new DataTable();
    if (!filterValues.Any(x => x.Value == null))
        filterTable = MySqlSELECT(nameDataBases, nameTable, null, filterValues);
    else
        filterTable = MySqlSELECT(nameDataBases, "SELECT * FROM `" +
                            nameTable + "` LIMIT 0 ");

    if (filterTable == null)
    {
        logger.Trace("Не удалось получить записи таблицы " + nameTable + ".");
        return filterTable;
    }
    filterTable.TableName = nameTable;
    return filterTable;
}

//Изменить строку в БД
public bool Update(string nameDataBases, string nameTable,
                Dictionary<string, object> values,
                Dictionary<string, object> definitions = null)
{
    using (connection)
```

```
    {
        try
        {
            connection.Open();
            connection.ChangeDatabase(nameDataBases);
            MySqlCommand command = connection.CreateCommand();

            command.CommandText = "UPDATE `" + nameTable + "` SET ";
            if (values != null && values.Count != 0)
                for (int i = 0; i < values.Count; i++)
                {
                    string key = values.ElementAt(i).Key;
                    object value = values.ElementAt(i).Value;
                    if (value.ToString() == "")
                        command.CommandText += "`" + key + "` = NULL";
                    else
                    {
                        command.CommandText += "`" + key + "` = @new" + key;
                        command.Parameters.AddWithValue("@new" + key, value);
                    }
                    if (i < values.Count - 1) command.CommandText += ", ";
                }

            if (definitions != null && definitions.Count != 0)
                for (int i = 0; i < definitions.Count; i++)
                {
                    if (i == 0)
                        command.CommandText += " WHERE ";
                    string key = definitions.ElementAt(i).Key;
                    object value = definitions.ElementAt(i).Value;
                    if (value.ToString() == "")
                        command.CommandText += "(`" + key + "` is NULL OR `" + key
                                    + "` = '')";
                    else
                    {
                        command.CommandText += "`" + key + "` = @old" + key;
                        command.Parameters.AddWithValue("@old" + key, value);
                    }
                    if (i < definitions.Count - 1) command.CommandText += " AND ";
                }

            command.ExecuteNonQuery();
            logger.Trace("Запрос на обновление записи таблицы " + nameTable +
                    " был выполнен.");
            return true;
        }
        catch (MySqlException error)
        {
            logger.Error(error.GetType().Name + ": " + error.Message);
            logger.Trace("Запрос на обновление записи таблицы " + nameTable +
                    " был отклонен.");
            MessageBox.Show("Сообщение: " + error.Message, error.GetType().Name);
            return false;
        }
        finally
        {
            connection.Close();
        }
    }
}
```

```
//Добавить новую строку в БД
public bool Insert(string nameDataBases, string nameTable,
                Dictionary<string, object> values)
{
   using (connection)
   {
      try
      {
         connection.Open();
         connection.ChangeDatabase(nameDataBases);
         MySqlCommand command = connection.CreateCommand();

         command.CommandText = "INSERT INTO `" + nameTable + "` (";
         if (values != null && values.Count != 0)
            for (int i = 0; i < values.Count; i++)
            {
               string key = values.ElementAt(i).Key;
               command.CommandText += "`" + key + "`";
               if (i < values.Count - 1) command.CommandText += ", ";
            }

         command.CommandText += ") VALUES (";

         if (values != null && values.Count != 0)
            for (int i = 0; i < values.Count; i++)
            {
               string key = values.ElementAt(i).Key;
               object value = values.ElementAt(i).Value;
               if (value.ToString() == "")
                  command.CommandText += "NULL";
               else
               {
                  command.CommandText += "@new" + key;
                  command.Parameters.AddWithValue("@new" + key, value);
               }
               if (i < values.Count - 1) command.CommandText += ", ";
            }

         command.CommandText += ")";
         command.ExecuteNonQuery();
         logger.Trace("Запрос на добавление новой записи в таблицу " +
                  nameTable + " был выполнен.");
         return true;
      }
      catch (MySqlException error)
      {
         logger.Error(error.GetType().Name + ": " + error.Message);
         logger.Trace("Запрос на добавление новой записи в таблицу " +
                  nameTable + " был отклонен.");
         MessageBox.Show("Сообщение: " + error.Message, error.GetType().Name);
         return false;
      }
      finally
      {
         connection.Close();
      }
   }
}

