АЛЕКСЕЕВ Виктор Федорович - кандидат технических наук, доцент кафедры проектирования информационно-компьютерных систем Белорусского государственного университета информатики и радиоэлектроники.
Закончил в 1977 году Минский радиотехнический институт.
Автор более 400 публикаций, в том числе: 2 монографии, 2 авторских свидетельства на изобретения, 2 патента Республики Беларусь.
Осуществляет подготовку аспирантов, магистрантов и студентов.

БОГАТКО Иван Николаевич - магистр техники и технологии, начальник отдела информационных технологий и защиты информации учреждения «Главный информационно-аналитический центр Министерства образования Республики Беларусь».
Закончил в 2014 году Белорусский государственный университет информатики и радиоэлектроники.
Автор более 15 публикаций.
Осуществляет подготовку студентов.

ПИСКУН Геннадий Адамович - кандидат технических наук, доцент кафедры проектирования информационно-компьютерных систем Белорусского государственного университета информатики и радиоэлектроники.
Закончил в 2008 году Белорусский государственный университет информатики и радиоэлектроники.
Автор около 100 публикаций, в том числе: 1 монография, 2 патента Республики Беларусь.
Осуществляет подготовку магистрантов и студентов.

В.Ф. АЛЕКСЕЕВ
И.Н. БОГАТКО
Г.А. ПИСКУН

СТРУКТУРЫ И БАЗЫ ДАННЫХ

ПОСОБИЕ ДЛЯ КУРСОВОГО ПРОЕКТИРОВАНИЯ

Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

В. Ф. Алексеев, И. Н. Богатко, Г. А. Пискун

СТРУКТУРЫ И БАЗЫ ДАННЫХ.

ПОСОБИЕ ДЛЯ КУРСОВОГО ПРОЕКТИРОВАНИЯ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники для специальности 1-39 03 02 «Программируемые мобильные системы» в качестве пособия

Минск БГУИР 2017

УДК 004.65(076)
ББК 32.973.26-018.2я73
А47

Р е ц е н з е н т ы:

кафедра программного обеспечения вычислительной техники и автоматизированных систем Белорусского национального технического университета (протокол №11 от 09.06.2016);

заведующий кафедрой здорового образа жизни учреждения образования «Белорусская государственная академия связи», доктор технических наук, профессор В. И. Курмашев

Алексеев, В. Ф.

А47 Структуры и базы данных. Пособие для курсового проектирования : пособие / В. Ф. Алексеев, И. Н. Богатко, Г. А. Пискун. – Минск : БГУИР, 2017. – 84 с. : ил.

ISBN 978-985-543-338-6.

Приводятся общие требования к курсовому проекту и методические рекомендации по выполнению разделов пояснительной записки и графической части проекта. Пособие может быть использовано студентами, магистрантами, аспирантами и специалистами, занимающимися разработкой баз данных.

УДК 004.65(076)
ББК 32.973.26-018.2я73

ISBN 978-985-543-338-6

СОДЕРЖАНИЕ

1 ОБЩИЕ ТРЕБОВАНИЯ К КУРСОВОМУ ПРОЕКТУ

1.1 Цель курсового проектирования

Целью курсового проектирования является получение практических навыков по проектированию, разработке и введению в эксплуатацию базы данных (БД) с учетом выбора оптимальной системы управления базой данных, проектирования основных запросов и выбора или разработки клиентского программного обеспечения для работы с БД. Для достижения поставленной цели выполняется изучение предметной области с последующей формализацией, анализ информационных потребностей клиентского программного обеспечения (ПО), осуществляется выбор программного интерфейса между клиентским ПО и БД, выполняется проектирование, программирование и начальное наполнение базы данных, а также проектируются основные запросы к БД от нескольких групп потенциальных пользователей.

Результатом выполнения курсового проекта является комплект документов, дающий ответ о содержании проектного решения.

1.2 Тематика курсового проектирования

Тематика курсового проектирования должна быть актуальной, соответствовать современному состоянию и перспективам развития науки, техники и образования.

Темой курсового проектирования является разработка БД и подключаемого к БД клиентского ПО, предназначенных для поддержки работы одной или нескольких предметных областей. Варианты предметной области и исходные данные для разработки БД представлены в приложении А.

1.3 Структура и содержание курсового проекта

Содержание и объем курсового проекта определяются кафедрой. Курсовой проект включает:

– задание на курсовое проектирование;

– пояснительную записку;

– графическую часть.

Задание на курсовое проектирование должно полностью отражать процесс разработки БД и приложения к ней. Задание на курсовое проектирование должно содержать:

– исходные данные к проекту, включающие в себя общие требования к проектируемой БД, используемым программным средствам и использованной системе управления базами данных (СУБД), языку программирования клиентского приложения, требования, предъявляемые к предметной области и информационным потребностям пользователей, а также другие специальные требования;

– содержание расчетно-пояснительной записки, представленное в виде перечня подлежащих разработке вопросов;

– перечень обязательного графического материала;

– календарный график работы над проектом на весь период проектирования.

Пример оформления задания по курсовому проекту представлен в приложении Б.

Пояснительная записка (ПЗ) содержит 20–30 страниц машинописного текста.

Графическая часть (ГЧ) курсового проекта должна обязательно включать:

– структуру проектируемой базы данных на одном листе формата А1 (для реляционной базы данных это диаграмма «сущность – связь»);

– *IDEF0/IDEF3*-диаграммы, отражающие бизнес-модели процессов и предназначенные для описания процессов и функций системы в предметной области базы данных.

Графическую часть курсового проекта рекомендуется разрабатывать с помощью наиболее современных *CASE*-средств для проектирования и документирования баз данных, а также с помощью графических редакторов (PhotoShop, CorelDraw, VisioPro, Adobe Illustrator и др.).

1.4 Структура и содержание пояснительной записки

Общими требованиями по оформлению пояснительной записки к курсовому проекту являются: четкость и логическая последовательность изложения материала, убедительность аргументации, краткость и ясность формулировок, исключающих неоднозначность толкования, конкретность изложения результатов, доказательств и выводов. Пояснительная записка к курсовому проекту должна в краткой и четкой форме раскрывать творческий замысел проекта, содержать принятые методы проектирования и разработки, сопровождаться иллюстрациями: изображениями рабочей программы, графиками, эскизами, диаграммами, схемами и т. п. Пояснительная записка включает следующие разделы:

1 Титульный лист.

2 Реферат.

3 Задание по курсовому проекту.

4 Содержание.

5 Перечень условных обозначений, символов и терминов.

6 Введение.

7 Анализ предметной области и ее формализация для проектирования базы данных.

8 Проектирование базы данных для основного вида деятельности рассматриваемой предметной области.

9 Применение разработанной базы данных.

10 Заключение.

11 Приложения.

12 Список использованных источников.

2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ЕЕ ФОРМАЛИЗАЦИЯ ДЛЯ ПРОЕКТИРОВАНИЯ БАЗЫ ДАННЫХ

2.1 Описание предметной области

Предметная область – часть реального мира, подлежащая изучению с целью организации управления и в конечном итоге автоматизации. Она содержит в себе информацию, которую необходимо формализовать для внесения в базу данных.

В результате формализации информации о предметной области мы получаем данные в виде сущностей и их атрибутов.

Данные – это информация, зафиксированная в некоторой форме, пригодной для последующей обработки, передачи и хранения, например, находящаяся в памяти ЭВМ или подготовленная для ввода в ЭВМ.

Сущность – это выделенный на концептуальном уровне объект для базы данных. Сущности представляют собой таблицы в базе данных, содержащие определенные данные, называемые атрибутами. Примерами сущностей являются отдельный студент, группа студентов, аудитория, время занятий, слова, числа, символы.

Экземпляр сущности – это конкретный представитель данной сущности.

Атрибуты сущностей содержат данные, внесенные пользователем или администратором. Для реляционных баз данных атрибуты представляются в виде столбцов таблиц.

Запись в БД представляет собой данные, записанные в столбцы определенной таблицы.

Анализ предметной области состоит из ее подробного описания, выделения полезной информации и ее последующей формализации в данные.

Описание предметной области представляет собой выявление полезной информации, ключевых закономерностей и правил, которые распространяются в ней. Формализация предметной области есть определение совокупности сущностей и их атрибутов, присущих именно этой предметной области, с последующим определением взаимосвязей между ними.

2.2 Анализ информационных потребностей пользователей и предварительное описание запросов

В процессе выполнения анализа информационных потребностей необходимо выполнить следующее:

– определить группы потенциальных пользователей базой данных;

– сформулировать ограничения и привилегии, которые будут присваиваться каждой группе, на предоставление, внесение, обновление, изменение и удаление данных.

Предварительное описание запросов заключается в определении всех потенциальных запросов от вышеописанных групп пользователей, обращающихся к хранимым в базе данным, краткое пояснение типа действия (изменение, обновление, добавление), совершаемого над этими данными, с указанием сущностей или атрибутов, с которыми эти действия совершаются.

Например, для базы данных новостного интернет-портала могут существовать следующие группы пользователей: администратор, журналист и посетитель сайта. Для каждой группы пользователей будут существовать свои ограничения на использование данных, за исключением администратора базы данных, который имеет полную свободу действий и может проводить с базой данных любые манипуляции.

Журналист, например, имеет ограничение на изменение и удаление данных, которые были внесены посетителями сайта. Этими данными может быть оценка статьи, выставленная посетителем и записываемая с целью определения рейтинга журналиста. При этом ему предоставлена привилегия обновлять ту информацию, которая внесена именно им.

Посетитель сайта в свою очередь будет иметь ограничение на изменение, обновление и удаление статей.

2.3 Определение требований и ограничений к базе данных с точки зрения предметной области

База данных представляет собой совокупность структурированных взаимосвязанных данных, относящихся к определенной предметной области и организованных для решения определенных задач разными пользователями.

Особенности каждой предметной области могут накладывать разнообразные требования и ограничения на хранимые в базе данные. Наиболее часто встречаются следующие:

– требование уникальности записей в определенных атрибутах (например, номер автомобиля не должен повторяться, тогда как другая информация может быть одинаковой);

– ограничения на неопределенность данных (обязательность указания владельца автомобиля, т. к. не может быть никому не принадлежащей машины);

– данные, хранимые в одном атрибуте для определенной записи, не могут быть больше, меньше (позже, раньше) или равными данным, хранимым в другом атрибуте (например, дата постановки на учет автомобиля не может быть раньше даты ввоза на территорию страны).

2.4 Постановка решаемой задачи

В процессе постановки решаемой в ходе курсового проектирования задачи необходимо:

– уяснить цель курсового проектирования с учетом выданного варианта предметной области;

– указать задачи, которые планируется решить для достижения поставленной цели;

– кратко описать планируемые результаты для каждой задачи с указанием сроков выполнения.

3 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ ДЛЯ ОСНОВНОГО ВИДА ДЕЯТЕЛЬНОСТИ РАССМАТРИВАЕМОЙ ПРЕДМЕТНОЙ ОБЛАСТИ

3.1 Разработка инфологической модели предметной области базы данных

В этом разделе будет рассмотрен один из способов создания полноценной базы данных.

Проектирование БД – одна из наиболее сложных и ответственных задач, связанных с созданием информационной системы. В результате решения этой задачи должны быть определены содержание БД, эффективный для всех ее будущих пользователей способ организации данных и инструментальные средства управления данными.

Цель инфологического моделирования – обеспечение наиболее естественных для человека способов сбора и представления той информации, которую предполагается хранить в создаваемой базе данных. Храниться данные будут в диаграмме «сущность – связь», которая будет создана с помощью СУБД.

Связь – это некоторая ассоциация между двумя сущностями. Одна сущность может быть связана с другой или сама с собою.

Диаграмма «сущность – связь» (*ER*-диаграмма) позволяет графически представить все элементы информационной модели согласно простым, интуитивно понятным, но строго определенным правилам – нотациям.

Прежде чем проектировать БД, необходимо привести данные к третьей нормальной форме и ответить на два вопроса:

– какой атрибут будет являться первичным ключом;

– какие атрибуты будут являться внешними ключами.

Нормальная форма (НФ) представляет собой ограничение на схему базы данных, вводимое с целью устранения определенных нежелательных свойств при выполнении реляционных операций.

Отношение находится в первой нормальной форме (1НФ), если все атрибуты отношения являются атомарными, т. е. не имеют компонентов. Иными словами, не должно быть повторений атрибутов в одной сущности.

Отношение находится во второй нормальной форме (2НФ), если оно находится в первой нормальной форме (1НФ) и при этом любой его атрибут, не входящий в состав первичного ключа, функционально полно зависит от первичного ключа. Другими словами, суть 2НФ заключается в том, чтобы каждый атрибут сущности зависел от первичного ключа данной сущности. Если какой-либо атрибут не зависит от первичного ключа, то создается новая сущность с этим атрибутом, которую можно связать с первичным ключом изначальной сущности.

Отношение находится в третьей нормальной форме (3НФ), если оно находится во второй нормальной форме (2НФ) и при этом любой его неключе-

вой атрибут нетранзитивно зависит от первичного ключа. Проще говоря, второе правило требует выносить все неключевые поля, содержимое которых может относиться к нескольким записям таблицы, в отдельные таблицы.

Первичный ключ представляет собой один из примеров уникального индекса и применяется для уникальной идентификации записей таблицы. Никакие две записи таблицы не могут иметь одинаковые значения первичного ключа. Первичный ключ обычно сокращенно обозначают как *PK* (*Primary Key*). Название первичного ключа должно быть уникальным и не совпадать ни с одним атрибутом других сущностей, а также со структурами запросов и названием привилегий.

В реляционных БД практически всегда разные таблицы логически связаны друг с другом. Первичные ключи как раз используются для однозначной организации такой связи.

Внешние ключи используются главным образом для проверки целостности данных, а не для объединения таблиц, как принято считать, однако в нашем случае внешний ключ будет применяться для связи между таблицами.

Внешние ключи – это как раз то, что делает реляционные базы реляционными, те связующие цепочки, которые связывают таблицы между собой. Они позволяют вам «разместить» покупателей в одной таблице, заказы – в другой, а товары из этих заказов – в третьей, таким образом в базе минимизируется избыточность данных. Чем меньше избыточных данных – тем больше у вас шансов сохранить целостность данных (две или более противоречащие друг другу записи – это всегда плохо). Название внешнего ключа одной сущности может совпадать с названием первичного ключа другой сущности, с которой идет связь, но лучше всего, чтобы название атрибута внешнего ключа было уникальным и не совпадало ни с одним атрибутом других сущностей, а также со структурами запросов и названием привилегий.

Названия всех сущностей и атрибутов необходимо записывать без пробелов или через нижнее подчеркивание, иначе могут возникнуть проблемы при реализации запросов. Если названия сущностей и атрибутов не подпадают под правило и на этапе проектирования запросов командная строка *MySQL* не определяет данные таблицы как существующие, то при запросе к каждой сущности и атрибуту необходимо добавлять с двух сторон обратные кавычки (`), которые ставятся клавишей тильда (~) на английской раскладке.

Сущность на *ER*-диаграмме представляется прямоугольником с именем в верхней части (рисунки 3.1 и 3.2). Для наименования всех элементов модели: сущности, атрибутов, записи в атрибутах, а также названия БД можно использовать как русский, так и английский язык, данные будут выводиться корректно в обоих случаях. Однако, чтобы постоянно не переключаться с английской раскладки на русскую, лучше использовать только английский язык, поскольку все команды и запросы в командной строке *MySQL* (далее просто командная строка, поскольку командная строка *Windows* нам практически не пригодится) бу-

дут производиться на английском языке. Кроме этого, таким образом можно избежать различного рода проблем на этапе создания клиентского приложения.

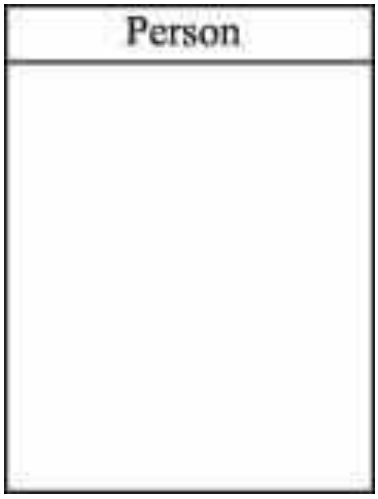

Рисунок 3.1 – Представление сущности *Person* (персонал) на *ER*-диаграмме

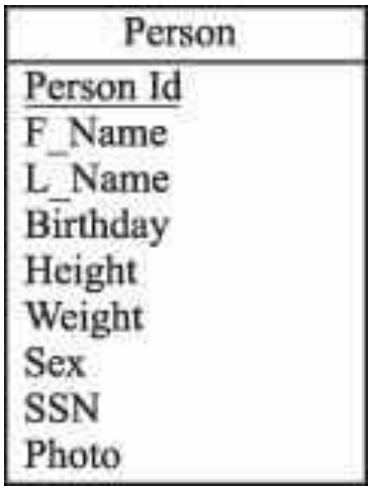

Рисунок 3.2 – Представление сущности *Person* с атрибутами
и уникальным идентификатором сущности

В прямоугольнике перечисляются атрибуты сущности, при этом атрибуты, составляющие уникальный идентификатор сущности, т. е. первичный ключ, подчеркиваются.

Итак, мы должны окончательно определить, какие данные будут храниться в нашей БД, и переходить к этапу переноса сущностей и атрибутов с бумаги в СУБД.

Системой управления базами данных называют программную систему, предназначенную для создания на ЭВМ общей базы данных для множества приложений, поддержания ее в актуальном состоянии и обеспечения эффективного доступа пользователей к содержащимся в ней данным в рамках предоставленных им полномочий. Для этого будем использовать инструмент *MySQL Workbench*, который свободно распространяется в сети Интернет.