//Удалить строку из БД
public bool Delete(string nameDataBases, string nameTable,
```

```
            Dictionary<string, object> definitions)
{
   using (connection)
   {
      try
      {
         connection.Open();
         connection.ChangeDatabase(nameDataBases);
         MySqlCommand command = connection.CreateCommand();

         command.CommandText = "DELETE FROM `" + nameTable + "`";
         if (definitions != null && definitions.Count != 0)
            for (int i = 0; i < definitions.Count; i++)
            {
               if (i == 0)
                  command.CommandText += " WHERE ";
               string key = definitions.ElementAt(i).Key;
               object value = definitions.ElementAt(i).Value;
               if (value.ToString() == "")
                  command.CommandText += "(`" + key + "` is NULL OR `" + key
                              + "` = '')";
               else
               {
                  command.CommandText += "`" + key + "` = @old" + key;
                  command.Parameters.AddWithValue("@old" + key, value);
               }
               if (i < definitions.Count - 1) command.CommandText += " AND ";
            }

         command.ExecuteNonQuery();
         logger.Trace("Запрос на удаление записи таблицы " + nameTable +
                  " был выполнен.");
         return true;
      }
      catch (MySqlException error)
      {
         logger.Error(error.GetType().Name + ": " + error.Message);
         logger.Trace("Запрос на удаление записи таблицы " + nameTable +
                  " был отклонен.");
         MessageBox.Show("Сообщение: " + error.Message, error.GetType().Name);
         return false;
      }
      finally
      {
         connection.Close();
      }
   }
}

//Значения конкретных столбцов
public DataTable GetColumnsValues(string nameDataBases, string nameTable,
                        List<string> nameColumns)
{
   DataTable ColumnsValues = MySqlSELECT(nameDataBases, nameTable, nameColumns,
                              null);
   if(ColumnsValues == null)
      logger.Trace("Не удалось получить значения ключей таблицы " +
               nameTable + ".");
   return ColumnsValues;
}
```

```
        //Типы столбцов
        public Dictionary<string, string> GetTypeColumns(string nameDataBases,
                                            string nameTable)
        {
            Dictionary<string, string> columnsTypeDictionary =
                                    new Dictionary<string, string>();

            List<string> columns = new List<string>();
            columns.Add("COLUMN_NAME");
            columns.Add("DATA_TYPE");
            Dictionary<string, object> definitions = new Dictionary<string, object>();
            definitions.Add("TABLE_SCHEMA", nameDataBases);
            definitions.Add("TABLE_NAME", nameTable);
            DataTable columnsType = MySqlSELECT("information_schema", "COLUMNS", columns,
                                        definitions);
            if (columnsType == null)
            {
                logger.Trace("Не удалось получить типы столбцов таблицы " +
                        nameTable + ".");
                return null;
            }

            foreach (DataRow columnType in columnsType.Rows)
            {
                string columnName = columnType["COLUMN_NAME"].ToString();
                string type = columnType["DATA_TYPE"].ToString();

                if (type == "tinyblob" || type == "longblob" || type == "blob"
                    || type == "mediumblob")
                    columnsTypeDictionary.Add(columnName, "blob");
                else if (type == "tinytext" || type == "longtext" || type == "text"
                        || type == "mediumtext")
                    columnsTypeDictionary.Add(columnName, "text");
                else
                    columnsTypeDictionary.Add(columnName, "other");
            }
            return columnsTypeDictionary;
        }
    }
}
```

QueryFactory.cs – класс, реализующий паттерн «Простая фабрика».

```
namespace InteractionDBMS
{
    //Определяет, объект какого типа создать
    public static class Query
    {
        public static IQuery query;
        public static string login = "", DBMS = "", server = "";

        public static void Build(string serv, string log, string pass,
            string nameDBMS, string connectionString)
        {
            server = serv; login = log; DBMS = nameDBMS;
            try
            {
                switch (DBMS)
                {
                    case "MySQL":
```

```
                connectionString += ";database=information_schema";
                query = MySqlQuery.CreateInstance(connectionString);
                break;
            default:
                query = null;
                throw new Exception();
        }
      }
      catch (Exception)
      {
        Logger logger = LogManager.GetCurrentClassLogger();
        string mes = "Отсутствует реализация класса Query для работы " +
              "с СУБД " + DBMS;
        logger.Fatal(mes);
        MessageBox.Show(mes, "Не удалось создать объект");
      }
    }
  }
}
```