После установки данного продукта могут возникнуть некоторые проблемы, результатом чего является неустановленная программа либо компонент программы. Это возникает из-за того, что программе требуются дополнительные библиотеки (либо программы), без которых тот или иной компонент программы не может быть установлен, как это продемонстрировано на рисунке 3.3.

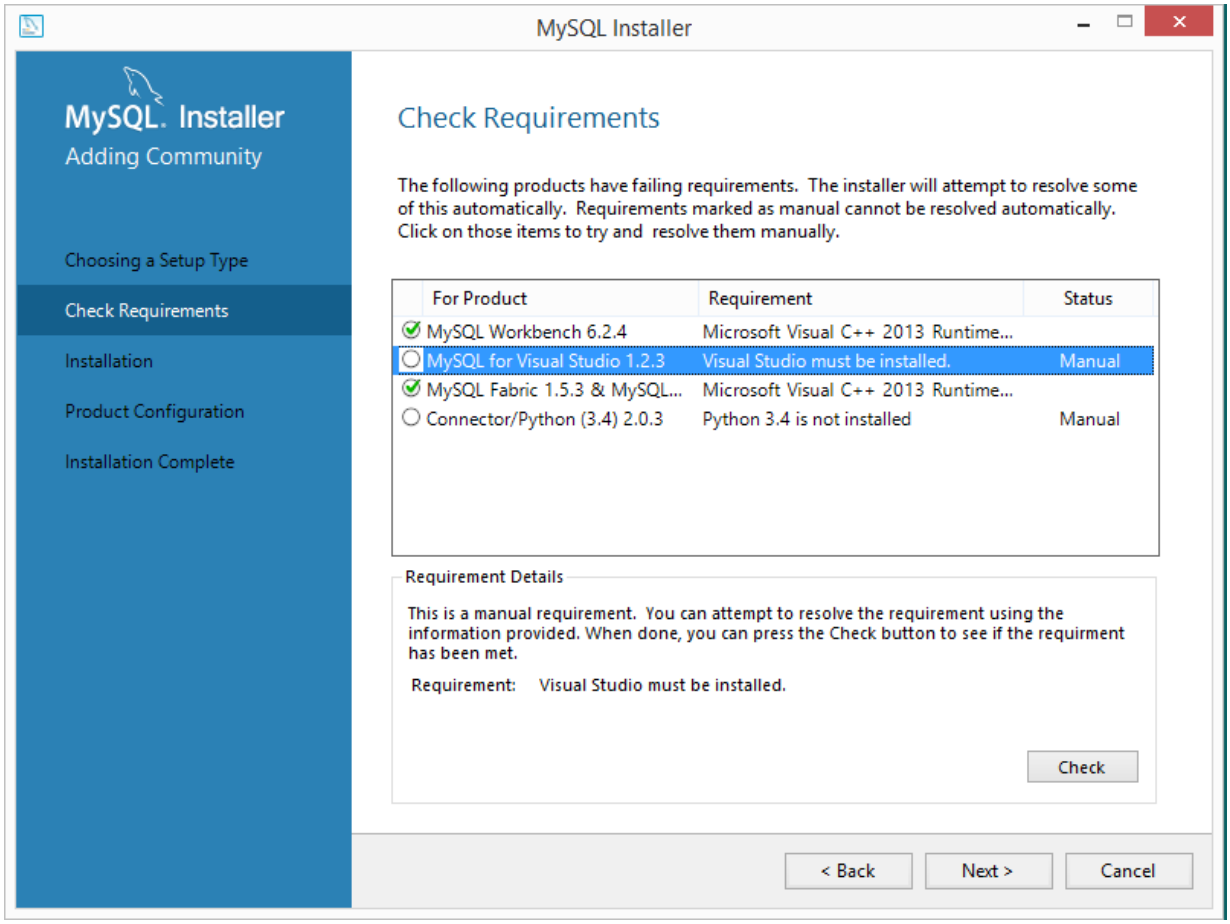

Рисунок 3.3 – Проверка требований при установке *MySQL Workbench*

Из рисунка 3.3 видно, что при невыполнении требований два компонента, помеченные галочкой, будут установлены, а непомеченные, соответственно, не будут. В случае, если требования не были выполнены и нужные компоненты не установлены, для добавления компонентов нет необходимости переустанавливать программу, достаточно лишь установить недостающие для компонентов библиотеки, запустить компонент *MySQL Installer* и нажать кнопку «*Add …*» (рисунок 3.4).

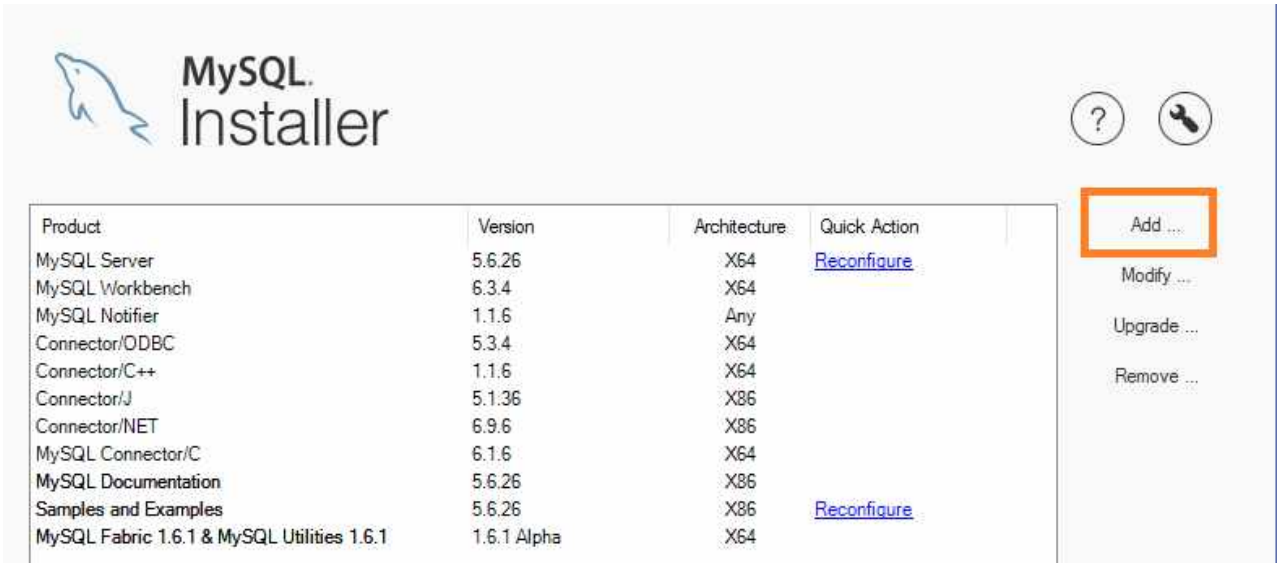

Рисунок 3.4 – Добавление компонентов

В следующем окне будет показан список доступных компонентов. Выбираем нужный компонент, переносим в правую колонку и устанавливаем.

В процессе установки мы добавляем пользователя и создаем для него логин и пароль, которые будут использоваться для авторизации в *MySQL Server* (рисунок 3.5), а также *MySQL Root Password*, применяемый для входа в командную строку. Лучше всего использовать один и тот же пароль в обоих случаях.

Рисунок 3.5 – Подключение к *MySQL Server*

По умолчанию каждый день ровно в полночь *MySQL Server* проходит проверку, исходя из которой сервер будет работать либо прекратит свою работу. Прекращение работы может случиться, например, при отсутствии Интернета несколько дней подряд в момент прохождения проверки. Если это произошло и сервер не запускается, то снова заходим в *MySQL Installer*, напротив *MySQL Server* нажимаем кнопку *Reconfigure* и следуем подсказкам мастера настройки.

После решения основных проблем с установкой программы переходим к ее запуску и ознакомлению с интерфейсом. Первое окно программы не содержит важной информации, поэтому переходим на вкладку «*File -> New Model*».

Мы можем поменять название БД с «*mydb*» на любое другое (рисунки 3.6 и 3.10). Чтобы далее не было никаких проблем, лучше всего переименовать БД перед созданием таблиц. Кроме того, можно сразу добавить таблицы (сущности) с атрибутами (рисунок 3.7) и на *ER*-диаграмме просто вынести их на рабочее пространство, просмотреть, что значат некоторые типы данных и многое другое. Для начала работы с диаграммой «сущность – связь» щелкнем дважды кнопкой мыши на «*Add Diagram*» (рисунок 3.8).

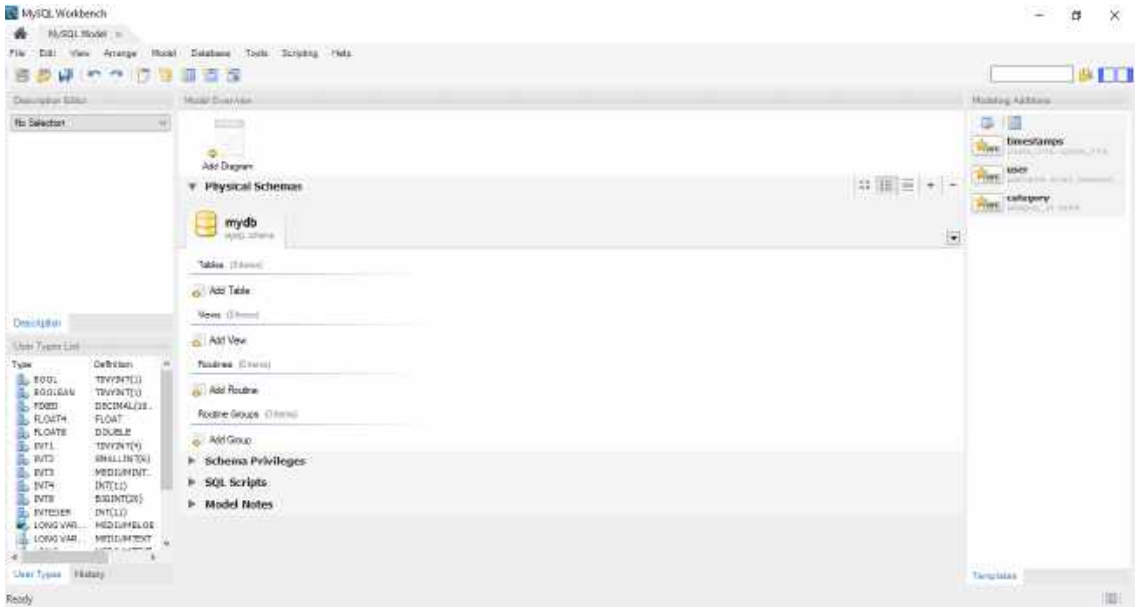

Рисунок 3.6 – Окно создания диаграммы

После добавления диаграммы рекомендуется сразу же сохранить ее у себя на диске. Самое важное: при сохранении диаграммы в пути не должно содержаться русских букв, иначе случится «вылет» программы (рисунок 3.9) и не сохранится ничего, даже **.bak*-файл, который создается как резервный файл после последнего сохранения (с помощью него можно вернуться к исходному файлу). Если же диаграмма была ранее сохранена на диске, но открыта по пути, где содержатся русские символы, то в случае сохранения снова случится «вылет» программы, но теперь будет создан **.bak*-файл в том же месте, где ранее был файл с диаграммой, который сохранит последние изменения в файле и поможет не потерять сделанные ранее изменения.

Возможно, это упущение исправят в следующих версиях программы, однако сейчас эта проблема есть, и она решается только путем сохранения файлов, когда в пути содержатся лишь английские символы.

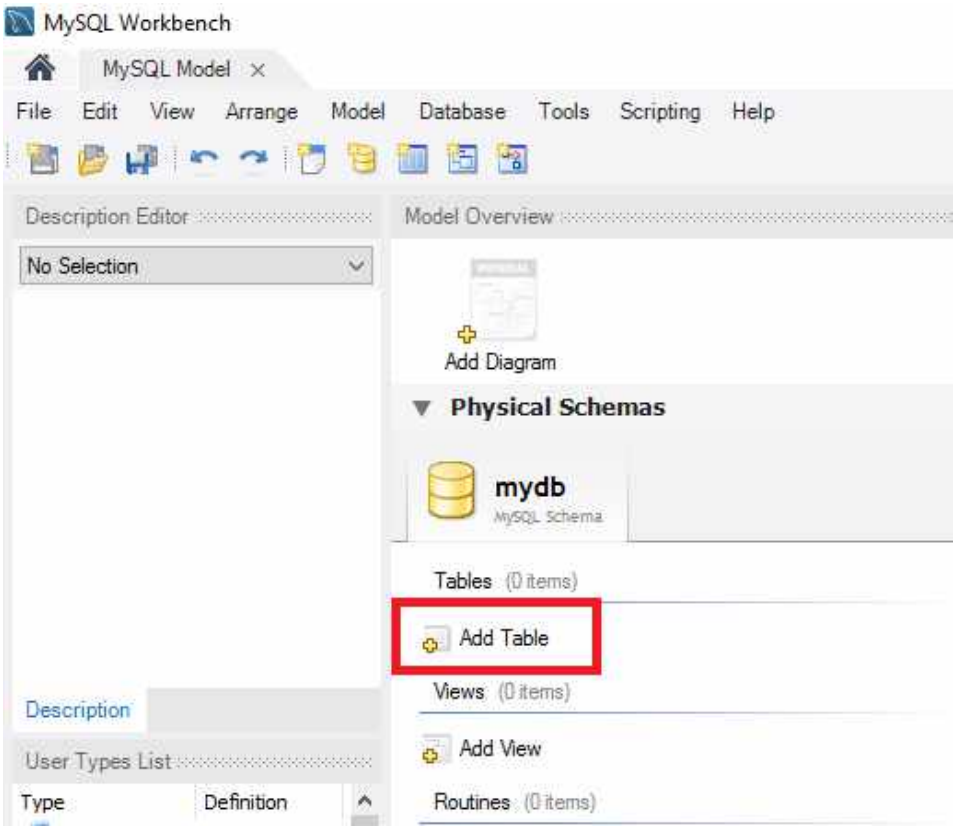

Рисунок 3.7 – Добавление таблиц без рабочей области

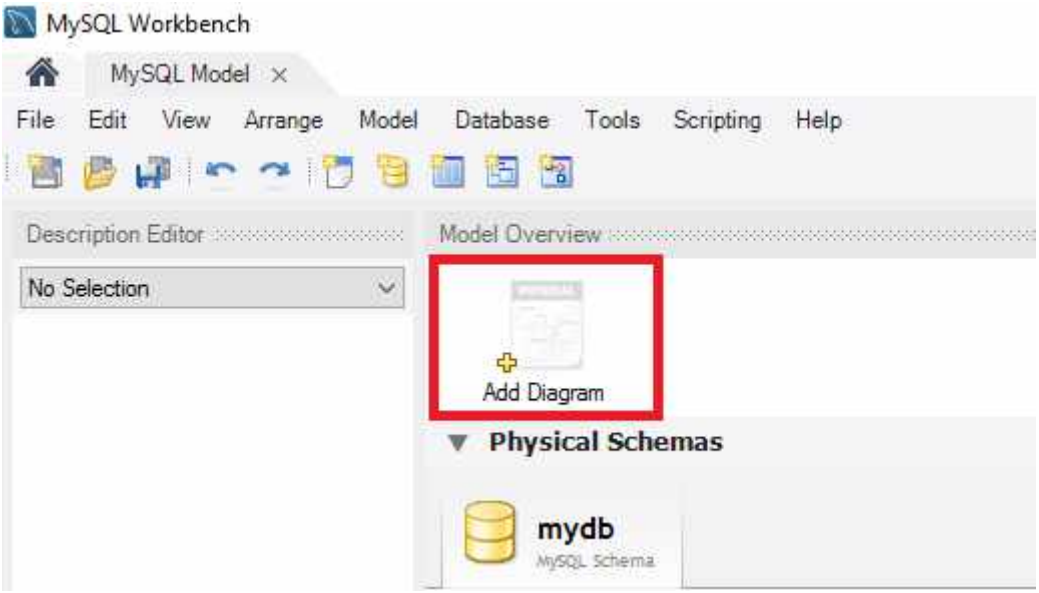

Рисунок 3.8 – Добавление к проекту *ER*-диаграммы

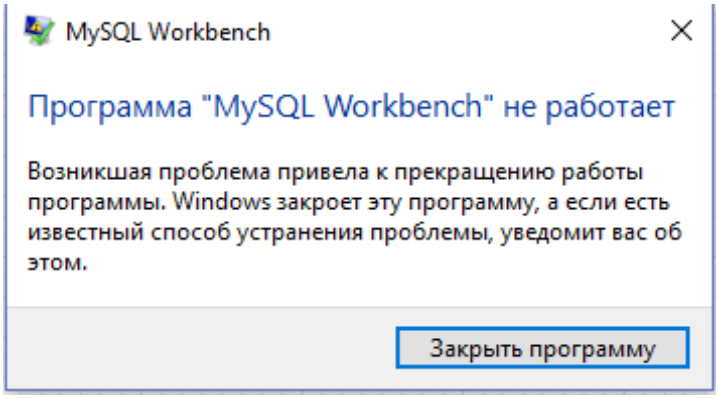

Рисунок 3.9 – Результат при сохранении файла,
в пути которого содержатся русские символы

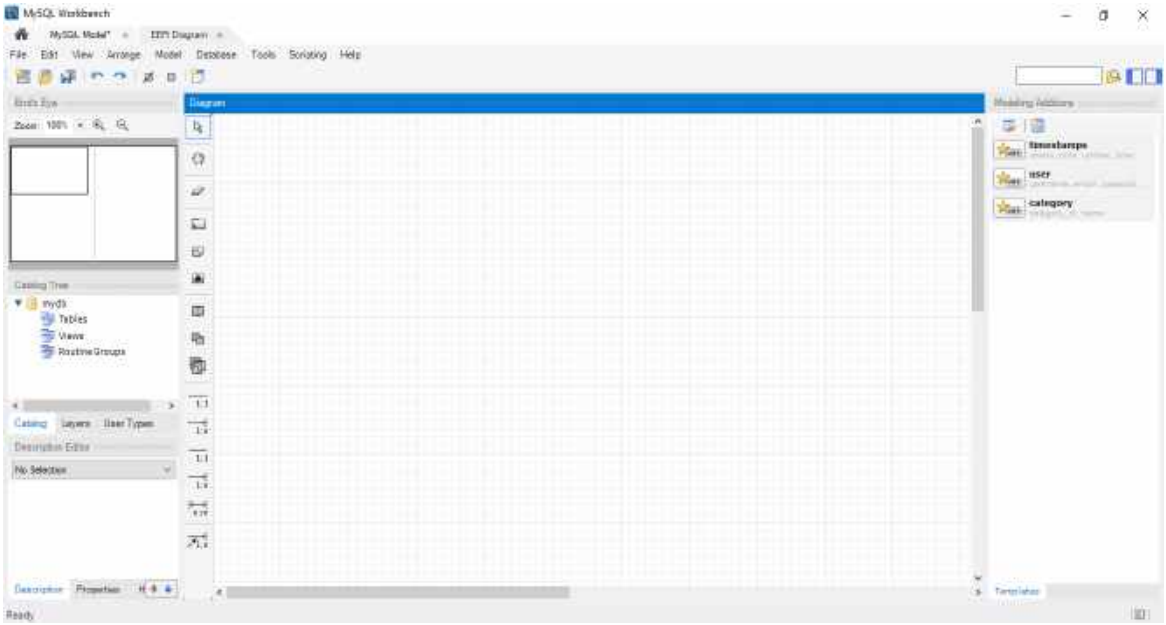

Рисунок 3.10 – Окно проектирования диаграммы «сущность – связь»

В окне проектирования диаграммы можно осуществить множество различных действий:

– создание новых таблиц (рисунок 3.11);

– редактирование существующих таблиц;

– перенос таблиц, созданных в предыдущем окне, на рабочую область (рисунок 3.12);

– изменение названия БД;

– изменение свойств таблицы: цвета, ширины, высоты и т. д.;

– добавление слоев для более красочного и структурированного представления данных (рисунок 3.13);

– создание текстовых заметок;

– добавление изображений на рабочую область и т. д.

Для внесения спроектированных таблиц в программную среду *MySQL Workbench* используем инструмент «*Place a New Table*» (или горячую

клавишу *T*), названием которого являлась сущность, а внесенные в эту таблицу данные – атрибутами.

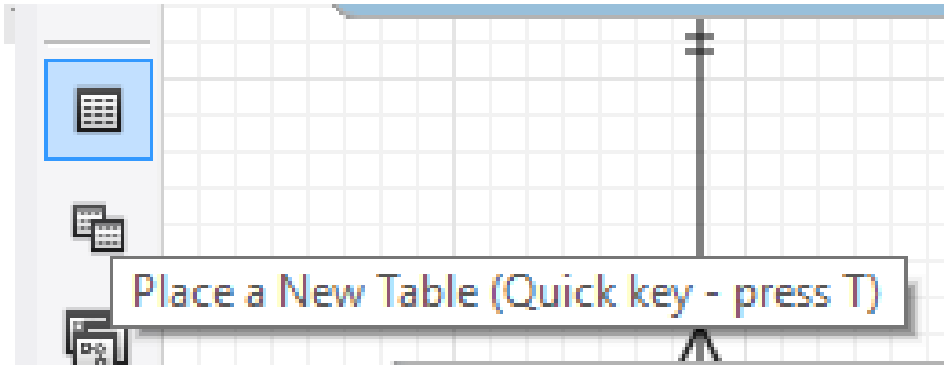

Рисунок 3.11 – Добавление таблицы

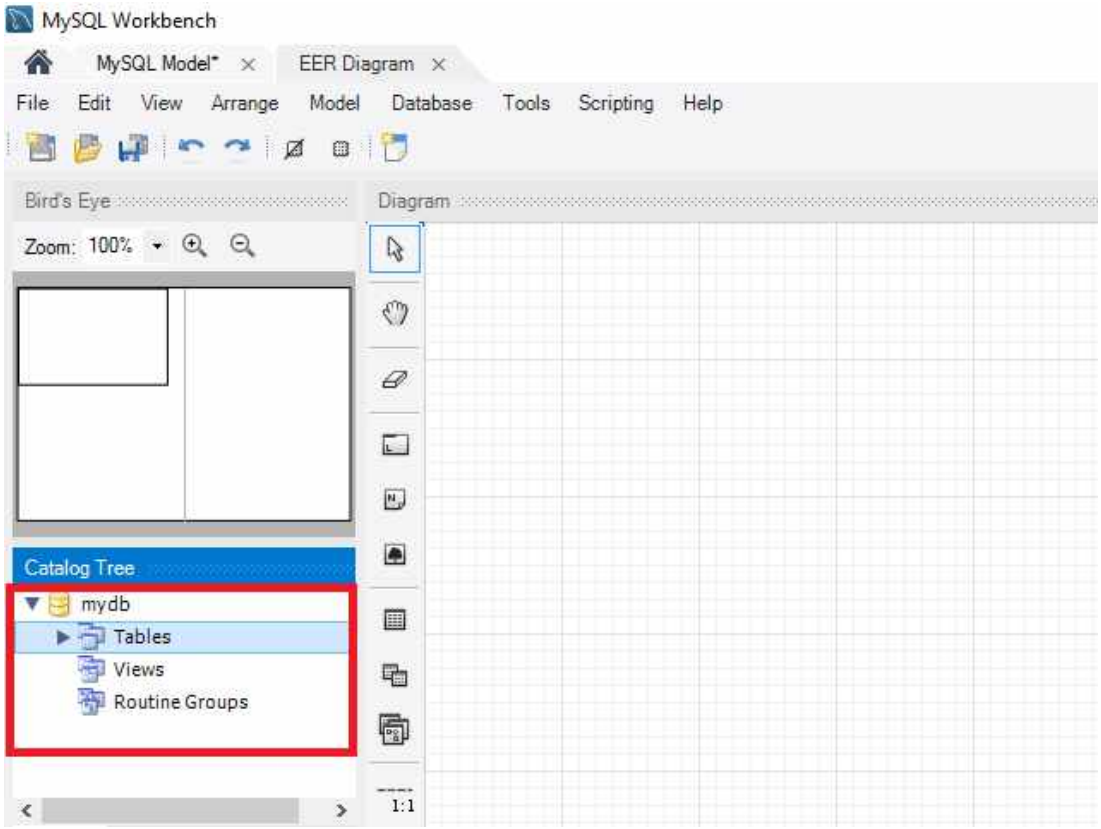

Рисунок 3.12 – Перенос созданных ранее таблиц на рабочую область

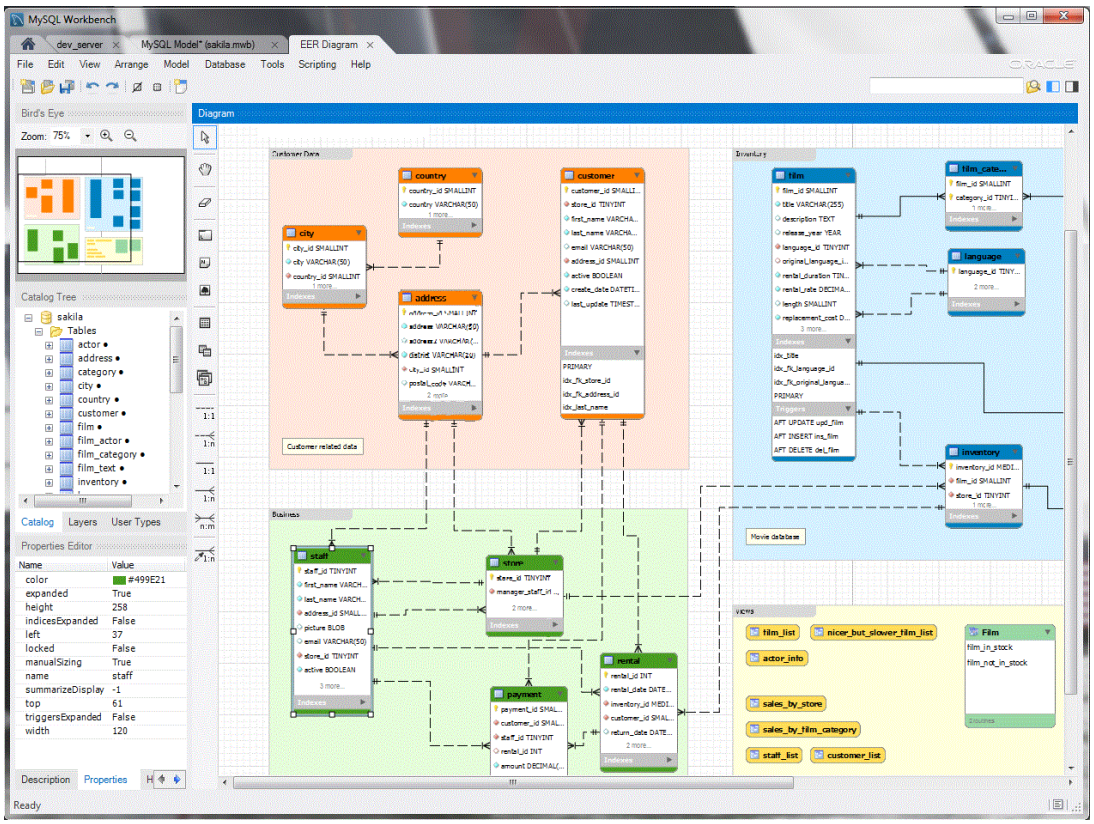

Рисунок 3.13 – Использование слоев

После добавления таблицы на рабочее пространство дважды щелкаем на ней кнопкой мыши и появляется окно, представленное на рисунке 3.14.

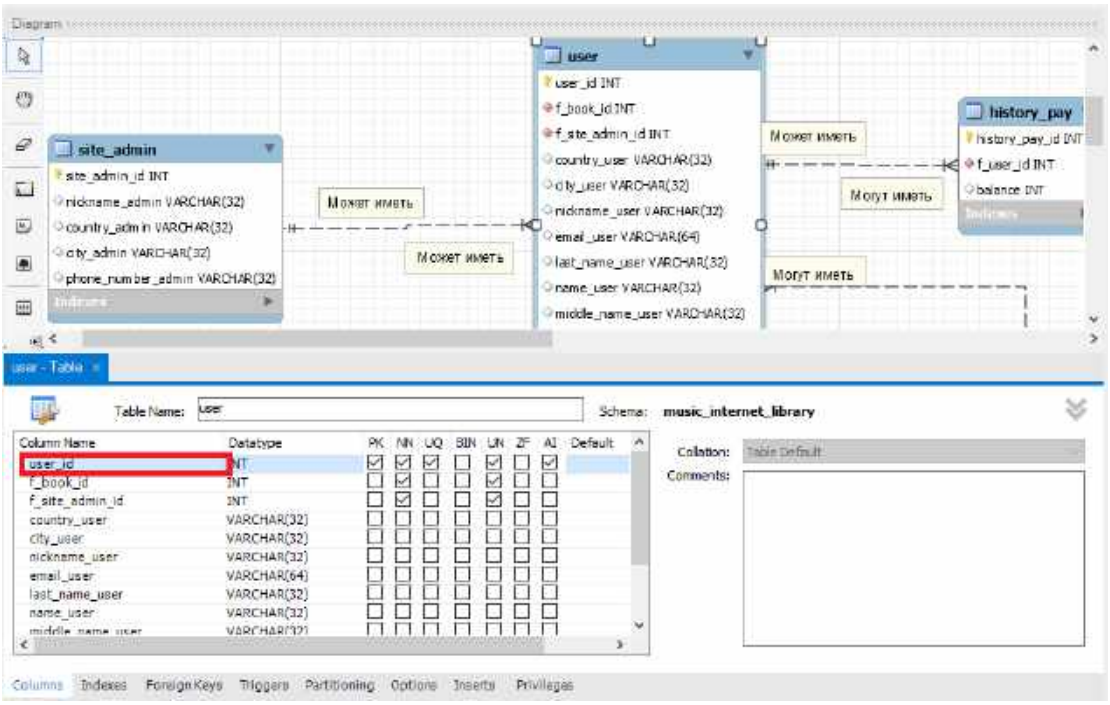

Рисунок 3.14 – Свойства таблицы

Для изменения размера рабочей области открываем вкладку «*Model -> Diagram Properties and Size…*» и выбираем нужный нам размер.

Поскольку программная среда *MySQL Workbench* изначально делит область на некоторые фрагменты, она сохраняет диаграмму на разных листах в зависимости от количества областей. Для экспорта *PDF*-файла (в случае возможности распечатки дома), на котором диаграмма расположена на одной странице, открываем «*File -> Export -> Export as Single Page PDF…*».

Переходим к переносу таблиц с бумажного носителя в программную среду. Щелкая дважды кнопкой мыши на месте, которое выделено в первой строке, создаем наш первый атрибут, который будет являться первичным ключом для сущности (этот атрибут можно изменить, поставив галочку напротив другого атрибута). Таким образом заполняем наши таблицы атрибутами. Напомним, что в одной сущности может присутствовать лишь один первичный ключ.

В процессе переноса будет предлагаться выбор используемых типов данных и выбор привилегий (ограничений) для каждого атрибута. На данный момент можно оставлять предложенные программой типы данных и привилегии. В подразделе 3.2 будет более подробно рассказано, какие существуют типы данных (диапазон, т. е. возможное количество записей в атрибуте, и описание типов данных) и какими привилегиями может обладать атрибут сущности. На рисунках 3.15 и 3.16 выделены соответственно элементы, в которых можно изменить типы данных и привилегии для каждого атрибута.

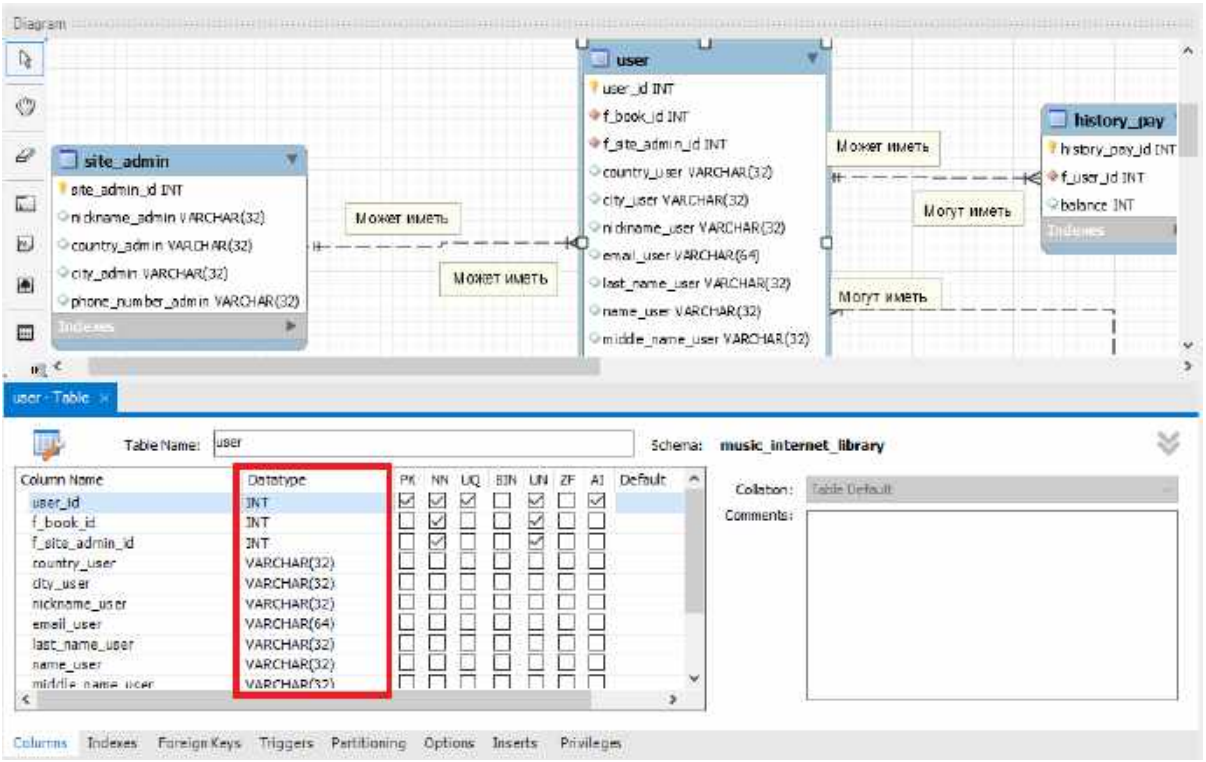

Рисунок 3.15 – Типы данных в *MySQL Workbench*

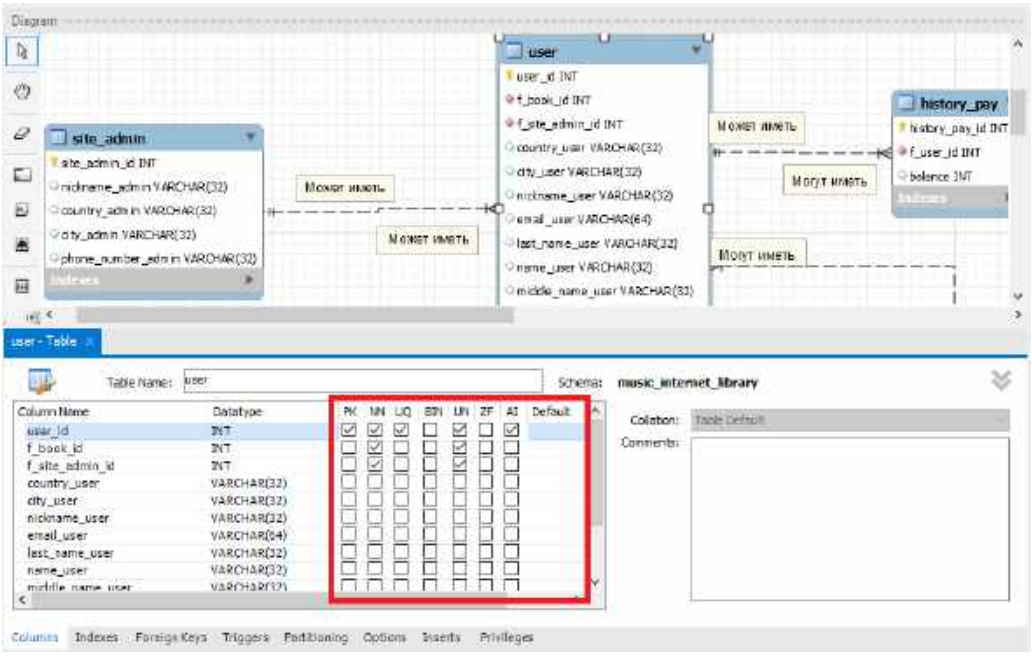

Рисунок 3.16 – Привилегии в *MySQL Workbench*

После переноса сущностей в программную среду мы будем иметь на рабочем пространстве лишь несвязанные между собой таблицы, которые содержат атрибуты. Соблюдение последовательности действий для создания связей между таблицами будет описано в подразделе 3.2.

3.2 Выбор и обоснование используемых типов данных и ограничений (доменов)

В *MySQL* используется множество типов данных, которые можно разделить на шесть групп:

– целые;
– вещественные;
– строковые;
– бинарные;
– даты и времени;
– перечисления и множества.

В таблице 3.1 приведены целочисленные типы данных, представляющие собой целое число: 1, 2, 3,…10,…100 500 и т. д.

Таблица 3.1 – Целочисленные типы данных

Тип данных	Диапазон значений
TINYINT	–128...+127
SMALLINT	–32 768...+32 767
MEDIUMINT	–8 388 608...+8 388 607
INT	–2 147 483 648...+2 147 483 647
BIGINT	–9 223 372 036 854 775 808...+9 223 372 036 854 775 807

В таблице 3.2 представлены вещественные типы данных.

Таблица 3.2 – Вещественные типы данных

Тип данных	Описание
FLOAT	Число с плавающей точкой небольшой точности
DOUBLE	Число с плавающей точкой двойной точности
REAL	Синоним для DOUBLE
DECIMAL	Дробное число, хранящееся в виде строки
NUMERIC	Синоним для DECIMAL

В соответствии с таблицей 3.2 вещественные типы записываются в следующем виде:

ТИП (ДЛИНА, ЗНАКИ) [UNSIGNED]

Длина – это количество знакомест, в которых будет размещено все число при его передаче, а **знаки** – это количество знаков после десятичной точки, которые будут учитываться.

Необязательный флаг UNSIGNED везде означает, что будет создано поле для хранения беззнаковых чисел (больших или равных 0), и удваивает диапазон положительных значений типа данных.

Типы данных NUMERIC и DECIMAL реализованы в *MySQL* как один и тот же тип. Они используются для величин, для которых важно сохранить повышенную точность, например, для денежных данных. Требуемая точность данных и масштаб могут задаваться (и обычно задаются) при объявлении столбца данных одного из этих типов, например:

salary DECIMAL(5,2)

В этом примере цифра 5 (точность) представляет собой общее количество значащих десятичных знаков, с которыми будет храниться данная величина, а цифра 2 (масштаб) задает количество десятичных знаков после запятой. Следовательно, в этом случае интервал величин, которые могут храниться в столбце *salary*, составляет от –99,99 до 99,99 (в действительности для данного столбца *MySQL* обеспечивает возможность хранения чисел вплоть до 999,99, поскольку можно не хранить знак для положительных чисел).

Тип FLOAT обычно используется для представления приблизительных числовых типов данных. Стандарт ANSI/ISO SQL92 допускает факультативное указание точности (но не интервала порядка числа) в битах в круглых скобках, следующих за ключевым словом FLOAT. Реализация *MySQL* также поддерживает факультативное указание точности. При этом если ключевое слово FLOAT в обозначении типа столбца используется без указания точности, *MySQL* выделяет 4 байта для хранения величин в этом столбце. Возможно также иное обозначение, с двумя числами в круглых скобках за ключевым словом FLOAT. В этом варианте первое число по-прежнему определяет требования к хранению величины в байтах, а второе число указывает количество разрядов после десятичной запятой, которые будут храниться и показываться (как для типов DECIMAL и NUMERIC). Если в столбец подобного типа попытаться записать число, содержащее больше десятичных знаков после запятой, чем ука-

зано для данного столбца, то значение величины при ее хранении в *MySQL* округляется для устранения излишних разрядов.

Для типов REAL и DOUBLE не предусмотрены установки точности. Но вопреки требованию стандарта, указывающему, что точность для REAL меньше, чем для DOUBLE, в *MySQL* оба типа реализуются как 8-байтовые числа с плавающей удвоенной точкой. Чтобы обеспечить максимальную совместимость, в коде, требующем хранения приблизительных числовых величин, должны использоваться тип FLOAT или DOUBLE без указания точности или количества десятичных знаков.

В таблице 3.3 представлены основные используемые строковые типы данных.

Таблица 3.3 – Строковые типы данных

Тип данных	Описание
VARCHAR	Может хранить не более 255 символов
CHAR	Может хранить не более 255 символов
TINYTEXT	Может хранить не более 255 символов
TEXT	Может хранить не более 65 535 символов
MEDIUMTEXT	Может хранить не более 16 777 215 символов
LONGTEXT	Может хранить не более 4 294 967 295 символов

В большинстве случаев применяется тип VARCHAR, или просто CHAR, позволяющий хранить строки, содержащие до 255 символов. В скобках после типа указывается длина строки (количество символов, содержащихся в строке).

Величины в столбцах VARCHAR() представляют собой строки переменной длины. Так же как и для столбцов CHAR(), можно задать столбец VARCHAR() любой длины между 1 и 255. Однако в противоположность CHAR() при хранении величин типа VARCHAR() используется только то количество символов, которое необходимо, плюс один байт для записи длины. Хранимые величины пробелами не дополняются, наоборот, концевые пробелы при хранении удаляются.

Если задаваемая в столбце CHAR() или VARCHAR() величина превосходит максимально допустимую длину столбца, то эта величина соответствующим образом усекается.

MySQL может без предупреждения изменить тип столбца CHAR или VARCHAR во время создания таблицы.

Различия между этими двумя типами столбцов в представлении результата хранения величин с разной длиной строки в столбцах проиллюстрированы таблицей 3.4.

Таблица 3.4 – Различия CHAR и VARCHAR

Величина	CHAR(4)	Требуемая память	VARCHAR(4)	Требуемая память
''	' '	4 байта	''	1 байт
'ab'	'ab '	4 байта	'ab'	3 байта
'abcd'	'abcd'	4 байта	'abcd'	5 байтов
'abcdefgh'	'abcd'	4 байта	'abcd'	5 байтов

Иными словами, тип данных VARCHAR() является динамическим символьным типом данных, а это значит, что при недоборе базового значения программы в 45 символов атрибут займет меньшее место в памяти. Однако здесь мы не учитываем тот факт, что в кластере в любом случае этот тип данных займет 45 символов, хоть он и является динамическим. Кроме того, при занятии, допустим, 17 символов этого типа данных все равно будет использоваться значение 32, т. к. число занимаемых в памяти символов выражается формулой 2^n. Поэтому для снижения нагрузки на сервер и улучшения индексации записей в атрибутах следует изменять изначальный параметр этого типа данных на значения, которые подпадают под формулу 2^n и которые не будут превышаться.

Тип данных TEXT также имеет четыре модификации – TINYTEXT, TEXT, MEDIUMTEXT и LONGTEXT, соответствующие четырем типам BLOB, о которых будет упомянуто далее в этом подразделе, и имеющие те же требования к объему памяти и максимальную длину. Единственное различие между типами BLOB и TEXT состоит в том, что сортировка и сравнение данных выполняются с учетом регистра для величин BLOB и без учета регистра для величин TEXT. Другими словами, TEXT – это тип данных, независимый от регистра BLOB.

Если размер задаваемого в столбце BLOB или TEXT значения превосходит максимально допустимую длину столбца, то это значение соответствующим образом усекается.

В таблице 3.5 представлены бинарные типы данных.

Таблица 3.5 – Бинарные типы данных

Тип данных	Описание
TINYBLOB	Может хранить не более 255 символов
BLOB	Может хранить не более 65 535 символов
MEDIUMBLOB	Может хранить не более 16 777 215 символов
LONGBLOB	Может хранить не более 4 294 967 295 символов

В соответствии с таблицей 3.5 тип данных BLOB представляет собой двоичный объект большого размера, который может содержать переменное количество данных. Существуют четыре модификации этого типа – TINYBLOB, BLOB, MEDIUMBLOB и LONGBLOB, отличающиеся только максимальной длиной хранимых величин. Тип данных BLOB в основном используется в тех случаях, когда необходимо применить команду шифрования и дешифрования без порчи (или возвращения NULL в случае других типов данных) каких-либо зашифрованных данных при дешифровании.

В таблице 3.6 приведено описание типа данных даты и времени.

Таблица 3.6 – Тип данных даты и времени

Тип данных	Описание
1	2
DATE	Дата в формате ГГГГ-ММ-ДД

1	2
TIME	Время в формате ЧЧ:ММ:СС (или в формате ЧЧЧ:ММ:СС для больших значений часов)
DATETIME	Дата и время в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС
TIMESTAMP	Дата и время в формате *timestamp*. Однако при получении значения поля оно отображается не в формате *timestamp*, а в виде ГГГГ-ММ-ДД ЧЧ:ММ:СС
YEAR	Год в формате ГГГГ

Для этих типов данных существует так называемая «проблема 2000 года». Она состоит в том, что для типов DATETIME, DATE, TIMESTAMP и YEAR даты с неоднозначным годом интерпретируются в *MySQL* по следующим правилам:

– величина года в интервале 00–69 конвертируется в 2000–2069;

– величина года в интервале 70–99 конвертируется в 1970–1999.

Во избежание этой проблемы следует писать величину года в виде 4-значного, а не 2-значного числа: «2016», а не «16».

По данным таблицы 3.6 видно, что типы DATETIME, DATE и TIMESTAMP являются по сути родственными типами данных.

Тип данных DATETIME используется для величин, содержащих информацию как о дате, так и о времени. Поддерживается диапазон величин от «1000-01-01 00:00:00» до «9999-12-31 23:59:59» («поддерживается» означает, что величины с более ранними временными значениями, возможно, тоже будут работать, но нет гарантии того, что они будут правильно храниться и отображаться).

Тип DATE используется для величин с информацией только о дате, без части, содержащей время. Поддерживается диапазон величин от «1000-01-01» до «9999-12-31». Для примера отобразим тип данных DATE в консольном окне *MySQL* (рисунок 3.17).

```
mysql> select*from card_reader;
+-----------------+-----------------+-----------------+--------------------------+---------------------------+---------------------------+
| id_card_reader  | id_reader_card  | id_librarian_card | data_issue_card_reader | period_issue_card_reader | data_return_card_reader |
+-----------------+-----------------+-----------------+--------------------------+---------------------------+---------------------------+
|               1 |               1 |               2 | 2015-03-13               | 2015-04-13                | 2015-04-14                |
|               2 |               2 |               4 | 2015-02-16               | 2015-03-16                | 2015-03-20                |
|               3 |               3 |               5 | 2015-01-20               | 2015-02-20                | 2015-02-27                |
|               4 |               4 |               1 | 2015-05-24               | 2015-06-24                | 2015-06-24                |
|               5 |               5 |               3 | 2015-07-28               | 2015-08-28                | 2015-08-26                |
+-----------------+-----------------+-----------------+--------------------------+---------------------------+---------------------------+
5 rows in set (0.70 sec)
```

Рисунок 3.17 – Представление типа данных DATE

Тип столбца TIMESTAMP обеспечивает тип представления данных, который можно использовать для автоматической записи текущих даты и времени при выполнении операций INSERT и UPDATE, о которых будет сказано в подразделе 3.3. При наличии нескольких столбцов типа TIMESTAMP только первый из них обновляется автоматически. Величины типа TIMESTAMP могут принимать значения от начала 1970 года до некоторого значения в 2037 году с разрешением в одну секунду. Эти величины выводятся в виде числовых значений.

Формат данных, в котором *MySQL* извлекает и показывает величины TIMESTAMP, зависит от количества показываемых символов. Это проиллю-

стрировано в приведенной ниже таблице 3.7. Полный формат TIMESTAMP составляет 14 десятичных разрядов, но можно создавать столбцы типа TIMESTAMP и с более короткой строкой вывода.

Таблица 3.7 – Формат типа данных TIMESTAMP

Тип столбца	Формат вывода
TIMESTAMP(14)	ГГГГММДДЧЧММСС
TIMESTAMP(12)	ГГММДДЧЧММСС
TIMESTAMP(10)	ГГММДДЧЧММ
TIMESTAMP(8)	ГГГГММДД
TIMESTAMP(6)	ГГММДД
TIMESTAMP(4)	ГГММ
TIMESTAMP(2)	ГГ

Остальные типы данных времени и даты урезаются в соответствии с их форматом (от меньших временных рамок к большим). Для примера покажем в таблице 3.8, как урезается тип данных DATETIME.

Таблица 3.8 – Формат типа данных DATETIME

Тип столбца	Формат вывода
DATETIME(14)	ГГГГ-ММ-ДД ЧЧ:ММ:СС
DATETIME(12)	ГГГГ-ММ-ДД ЧЧ:ММ
DATETIME(10)	ГГГГ-ММ-ДД ЧЧ
DATETIME(8)	ГГГГ-ММ-ДД
DATETIME(6)	ГГГГ-ММ
DATETIME(4)	ГГГГ
DATETIME(2)	ГГ

Величины TIME могут изменяться в пределах от «–838:59:59» до «838:59:59». Причина того, что «часовая» часть величины может быть настолько большой, заключается в том, что тип TIME может использоваться не только для представления времени дня (должно быть меньше 24 часов), но также для представления общего истекшего времени или временного интервала между двумя событиями (может быть значительно больше 24 часов или даже отрицательным).

Тип YEAR – это однобайтный тип данных для представления значений года. Диапазон возможных значений от 1901 до 2155. Недопустимые величины YEAR преобразуются в 0000.

Тип перечисления ENUM – это столбец, который может принимать значение из списка допустимых значений, явно перечисленных в спецификации столбца в момент создания таблицы. Значения перечисления записываются через запятую в одинарных кавычках.

Например:

ENUM('20', 'один','два','три','сибд')

С помощью этого типа данных в графическом интерфейсе можно будет реализовать выпадающий список, содержащий эти четыре значения, находящиеся в кавычках. ENUM может иметь максимум 65 535 элементов.

SET – это строковый тип, который может принимать ноль или более значений, каждое из которых должно быть выбрано из списка допустимых значений, определенных при создании таблицы. Элементы множества SET разделяются запятыми. Как следствие, сами элементы множества не могут содержать запятых.

Например, столбец, определенный как SET ("один", "два") NOT NULL, может принимать такие значения:

– "";

– "один";

– "два";

– "один, два".

Множество SET может иметь максимум 64 различных элемента.

Требования к объему памяти для столбцов каждого типа, поддерживаемого *MySQL*, перечислены по категориям в таблицах 3.9–3.11.

Таблица 3.9 – Требования к памяти для числовых типов

Тип столбца	Требуемая память
TINYINT	1 байт
SMALLINT	2 байта
MEDIUMINT	3 байта
INT	4 байта
INTEGER	4 байта
BIGINT	8 байтов
FLOAT(X)	4, если X <= 24 или 8, если 25 <= X <= 53
FLOAT	4 байта
DOUBLE	8 байтов
REAL	8 байтов
DECIMAL(M,D)	M+2 байта, если D > 0, M+1 байта, если D = 0 (D+2, если M < D)
NUMERIC(M,D)	M+2 байта, если D > 0, M+1 байта, если D = 0 (D+2, если M < D)

Таблица 3.10 – Требования к памяти для типов даты и времени

Тип столбца	Требуемая память
DATE	3 байта
DATETIME	8 байтов
TIMESTAMP	4 байта
TIME	3 байта
YEAR	1 байт

Таблица 3.11 – Требования к памяти для символьных типов

Тип столбца	Требуемая память
1	2
CHAR(M)	M байтов, 1 <= M <= 255

1	2
VARCHAR(M)	L+1 байтов, где L <= M и 1 <= M <= 255
TINYBLOB, TINYTEXT	L+1 байт, где L < 2^8
BLOB, TEXT	L+2 байт, где L < 2^16
MEDIUMBLOB, MEDIUMTEXT	L+3 байт, где L < 2^24
LONGBLOB, LONGTEXT	L+4 байт, где L < 2^32
ENUM('value1','value2',...)	1 или 2 байта, в зависимости от количества перечисляемых величин (максимум 65 535)
SET('value1','value2',...)	1, 2, 3, 4 или 8 байтов, в зависимости от количества элементов множества (максимум 64)

VARCHAR, BLOB и TEXT являются типами данных с переменной длиной строки, для таких типов требования к памяти в общем случае определяются реальным размером величин в столбце (представлен символом L в таблице 3.11), а не максимально возможным для данного типа размером. Например, столбец VARCHAR(10) может содержать строку с максимальной длиной 10 символов.

В случае типов данных BLOB и TEXT требуется 1, 2, 3 или 4 байта для записи длины значения данного столбца в зависимости от максимально возможной длины для данного типа.

Размер объекта ENUM определяется количеством различных перечисляемых величин: 1 байт используется для перечисления до 255 возможных величин. Используя 2 байта, можно перечислить до 65 535 величин.

Размер объекта SET определяется количеством различных элементов множества. Если это количество равно N, то размер объекта вычисляется по формуле (N+7)/8 и полученное число округляется до 1, 2, 3, 4 или 8 байтов.

После рассмотрения типов данных перейдем к привилегиям (ограничениям) и представим их в виде таблицы 3.12.

Таблица 3.12 – Привилегии в *MySQL Workbench*

Привилегия	Описание
1	2
PK – Primary Key	Обозначает первичный ключ данной сущности. Для каждой сущности он один-единственный. Связи на *ER*-диаграмме будут являться неидентифицирующими, т. е. независящими друг от друга. Таким образом, можно будет удалить записи внешнего ключа, содержащие в себе ссылку на первичный ключ другой таблицы, и наоборот. С идентифицирующими связями эта процедура могла бы сильно усложниться, поскольку если запись первичного ключа будет использоваться по внешним ключам в другой таблице, то прежде чем удалить запись первичного ключа, если это будет необходимо, придется сначала удалить все записи на внешних ключах, и только потом идентифицирующая связь даст удалить запись первичного ключа

1	2
NN – Not Null	Означает, что при внесении данных поле не будет пустым. В пример возьмем атрибуты первичного ключа. При внесении данных они по умолчанию должны содержать какие-либо данные
UQ – Unique Index	Содержит в себе уникальное значение в пределах атрибута (столбца), которое не будет повторяться. Например, название книги в различных записях может повторяться, но год издания будет другой. Следовательно, это значение не будет уникальным. В другом случае, если взять университет, то в нем с очень малой вероятностью будут присутствовать две кафедры, названия которых будут повторяться. Следовательно, это значение будет уникальным
BIN – Binary	Содержит поле типа TEXT с учетом регистра символов при поиске значений, хранящихся в нем
UN – Unsigned	Содержит только положительные значения. Применим в основном для первичных ключей, поскольку им нельзя задавать отрицательные значения
ZF – Zero Fill	Заполнение нулями. Добавляет к значению в строке нули слева, если длина значения меньше длины поля. Например, установили длину поля INT(3), туда кладем число 1, соответственно, слева будут дописаны 2 нуля – 001, но это по-прежнему число 1, просто пустые символы были заполнены нулями
AI – Auto Incremental	Применяется только к первичному ключу и означает, что первичный ключ будет автоматически заполняться натуральными значениями: 1, 2, 3 и т. д.

Из описания первичного ключа (*Primary Key*) и некоторой части теории об идентифицирующих и неидентифицирующих связях следует, что в данном курсовом проекте приоритетнее всего использовать именно неидентифицирующие связи, о которых более подробно будет сказано далее в этом подразделе.

После того как атрибутам присвоены нужные типы данных и привилегии, остается связать таблицы по внешним ключам. Для этого переходим на вкладку «*Foreign Keys*», как показано на рисунке 3.18.

В этой вкладке мы указываем имя внешнего ключа, которое может быть любым и совпадать с названием атрибута в сущности. Но во избежание ошибки при компилировании скрипта, связанной с дублированием внешнего ключа (*duplicate foreign key*), лучше всего создавать уникальное имя внешнего ключа, не совпадающее с названиями атрибутов в сущностях.

После указания имени внешнего ключа из выпадающего списка выбираем, к какой таблице будет идти внешний ключ.

Выбрав таблицу, указываем, между какими двумя атрибутами двух сущностей будет происходить связь. Если галочка не ставится, то привилегии в этих двух атрибутах противоречат друг другу или несовместимы.

После осуществления этих действий будет создано отношение один к одному в виде связи двух таблиц. Щелкнув дважды кнопкой мыши на этой связи и перейдя во вкладку «*Foreign Key*», можем выбрать, какое это отношение – идентифицирующее или неидентифицирующее – и каков тип отношений – один к одному, один ко многим или многие ко многим (рисунок 3.19).

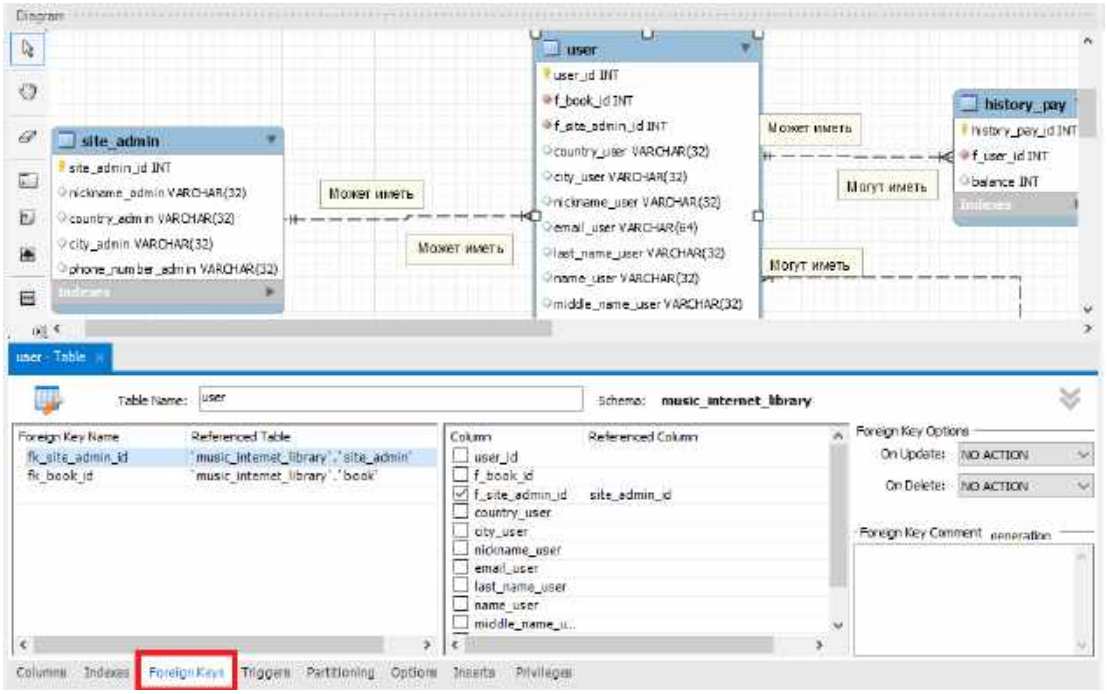

Рисунок 3.18 – Добавление внешнего ключа

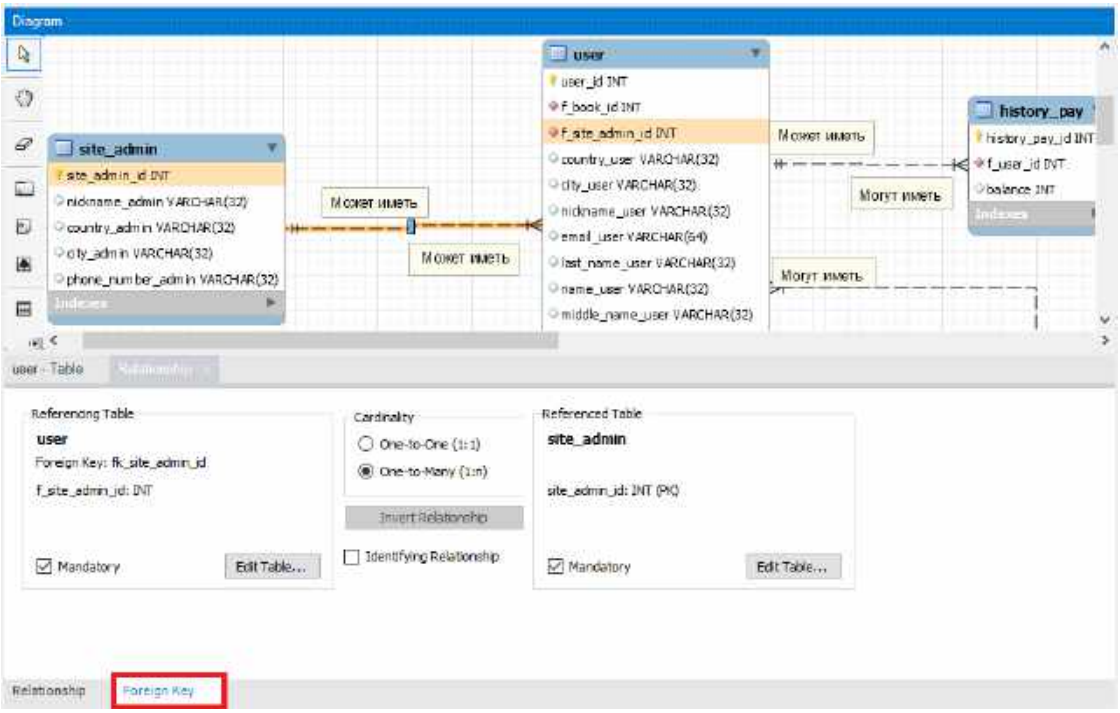

Рисунок 3.19 – Редактирование связи внешнего ключа

Отношения бывают определенных типов (рисунок 3.20). Если мы хотим создать физические таблицы в *MySQL*, то отношения между таблицами должны быть каким-то образом отображены.

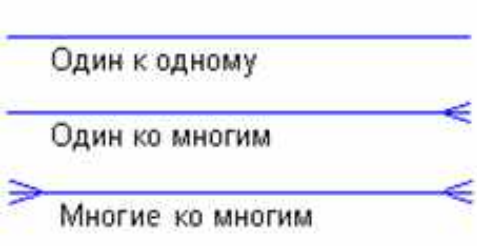

Рисунок 3.20 – Типы связей

Есть несколько правил, которые определяют отношения между таблицами:

– отношение 1:1 (один к одному): первичный ключ для одной из таблиц включен в качестве внешнего ключа в другой таблице;

– отношение 1:n (один ко многим): первичный ключ из таблицы 1 добавляется в качестве внешнего ключа в таблицу n;

– отношение n:m (многие ко многим): создается новая таблица (таблица связи), первичный ключ состоит из первичных ключей двух оригинальных таблиц.

Правильно построенной БД считается та, у которой между сущностями присутствуют только типы связей «один ко многим».

Для отношений 1:1 и 1:n мы имеем два различных типа символов: идентифицирующие и неидентифицирующие.

Отношение считается идентифицирующим, когда одна таблица полностью зависит от других и не может существовать без них. Строка в такой таблице зависит от строки в другой таблице. Типичным примером является создание отдельной таблицы для хранения телефонов пользователей. Их необходимо хранить в другой таблице, потому что у одного пользователя может быть несколько телефонов, но каждая строка в этой таблице полностью зависит от пользователя – значит, она относится к пользователю.

Отношение считается неидентифицирующим, когда одна таблица не зависит от других, и может существовать без них.

Осталось лишь подписать созданные связи. Для этого необходимо определить модальность связи. Каждая связь может иметь одну из двух модальностей связи, представленных на рисунке 3.21.

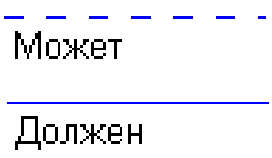

Рисунок 3.21 – Виды модальности связей

Модальность «может» означает, что экземпляр одной сущности может быть связан с одним или несколькими экземплярами другой сущности, а может быть и не связан ни с одним экземпляром.

Модальность «должен» означает, что экземпляр одной сущности обязан быть связан не менее чем с одним экземпляром другой сущности.

Связь может иметь разную модальность с разных концов. Каждая связь может быть прочитана как слева направо, так и справа налево. Типичный пример представлен на рисунке 3.22.

Рисунок 3.22 – Пример модальности связей

Пример представлен для отображения сути правильной подписи связей. Однако сама структура не представлена в 3НФ.

Здесь сразу возникает очевидная связь между следующими сущностями: «покупатели могут покупать много товаров» и «товары могут продаваться многим покупателям».

Помимо всего этого, прямо в программе предусмотрено добавление новых записей в каждый атрибут в предпоследней вкладке «*Inserts*» (см. рисунок 3.18), однако делать это не рекомендуется. При необходимости заполнить определенными данными атрибуты сущности будем использовать командную строку.

На этом проектирование БД на этапе создания диаграммы «сущность – связь» в программе *MySQL Workbench* закончена.

Далее необходимо сгенерировать созданные таблицы и связи в скрипт (код) на языке *SQL* (пример представлен в приложении А) при помощи команды на панели настроек «*Database -> Forward Engineer*», как показано на рисунке 3.23, и на этапе «*Review SQL Script*» сохранить его у себя на компьютере, поскольку в последующем этот код будет использован в качестве приложения для курсового проекта.

В случае ошибки при окончании компиляции, связанной с дублированием внешнего ключа (*duplicate foreign key*), можно использовать указанный ранее вариант: создавать уникальное имя внешнего ключа, не совпадающее с названиями атрибутов в сущностях.

Если в БД были сделаны какие-либо изменения (удален тот или иной атрибут или сущность), то необходимо заново сгенерировать скрипт с помощью

тех же действий. При этом записи, которые были добавлены в атрибуты, не будут удалены. Таким образом будет изменена лишь структура БД.

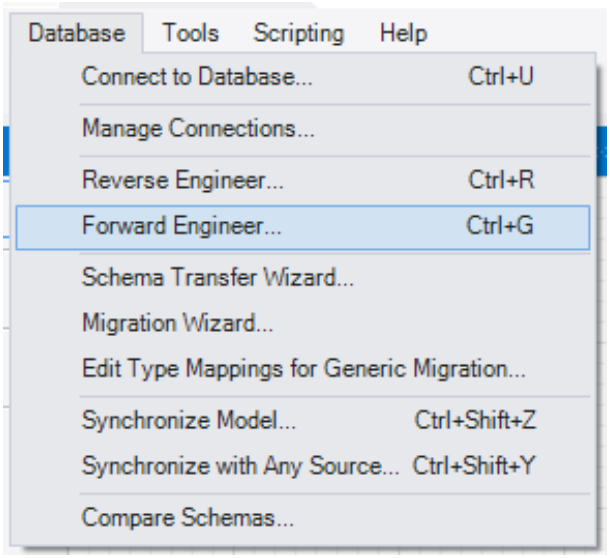

Рисунок 3.23 – Генерирование скрипта

После успешного завершения в командной строке, которая называется «MySQL 5.6 Command Line Client», появилась наша база данных без единой записи. После запуска командной строки необходимо будет ввести пароль (рисунок 3.24), который мы указывали в строке *MySQL Root Password* при установке программы.

Рисунок 3.24 – Первый запуск командной строки

После правильного ввода пароля и успешной авторизации (рисунок 3.25) мы являемся полноценными пользователями командной строки и можем переходить к следующему этапу, связанному с проектированием запросов.

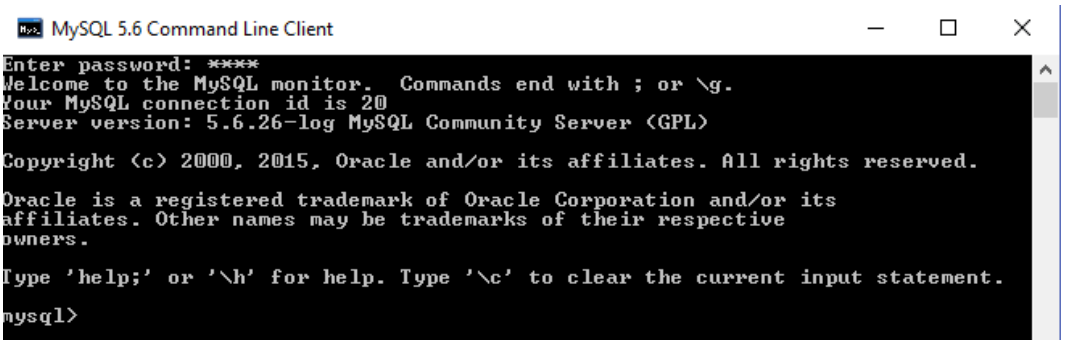

Рисунок 3.25 – Вход в командную строку

Если после ввода пароля командная строка сразу завершает свою работу, то был неверно введен пароль.

3.3 Проектирование запросов к базе данных

Запрос (команда) строится на основе одной или нескольких взаимосвязанных таблиц, позволяя комбинировать содержащуюся в них информацию. При этом могут использоваться как таблицы БД, так и сохраненные таблицы, полученные в результате выполнения других запросов. Кроме того, запрос может строиться непосредственно на другом запросе.

Запрос позволяет выбрать необходимые данные из одной или нескольких взаимосвязанных таблиц и получить результат в виде новой таблицы, содержащей данные из одной или нескольких таблиц. Полученная таблица может использоваться в качестве источника данных в следующих запросах, формах, отчетах, страницах доступа к данным. Через запрос можно производить вычисление, изменение данных в таблицах, добавление и удаление записей и многое другое. Приведем основные простые запросы, но при этом каждому в каждой команде будет прилагаться пример практического использования того или иного запроса.

После входа в командную строку необходимо проверить наличие созданной БД. Для этого выведем список всех БД, существующих на данном сервере *MySQL*, командой

```
SHOW DATABASES;
```

Даже если еще не было создано ни одной базы данных, в полученном списке можно увидеть три основные системные базы данных (рисунок 3.26):

– information_schema – информационная база данных, из которой можно получить сведения о всех остальных базах, структуре данных в них и всевозможных объектах: таблицах, столбцах, первичных и внешних ключах, правах доступа, хранимых процедурах, кодировках и др.; эта база данных доступна только для чтения и является виртуальной, т. е. она не хранится в виде каталога на диске: вся информация, запрашиваемая из этой БД, предоставляется динамически сервером *MySQL*;

– mysql – служебная база данных, которую использует сервер *MySQL*; в ней хранятся сведения о зарегистрированных пользователях и их правах доступа, справочная информация и прочее;

– test – пустая база данных, которую можно использовать для «пробы пера» или просто удалить.

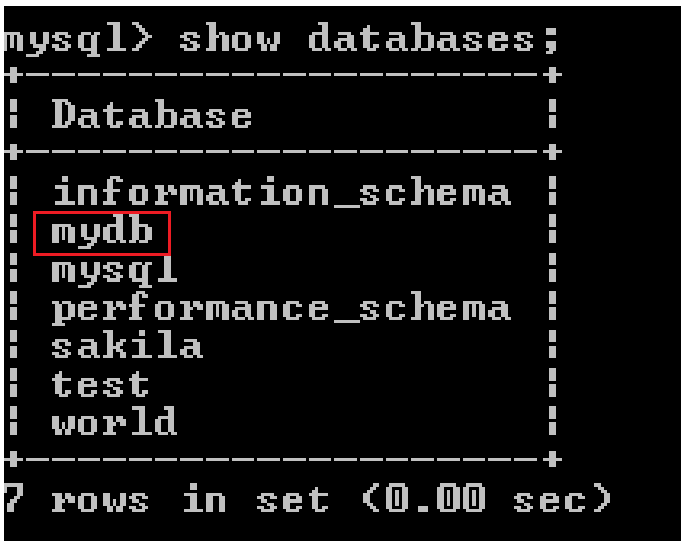

Рисунок 3.26 – Список БД

Если название БД на первоначальном этапе проектирования не менялось, то результатом будет несколько БД (см. рисунок 3.26), одна из которых будет созданная нами БД – «*mydb*». Остальные БД создаются автоматически самой программой при установке.

Следует учесть, что в командной строке при вводе команды (запроса) не учитывается регистр символов. Как видно из рисунка 3.26 и исходной команды, запрос, написанный в верхнем регистре, будет работать и в нижнем регистре. То же касается и данных (записей) в таблицах. Однако для выделения структуры запроса он будет записан в верхнем регистре.

При использовании довольно старой версии программы вместо русского языка, например, при выводе списка БД, увидим то, что представлено на рисунке 3.27.

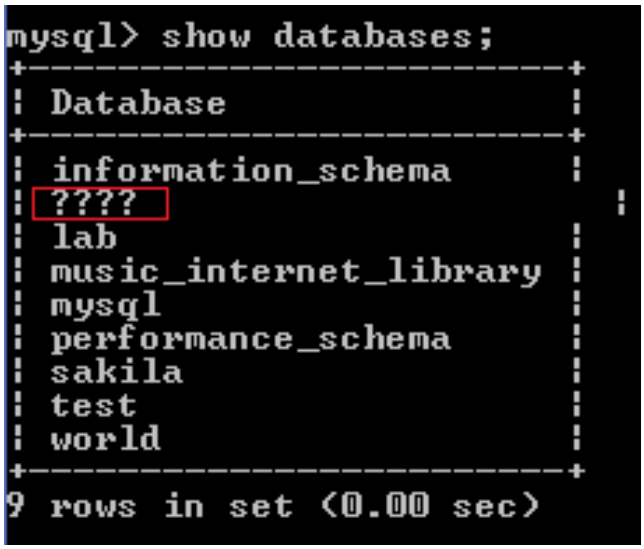

Рисунок 3.27 – Пример неверной кодировки

Чтобы избежать проблем с кодировкой русскоязычных данных, перед началом работы с данными необходимо выполнить команду

```
SET NAMES cp866;
```

Команду SET NAMES необходимо повторять при каждом подключении к серверу с помощью командной строки. Эта команда указывает серверу, что данная командная строка использует кодировку CP-866 (это кодировка командной строки Windows), и сервер будет автоматически выполнять преобразование кодировок при обмене данными с командной строкой.

В более поздних версиях эта кодировка для командной строки установлена по умолчанию. Для просмотра кодировки щелкаем кнопкой мыши на иконке командной строки, которая выделена в левом верхнем углу, и выбираем пункт свойства (рисунок 3.28).

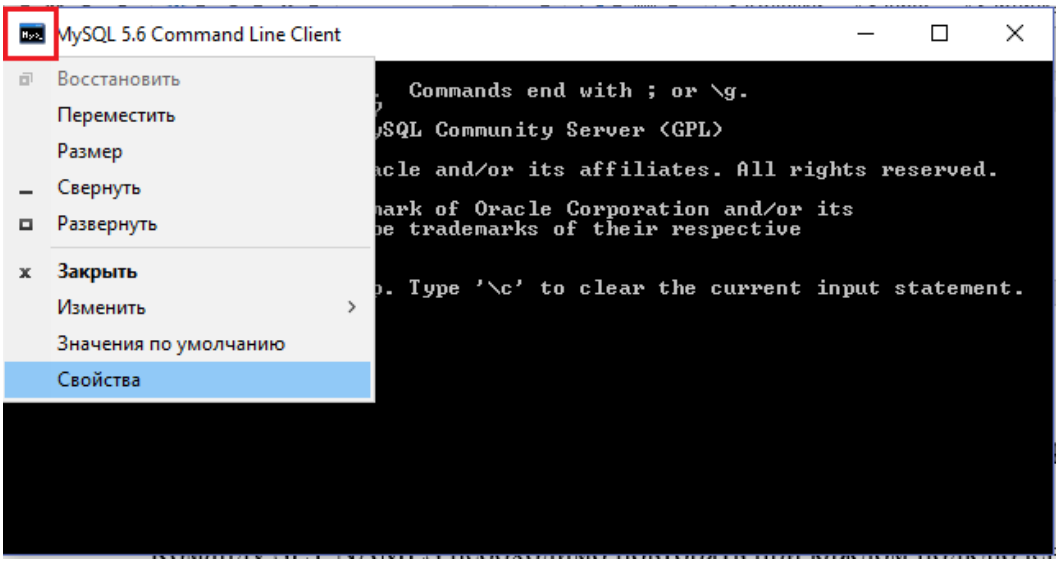

Рисунок 3.28 – Свойства командной строки

Далее открывается окно, в котором можно изменить размер командной строки, посмотреть кодировку и многое другое (рисунки 3. и 3.30).

Изменение размера командной строки используется, когда необходимо вывести данные из довольно большого множества атрибутов из одной или разных таблиц, а в это окно таблица не помещается и происходит нагромождение, представленное на рисунке 3.31. При изменении размера командной строки на больший колонки (атрибуты) встанут на свои места и отображаемые данные будут представлены в более привычном нам виде.

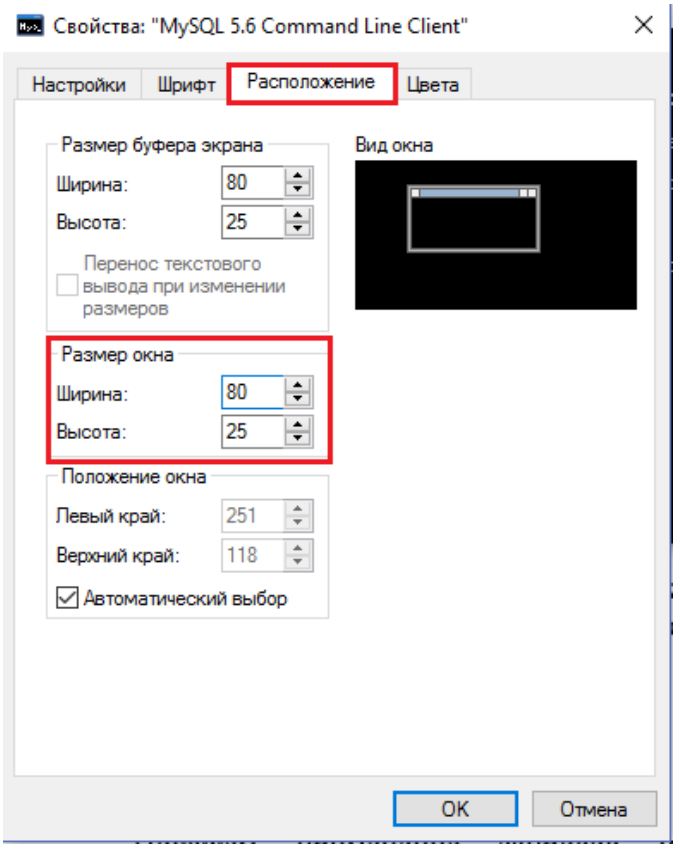

Рисунок 3.29 – Изменение размера командной строки

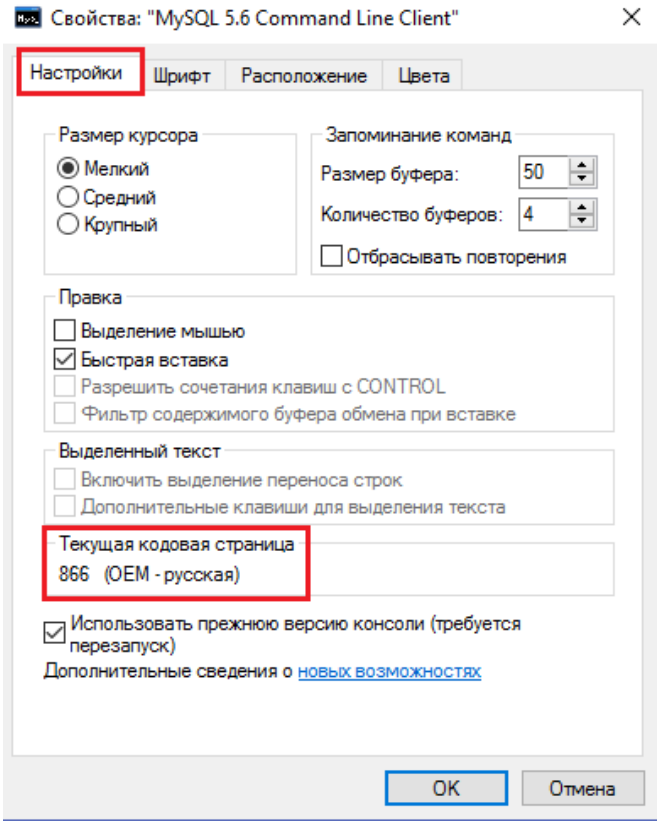

Рисунок 3.30 – Просмотр кодировки командной строки

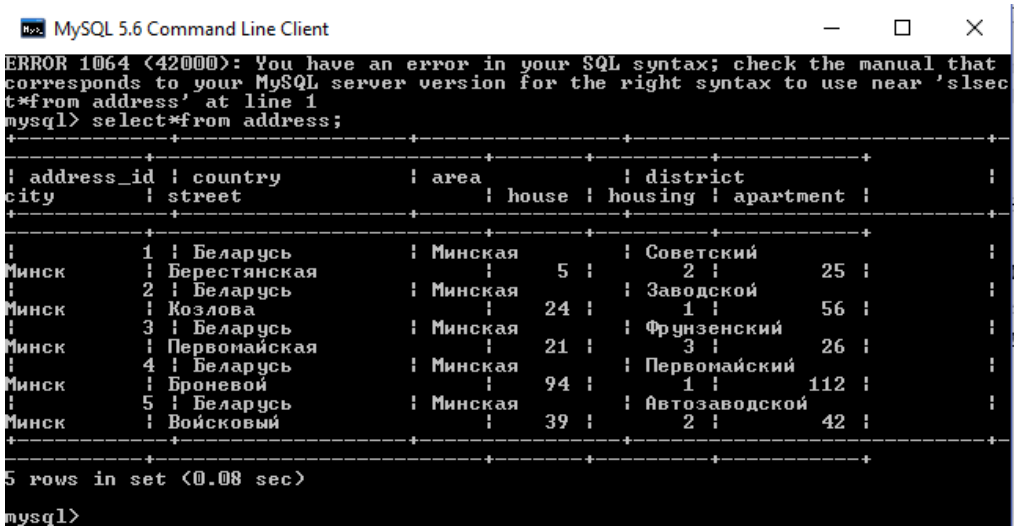

Рисунок 3.31 – Пример использования маленького размера командной строки

Перед всеми манипуляциями в командной строке рекомендуется сделать ее размер в 5–10 раз больше, чем исходное значение.

После просмотра списка БД необходимо зайти в созданную нами БД (рисунок 3.32). Для этого используем команду без точки с запятой или с ней, т. к. сам по себе знак «;» означает окончание запроса. В данном случае это необязательное условие, поскольку это не запрос:

USE <Имя базы данных>

Если название БД не менялось, используем команду

USE mydb

В нашем случае название БД – «*lab*», поэтому команда будет выглядеть следующим образом:

USE lab

Результат команды в командной строке представлен на рисунке 3.32.

Рисунок 3.32 – Вход в БД

Итак, находясь в нашей БД, можно переходить к любым действиям над ней. Однако для начала необходимо проверить, все ли таблицы были перенесены с *ER*-диаграммы в нашу БД. Это можно сделать командой

SHOW TABLES;

Если количество таблиц (сущностей) в командной строке и количество таблиц на *ER*-диаграмме совпадает, то можно непосредственно переходить к изучению и проектированию запросов.

Для удобства пользователей в случае длинного запроса командная строка поддерживает многострочный ввод команд, как показано на рисунке 3.33.

Рисунок 3.33 – Пример многострочного ввода команд

Теперь переходим к работе с таблицами. Для показа результатов запросов будет создана исходная таблица *reader*, изображенная на рисунке 3.34, и над ней будут производиться различные манипуляции.

```
mysql> select*from reader;
+-----------+------------------+-------------+--------------------+---------------------+
| id_reader | last_name_reader | name_reader | middle_name_reader | phone_number_reader |
+-----------+------------------+-------------+--------------------+---------------------+
|         1 | yaroshenko       | alexander   | leonidovich        | 2749264             |
|         2 | loiko            | evgeni      | ivanovich          | 5820587             |
|         3 | jih              | andrei      | ivanovich          | 2948104             |
|         4 | grishechko       | vladislav   | andreevich         | 9275925             |
|         5 | tolchikova       | anna        | sergeevna          | 1875043             |
+-----------+------------------+-------------+--------------------+---------------------+
5 rows in set (0.00 sec)
```

Рисунок 3.34 – Тестовая таблица для применения запросов

Команда, которая выводит все данные, содержащиеся в таблице:

*SELECT * FROM <Имя таблицы>;*

Для выбора одной или нескольких таблиц нужно вместо звездочки указать название тех столбцов, которые необходимо вывести, например:

SELECT last_name_reader, name_reader FROM reader;

На рисунке 3.35 показан результат этого запроса.

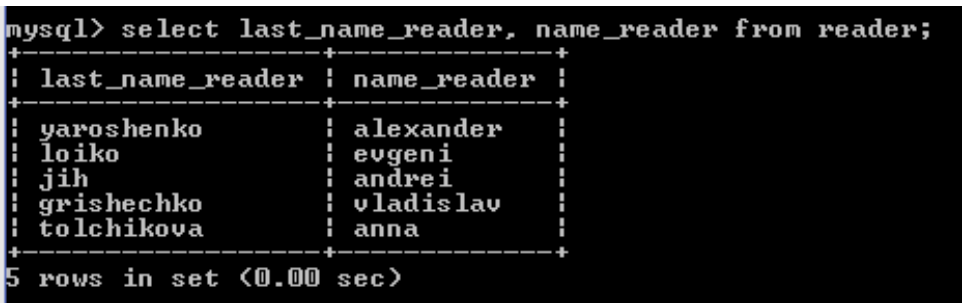

Рисунок 3.35 – Выбор нескольких столбцов из таблицы

Получать с помощью запроса можно не только значения столбцов, но и значения, вычисленные с помощью выражений, например:

SELECT phone_number_reader/10000 FROM reader;

Результат этого запроса представлен на рисунке 3.36.

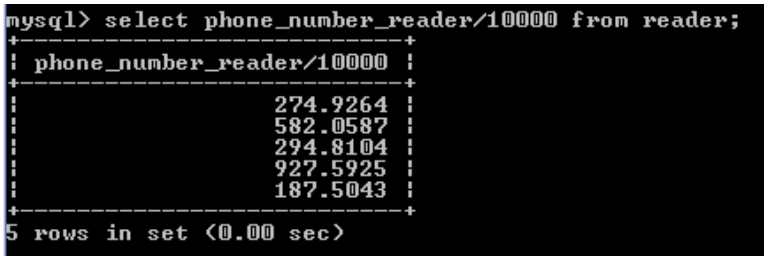

Рисунок 3.36 – Вычисление с помощью запроса

Этот запрос выведет столбец *phone_number_reader* с уменьшенными в 10 000 раз значениями.

Для вывода количества записей, содержащихся в таблице, используем команду

SELECT COUNT(*) FROM reader;

Результат этого запроса представлен на рисунке 3.37.

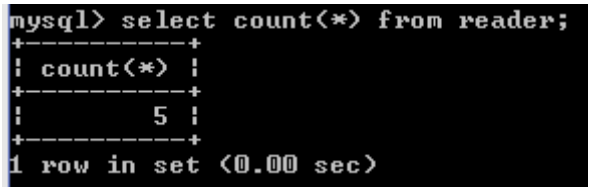

Рисунок 3.37 – Количество записей в таблице

Для вывода минимального или максимального значения в столбце (включая буквенную составляющую и формат даты и времени) используем ключевые слова *MIN* и *MAX* соответственно:

SELECT MIN(<название столбца>) FROM <Название таблицы>;
SELECT MAX(<название столбца>) FROM <Название таблицы>;

Результат применения этих запросов представлен на рисунке 3.38.

```
mysql> select min(id_reader) from reader;
+----------------+
| min(id_reader) |
+----------------+
|              1 |
+----------------+
1 row in set (0.00 sec)

mysql> select max(id_reader) from reader;
+----------------+
| max(id_reader) |
+----------------+
|              5 |
+----------------+
1 row in set (0.00 sec)
```

Рисунок 3.38 – Вывод минимального
и максимального значения в столбце

Для вывода не всех записей, а лишь некоторого их количества, используем ключевое слово *LIMIT*:

```
SELECT * FROM <Имя таблицы>
LIMIT <Номер записи, количество записей>;
```

Номер записи означает, после какой записи в таблице начинать нумерацию. Этот атрибут запроса необязательный. Можно вывести только количество записей, которое означает, сколько записей необходимо вывести, начиная с определенного номера, если таковой параметр присутствует в запросе (по умолчанию с первой записи), например:

*SELECT * FROM reader LIMIT 2;*

Этот запрос выведет две первые записи в таблице, поскольку необязательный параметр *номер записи* отсутствует. Результат этого запроса представлен на рисунке 3.39.

```
mysql> select*from reader limit 2;
+-----------+------------------+-------------+--------------------+---------------------+
| id_reader | last_name_reader | name_reader | middle_name_reader | phone_number_reader |
+-----------+------------------+-------------+--------------------+---------------------+
|         1 | yaroshenko       | alexander   | leonidovich        | 2749264             |
|         2 | loiko            | evgeni      | ivanovich          | 5820587             |
+-----------+------------------+-------------+--------------------+---------------------+
2 rows in set (0.00 sec)
```

Рисунок 3.39 – Вывод первых двух записей

Немного изменив команду и добавив номер записи, команда выведет три записи, которые находятся после первой записи:

*SELECT * FROM reader LIMIT 1,3;*

Результат этого запроса представлен на рисунке 3.40.

```
mysql> select*from reader limit 1,3;
+-----------+------------------+-------------+--------------------+---------------------+
| id_reader | last_name_reader | name_reader | middle_name_reader | phone_number_reader |
+-----------+------------------+-------------+--------------------+---------------------+
|         2 | loiko            | evgeni      | ivanovich          | 5820587             |
|         3 | jih              | andrei      | ivanovich          | 2948104             |
|         4 | grishechko       | vladislav   | andreevich         | 9275925             |
+-----------+------------------+-------------+--------------------+---------------------+
3 rows in set (0.00 sec)
```

Рисунок 3.40 – Вывод трех записей после первой

С помощью запросов можно также вычислять значения без обращения к какой-либо таблице, например:

*SELECT 5*5;*

Этот запрос возвращает результат 25, что иллюстрирует рисунок 3.41.

```
mysql> select 5*5;
+-----+
| 5*5 |
+-----+
|  25 |
+-----+
1 row in set (0.00 sec)
```

Рисунок 3.41 – Вычисление с помощью запросов

Чтобы исключить повторения из результата запроса, добавьте в текст запроса ключевое слово *DISTINCT*:

SELECT DISTINCT <Имя столбца>
FROM <Имя таблицы>;

Например,

SELECT DISTINCT middle_name_reader FROM reader;

Таким образом, мы исключили повторения одинаковых отчеств двух людей во второй и третьей записи, как показано на рисунке 3.42.

```
mysql> select distinct middle_name_reader from reader;
+--------------------+
| middle_name_reader |
+--------------------+
| leonidovich        |
| ivanovich          |
| andreevich         |
| sergeevna          |
+--------------------+
4 rows in set (0.04 sec)
```

Рисунок 3.42 – Исключение повторов

Чтобы упорядочить строки, выведенные запросом, по значениям одного из столбцов, добавьте в текст запроса выражение

ORDER BY <Имя столбца> [ASC или DESC]

Ключевое слово ASC означает, что сортировка выполняется по возрастанию, DESC – по убыванию значений. Сортировка действует как на буквенную (включая русский алфавит), так и на числовую составляющую (включая дату и время). Если ни то, ни другое слово не указано, выполняется сортировка по возрастанию. Кроме того, для сортировки можно использовать сразу несколько столбцов, тогда строки будут отсортированы по значениям первого из столбцов, строки с одинаковым значением в первом столбце будут отсортированы по значениям второго из столбцов и т. д., например:

*SELECT * FROM reader ORDER BY phone_number_reader ASC;*

Этот запрос выведет все столбцы из таблицы, сортируя их по столбцу *phone_number_reader* по возрастанию номеров, как представлено на рисунке 3.43.

```
mysql> select*from reader order by phone_number_reader asc;
+-----------+------------------+-------------+--------------------+---------------------+
| id_reader | last_name_reader | name_reader | middle_name_reader | phone_number_reader |
+-----------+------------------+-------------+--------------------+---------------------+
|         5 | tolchikova       | anna        | sergeevna          | 1875043             |
|         1 | yaroshenko       | alexander   | leonidovich        | 2749264             |
|         3 | jih              | andrei      | ivanovich          | 2948104             |
|         2 | loiko            | evgeni      | ivanovich          | 5820587             |
|         4 | grishechko       | vladislav   | andreevich         | 9275925             |
+-----------+------------------+-------------+--------------------+---------------------+
5 rows in set (0.00 sec)
```

Рисунок 3.43 – Сортировка по возрастанию

В столбце *phone_number_reader* имеются две записи, которые начинаются с одной цифры. Почему результат запроса именно такой? Потому что после сравнения первых чисел, при их совпадении, проверяются следующие числа слева направо.

Чтобы выбрать из таблицы строки, удовлетворяющие какому-либо критерию, добавьте в текст запроса выражение

WHERE <Условие отбора>

Например,

*SELECT * FROM reader WHERE id_reader = “2”;*

Такой запрос выведет строку со всеми столбцами, у которого столбец *id_reader* равен значению 2. Результат представлен на рисунке 3.44.

```
mysql> select*from reader where id_reader = 2;
+-----------+------------------+-------------+--------------------+---------------------+
| id_reader | last_name_reader | name_reader | middle_name_reader | phone_number_reader |
+-----------+------------------+-------------+--------------------+---------------------+
|         2 | loiko            | evgeni      | ivanovich          | 5820587             |
+-----------+------------------+-------------+--------------------+---------------------+
1 row in set (0.05 sec)
```

Рисунок 3.44 – Выбор по условию

В случае, если строка не содержит значения *id_reader*, которое указано в запросе, результатом будет пустое множество, представленное на рисунке 3.45.

```
mysql> select*from reader where id_reader = 10;
Empty set (0.00 sec)
```

Рисунок 3.45 – Отсутствие значения в столбце

Чтобы выбрать строку, которая начинается или заканчивается на определенный набор символов, необходимо добавить ключевое слово *LIKE* и указать в одинарных или двойных кавычках, с чего начинается или на что заканчивается строка, например:

SELECT last_name_reader, name_reader, middle_name_reader
FROM reader
WHERE name_reader LIKE 'an%';

Таким образом, этот запрос выведет имена читателей, которые начинаются с букв «*an*». Результат представлен на рисунке 3.46.

```
mysql> select last_name_reader, name_reader, middle_name_reader
    -> from reader
    -> where name_reader like 'an%';
+------------------+-------------+--------------------+
| last_name_reader | name_reader | middle_name_reader |
+------------------+-------------+--------------------+
| jih              | andrei      | ivanovich          |
| tolchikova       | anna        | sergeevna          |
+------------------+-------------+--------------------+
2 rows in set (0.00 sec)
```

Рисунок 3.46 – Выбор записи по первым двум буквам

Если знак процента будет стоять в начале строки, «%*ko*», то запрос выведет фамилии читателя, которые заканчиваются на «*ko*», например:

SELECT last_name_reader, name_reader, middle_name_reader
FROM reader
WHERE *last_name_reader LIKE '%ko';*

Результат этого запроса представлен на рисунке 3.47.

```
mysql> select last_name_reader, name_reader, middle_name_reader
    -> from reader
    -> where last_name_reader like '%ko';
+------------------+-------------+--------------------+
| last_name_reader | name_reader | middle_name_reader |
+------------------+-------------+--------------------+
| yaroshenko       | alexander   | leonidovich        |
| loiko            | evgeni      | ivanovich          |
| grishechko       | vladislav   | andreevich         |
+------------------+-------------+--------------------+
3 rows in set (0.00 sec)
```

Рисунок 3.47 – Выбор записи по двум последним буквам

Для добавления одной или нескольких записей в таблицу можно использовать команду

INSERT [INTO] <Имя таблицы>

```
[(<Список столбцов>)]
VALUES
(<Список значений 1>),
(<Список значений 2>),
...
(<Список значений N>);
```

В этой команде указываются:

– имя таблицы, в которую добавляются записи;

– список столбцов, в которые необходимо добавить записи;

– список значений, которые представляют собой эту запись.

Во избежание каких-либо ошибок, список значений лучше всего указывать в двойных кавычках, например:

INSERT INTO reader (id_reader, last_name_reader, name_reader, middle_name_reader, phone_number_reader)
VALUES (“6”, “check”, “new”, “record”, “1234567”);

Результат этого запроса и проверка того, добавлена ли запись, представлены на рисунке 3.48.

```
mysql> insert into reader (id_reader, last_name_reader, name_reader,
    -> middle_name_reader, phone_number_reader)
    -> values ("6", "check", "new", "record", "1234567");
Query OK, 1 row affected (0.14 sec)

mysql> select*from reader;
+-----------+------------------+-------------+--------------------+---------------------+
| id_reader | last_name_reader | name_reader | middle_name_reader | phone_number_reader |
+-----------+------------------+-------------+--------------------+---------------------+
|         1 | yaroshenko       | alexander   | leonidovich        | 2749264             |
|         2 | loiko            | evgeni      | ivanovich          | 5820587             |
|         3 | jih              | andrei      | ivanovich          | 2948104             |
|         4 | grishechko       | vladislav   | andreevich         | 9275925             |
|         5 | tolchikova       | anna        | sergeevna          | 1875043             |
|         6 | check            | new         | record             | 1234567             |
+-----------+------------------+-------------+--------------------+---------------------+
6 rows in set (0.00 sec)
```

Рисунок 3.48 – Добавление новой записи

В случае, если в таблице содержатся все пять атрибутов в таком же порядке, то можно сократить запрос, опустив список столбцов:

INSERT INTO reader
VALUES (“6”, “check”, “new”, “record”, “1234567”);

Удалив прежнюю запись (запрос на удаление рассмотрен далее в подразделе) и добавив этот запрос, результат будет точно такой же, как в предыдущем запросе (рисунок 3.49).

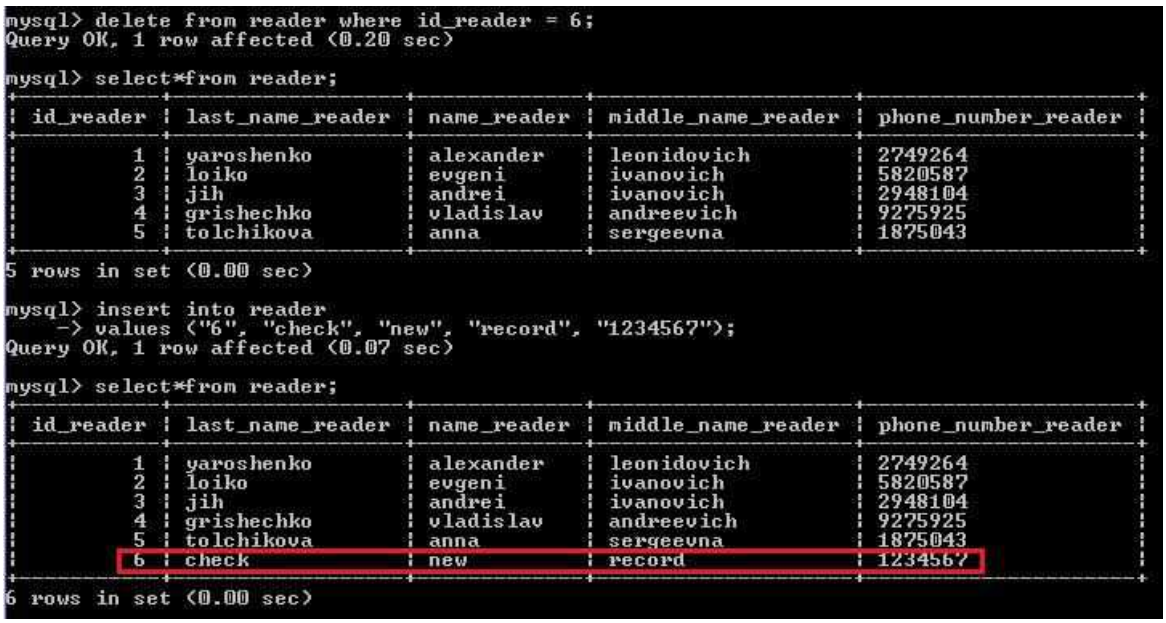

Рисунок 3.49 – Сокращенный запрос на добавление записи

Команда UPDATE позволяет установить новые значения в одной или нескольких строках:

```
UPDATE <Имя таблицы>
SET <Имя столбца 1> = <Значение 1>,
...,
<Имя столбца N> = <Значение N>
[WHERE <Условие отбора>]
[ORDER BY <Имя столбца> [ASC или DESC]]
[LIMIT <Количество записей>];
```

Например, нам необходимо обновить данные о телефонном номере для читателя с *id* = 6. Запрос будет выглядеть следующим образом:

UPDATE reader SET phone_number_reader = “7654321”
WHERE id_reader = 6;

После выполнения запроса видим, что по сравнению с верхней таблицей в шестой записи номер телефона читателя изменился с «1234567» на «7654321», что продемонстрировано на рисунке 3.50.

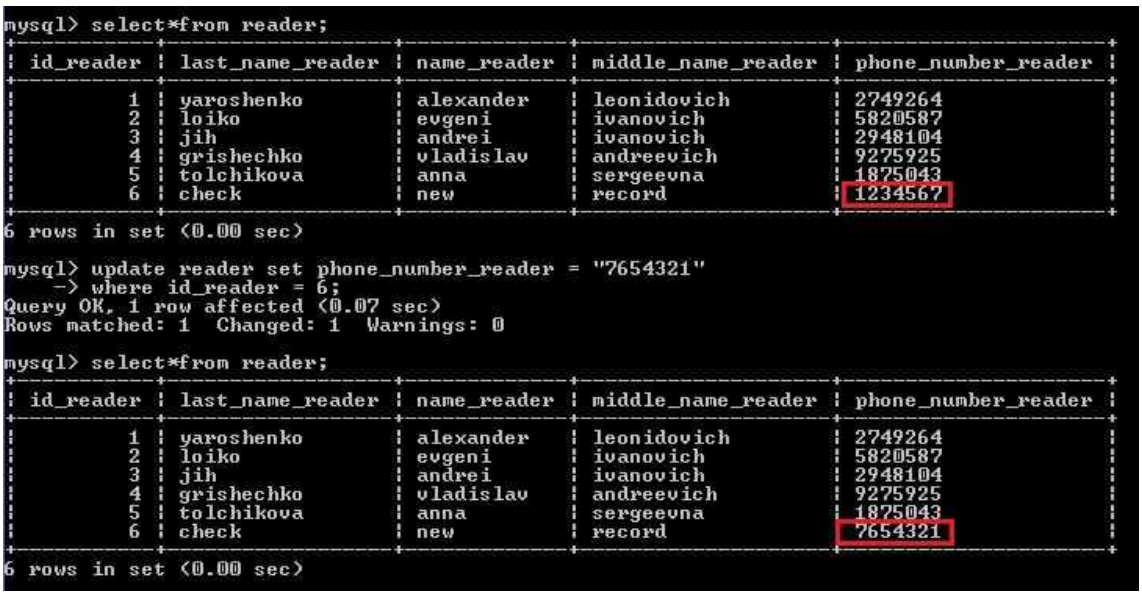

Рисунок 3.50 – Изменение данных в записи

Для удаления записи из таблицы используется команда

DELETE FROM <Имя таблицы>
[WHERE <Условие отбора>]
[ORDER BY <Имя столбца> [ASC или DESC]]
[LIMIT <Количество строк>];

Например, необходимо удалить информацию об уволившемся сотруднике, который имел *id* = 5. Запрос будет выглядеть следующим образом:

DELETE FROM reader WHERE id_reader = 6;

Результат этого запроса представлен на рисунке 3.51.

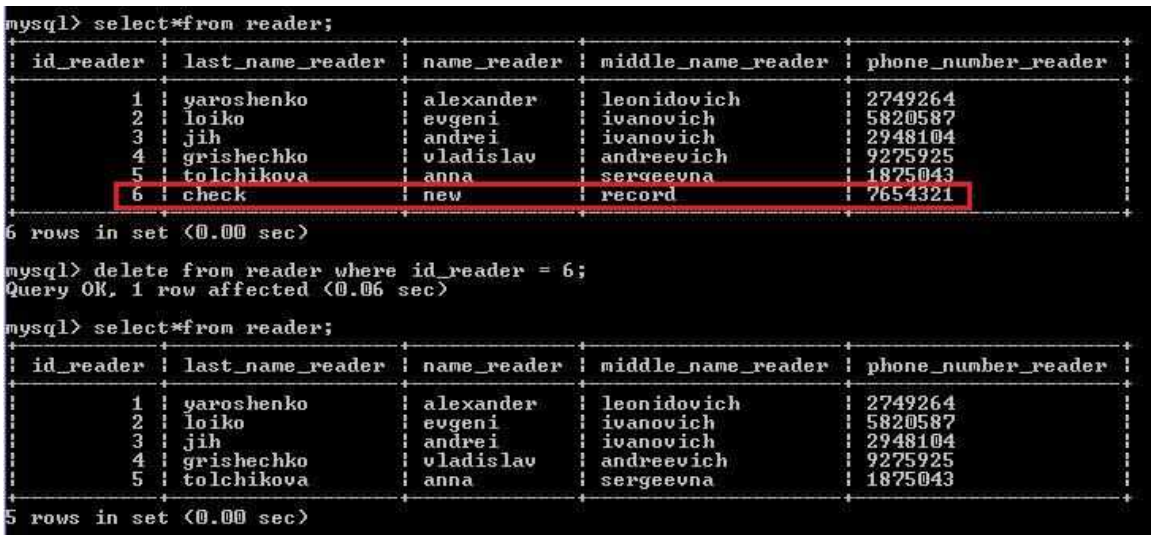

Рисунок 3.51 – Удаление записи из таблицы

После выполнения данного запроса видим, что по сравнению с верхней таблицей была удалена шестая запись.

Параметры команды DELETE аналогичны соответствующим параметрам команды UPDATE.

Для удаления всей таблицы из базы данных используется команда

```
DROP TABLE <Имя таблицы>;
```

Эта команда применяется, например, если была создана лишняя или ненужная таблица, как показано на рисунке 3.52.

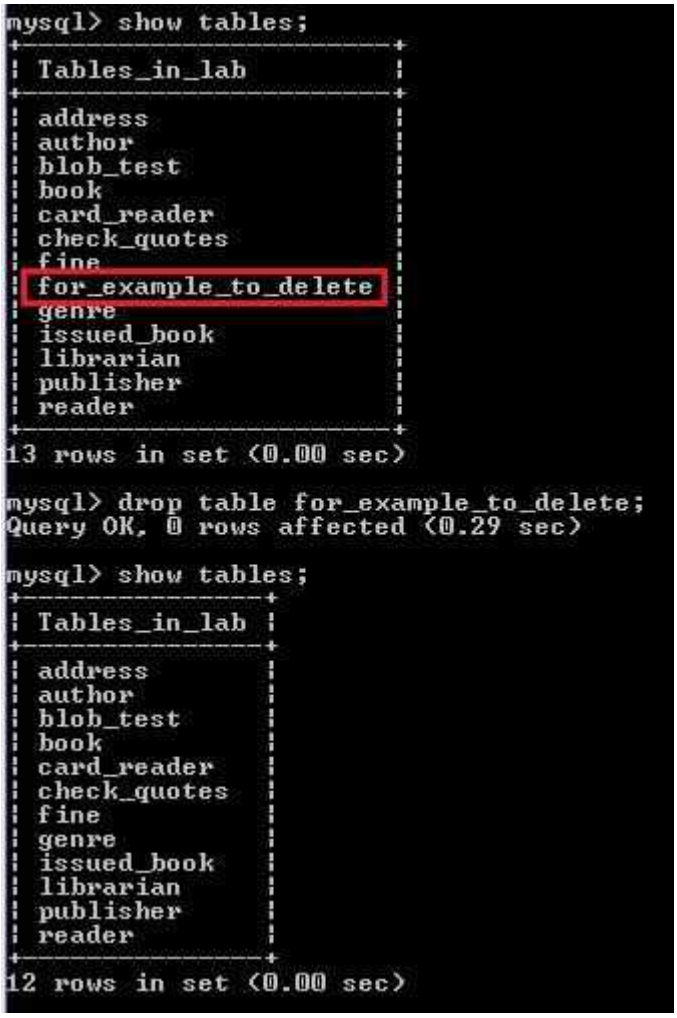

Рисунок 3.52 – Удаление таблицы из БД

После выполнения запроса видно, что таблиц было 13, а стало 12. Одна таблица *for_example_to_delete* действительно была удалена.

Для удаления базы данных используется команда

```
DROP DATABASE <Название БД>;
```

Для просмотра описания таблицы могут использоваться две команды, которые помогут понять, что из себя представляют таблицы, не глядя на скрипт. Каждая из них обладает своими выходными данными:

```
SHOW CREATE TABLE <Название таблицы>;
```

или

```
DESCRIBE <Название таблицы>;
```

Первая команда описывает полную структуру таблицы в соответствии со скриптом, а вторая команда – таблицу в привычном виде, как в программе *MySQL Workbench*. На рисунках 3.53 и 3.54 представлены результаты этих команд.

```
mysql> show create table reader;
+--------+-----------------------------------------------------
| Table  | Create Table
+--------+-----------------------------------------------------
| reader | CREATE TABLE `reader` (
  `id_reader` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `last_name_reader` varchar(32) DEFAULT NULL,
  `name_reader` varchar(32) DEFAULT NULL,
  `middle_name_reader` varchar(32) DEFAULT NULL,
  `phone_number_reader` varchar(32) DEFAULT NULL,
  PRIMARY KEY (`id_reader`),
  UNIQUE KEY `id_reader_UNIQUE` (`id_reader`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8 |
+--------+-----------------------------------------------------
1 row in set (0.00 sec)
```

Рисунок 3.53 – Полная структура таблицы

```
mysql> describe reader;
+---------------------+-------------------+------+-----+---------+----------------+
| Field               | Type              | Null | Key | Default | Extra          |
+---------------------+-------------------+------+-----+---------+----------------+
| id_reader           | int(10) unsigned  | NO   | PRI | NULL    | auto_increment |
| last_name_reader    | varchar(32)       | YES  |     | NULL    |                |
| name_reader         | varchar(32)       | YES  |     | NULL    |                |
| middle_name_reader  | varchar(32)       | YES  |     | NULL    |                |
| phone_number_reader | varchar(32)       | YES  |     | NULL    |                |
+---------------------+-------------------+------+-----+---------+----------------+
5 rows in set (0.03 sec)
```

Рисунок 3.54 – Более привычное представление структуры таблицы

На первом рисунке видно, что описание таблицы выводится в виде кода на языке *SQL*, который показывает, что из себя представляет таблица и каждый атрибут этой таблицы, включая тип данных, привилегии и многое другое. На втором рисунке видим практически тот же вариант представления таблицы, как в *MySQL Workbench*.

Для просмотра текущей выбранной БД используется команда

```
SELECT DATABASE();
```

Результат представлен на рисунке 3.55.

Рисунок 3.55 – Просмотр выбранной БД

Если в данное время нет активной базы данных, то функция DATABASE() возвращает пустую строку. Результат представлен на рисунке 3.56.

Рисунок 3.56 – Просмотр невыбранной БД

Для просмотра текущего активного пользователя БД используется команда

```
SELECT USER();
```

Результат представлен на рисунке 3.57.

Рисунок 3.57 – Просмотр текущего пользователя БД

Для просмотра существующих пользователей используется команда

```
SELECT * FROM mysql.user;
```

Выходная таблица представляет собой множество колонок, поэтому мы покажем лишь ту основную часть, которая наиболее полезна для нас (рисунок 3.58).

```
mysql> select*from mysql.user;
+---------------+-------------+-------------------------------------------+-------------+
| Host          | User        | Password                                  | Select_priv |
+---------------+-------------+-------------------------------------------+-------------+
| localhost     | root        | *F20A3B226A4AEFA22D54DF08B9FABAB6B52E92EB | Y           |
| 127.0.0.1     | root        | *F20A3B226A4AEFA22D54DF08B9FABAB6B52E92EB | Y           |
| ::1           | root        | *F20A3B226A4AEFA22D54DF08B9FABAB6B52E92EB | Y           |
| %             | Ya_roshenko | *F20A3B226A4AEFA22D54DF08B9FABAB6B52E92EB | Y           |
| anna          | moderator   | *8342278FD80E338FC16478FB1C13FA4F04C8A16C | N           |
| leonid        | journalist  | *0E558D9FBD602CDA0C9F3F7A8BC5F4F53401CD7C | N           |
| alex          | reg_user    | *8258F2618980E77E5220ECD738182656223809C1 | N           |
| vlad          | unreg_user  | *432FEE6D1D2A7015E04C3525DED4E21984205512 | N           |
| tatyana       | watcher     | *C64A8AE495AF642E194460AF26820496C1F994AA | N           |
| %             | Zava2012    | *F20A3B226A4AEFA22D54DF08B9FABAB6B52E92EB | Y           |
| galt          | zola        | *98CE51162E10917D664147DF2E8D4FA0C7EC5C8B | N           |
| %             | root        | *F20A3B226A4AEFA22D54DF08B9FABAB6B52E92EB | N           |
| localhost     | unreg_user  | *432FEE6D1D2A7015E04C3525DED4E21984205512 | N           |
| localhost     | reg_user    | *8258F2618980E77E5220ECD738182656223809C1 | N           |
| 192.168.1.105 | journalist  | *0E558D9FBD602CDA0C9F3F7A8BC5F4F53401CD7C | N           |
| 192.168.1.150 | moderator   | *8342278FD80E338FC16478FB1C13FA4F04C8A16C | N           |
| 192.168.1.%   | watcher     | *C64A8AE495AF642E194460AF26820496C1F994AA | N           |
| 192.168.1.0/16| zola        | *98CE51162E10917D664147DF2E8D4FA0C7EC5C8B | N           |
| 192.168.43.131| vasily      | *85012D571AE8732730DE98314CF04A3BB2269508 | N           |
+---------------+-------------+-------------------------------------------+-------------+
19 rows in set (0.00 sec)
```

Рисунок 3.58 – Просмотр списка пользователей

На рисунке 3.58 мы видим, что этот запрос содержит в себе имя хоста, подсеть или конкретный *IP*-адрес класса С, к которому принадлежит пользователь и который может подключиться к БД только с этого хоста или *IP*-адреса.

Второй столбец содержит в себе имена пользователей, третий столбец – пароль в зашифрованном виде. Четвертый и некоторые последующие столбцы содержат в себе колонки, в которых описаны привилегии для пользователей и показано, какие привилегии какому пользователю назначены. Для просмотра всех столбцов введите эту команду у себя в командной строке, поскольку команда содержит в себе множество столбцов, которые интуитивно понятны (*Y* – наличие привилегий, *N* – отсутствие привилегий), но которые сложно охватить двумя-тремя рисунками. Для этого окна следует увеличить размер командной строки, иначе будет более худший результат, чем показанный на рисунке 3.31. Где и как изменить размер командной строки показано на рисунке 3.29.

Касательно пользователей, добавления привилегий и многого другого будет рассказано в подразделе 4.2, а про общую безопасность, шифрование паролей и сохранность данных – в подразделе 4.4.

3.4 Программная реализация и документирование базы данных

В качестве логической модели базы данных предлагается взять реляционную модель. Стандартный язык запросов *SQL* позволяет из таблиц строить производные таблицы, отвечающие на простые и сложные запросы.

Программная реализация предполагает собой первоначально выбор языка программирования, на котором будет происходить написание клиентского приложения. Выбор языка программирования абсолютно и целиком зависит от предпочтений студента. Однако здесь необходимо опираться на сами языки программирования, их плюсы и минусы, простоту интеграции и реализации БД с этим языком программирования.

В настоящее время проще всего интеграция БД *MySQL* происходит с тремя языками программирования *C++*, *Java* и *Perl* и языком сценариев *PHP*.

Также необходимо определить, каким требования будет соответствовать клиентское приложение при его использовании обычным пользователем: безопасность, отказоустойчивость, надежность, функциональность, кроссплатформенность. А также следует определить, с какой целью это приложение нужно будет пользователю.

Далее выбирается *CASE*-средство, поддерживающее этот язык программирования.

Затем выбирается среда разработки клиентского приложения. Среда разработки, как и язык программирования, полностью зависит от предпочтения студента. В данном контексте не существует никаких рамок касательно этого. Кроме того, необходимо не забывать про интересы пользователей данного приложения.

Документирование БД предполагает относительно короткое описание структуры БД. Например, названия таблиц, их полей и других объектов базы данных должны говорить о назначении этого объекта, так же как названия классов и методов в коде. И не стоит забывать, что при изменении сущности или атрибута на структуре, необходимо будет внести эти изменения в документацию.

В сравнение можно привести пример из схемотехнической части. При нумерации компонентов на принципиальной электрической схеме какой-то элемент был сдвинут левее нынешнего *R*1. Допустим, это был компонент *R*2. Следовательно, по ГОСТ 2.702–75 на принципиальной электрической схеме нумерацию левого компонента необходимо будет изменить на *R*1, а правого – на *R*2. Однако на плате компоненты уже были расставлены. Забыв поменять номер компонента на печатной плате мы, возможно, кардинально изменим функционал изделия вплоть до его отказа. То же самое происходит и в базах данных.

Еще одним средством документирования БД являются комментарии к атрибутам прямо в коде, чтобы проектировщик знал, какие функции и задачи выполняет определенный атрибут (рисунок 3.59).

```
`nickname_admin` VARCHAR(32) NULL COMMENT 'имя администратора',
`country_admin` VARCHAR(32) NULL COMMENT 'демонстрация',
`city_admin` VARCHAR(32) NULL COMMENT 'работы',
`phone_number_admin` VARCHAR(32) NULL COMMENT 'комментариев',
```

Рисунок 3.59 – Демонстрация комментариев

Требования к комментариям такие же, как к комментариям в коде: должны раскрывать назначение объекта, если оно не очевидно. Текстовое описание таблицы может содержать максимум 60 символов.

4 ПРИМЕНЕНИЕ РАЗРАБОТАННОЙ БАЗЫ ДАННЫХ

4.1 Руководство пользователя

Руководство пользователя представляет собой справку (документ), с помощью которой обычный пользователь сможет понять, для чего нужна каждая кнопка в этом приложении, что будет, если нажать эту кнопку, какие данные заполнять в этой строке и т. д. То есть этот документ предназначен для пользователей персональных компьютеров или ноутбуков, которые не разбираются в том, что выходит за границы слова Интернет.

Чаще всего такая документация содержит в себе иллюстрации, в которых четко написано и расписано, что делает та или иная кнопка, какие действия при этом происходят и что ждет пользователя при нажатии этой кнопки в последующем.

Итак, руководство пользователя – это некий небольшой документ для осуществления связи пользователя с клиентским приложением, который содержит различные иллюстрации с описанием всевозможных функций как с точки зрения простого пользователя, так и с точки зрения администратора или программиста. Как база данных, так и приложение с документацией делается для пользователя. Таким образом, пользователю данного приложения важнее всего будет решать насущные для него задачи и необходимо, чтобы документ помогал ему в этом.

Практически все программы (исключая *Portable*-версии) имеют установочный дистрибутив. В документе должно быть четко прописано как содержание данного дистрибутива, так и порядок установки и запуска клиентского приложения.

Кроме этого, все документы должны удовлетворять действующим стандартам.

В настоящее время довольно редко можно увидеть приложение, которое не содержало бы ошибок (багов). Если человек не успевает сделать приложение в срок, то этим он ставит себя в положение, когда требуется чем-то жертвовать: функционалом, безопасностью или даже сохранностью данных. В таком случае в документе требуется указать «аварийные ситуации», т. е. то, что на данный момент не стоит нажимать, но по итогу это будет сделано либо исправлено в зависимости от степени завершенности приложения.

И, конечно же, как это обычно происходит, наше клиентское приложение в невероятно редком случае будет уникальным и не иметь аналогов в постсоветском пространстве или вне его. Таким образом, пользователю необходимо предоставить рекомендуемую (дополнительную) литературу или курсы для обучения, с помощью которых он сможет в совершенстве овладеть данным приложением.

4.2 Администрирование базы данных

В обязанности администратора *MySQL* входит также создание и настройка учетных записей пользователей *MySQL*. В процессе этой настройки необходимо определить, какие пользователи будут иметь возможность подключения к серверу, откуда они смогут подключиться и что смогут делать после подключения.

Для создания пользователя, не обладающего никакими привилегиями, можно использовать команду USAGE. Для создания пользователей с какими-либо привилегиями используется команда с оператором GRANT, описанная в данном подразделе.

Чтобы избежать проблем с неверным синтаксисом, оба параметра следует брать в одинарные кавычки:

```
CREATE USER 'user'@'host';
```

Параметр после собачки необязателен, однако он позволяет задать правила входа для пользователя.

Эта команда практически не используется, поскольку присутствует более усовершенствованная команда с оператором GRANT, позволяющая сразу создавать пользователя и присваивать ему привилегии.

Создадим несколько пользователей для различных ситуаций, как показано на рисунке 4.1.

```
mysql> create user 'one'@'find';
Query OK, 0 rows affected (0.00 sec)

mysql> create user 'two'@'192.168.1.15';
Query OK, 0 rows affected (0.00 sec)

mysql> create user 'three'@'192.168.10.%';
Query OK, 0 rows affected (0.00 sec)

mysql> create user 'four'@'192.168.1.15/30';
Query OK, 0 rows affected (0.00 sec)

mysql> create user 'five'@'%';
Query OK, 0 rows affected (0.00 sec)

mysql> create user 'six'@'%.bsuir.by';
Query OK, 0 rows affected (0.00 sec)
```

Рисунок 4.1 – Создание пользователей без привилегий

Первый пользователь сможет заходить в БД только из-под хоста *find*, второй – только с *IP*-адреса 192.168.1.15, третий – с набора *IP*-адресов 192.168.10.0 – 192.168.10.255, т. е. с любого адреса класса С подсети 192.168.10.0, четвертый пользователь сможет заходить только с двух хостов (192.168.1.16 и 192.168.1.17) в подсети 192.168.1.15 (т. к. количество хостов в подсети рассчитывается формулой $2^n - 2$, где n – количество оставшихся битов в маске), пятый – с любого компьютера, шестой – только из-под домена «*bsuir.by*».

Кроме этого, в виде двух записей можно указать, с каких, например, двух доменов пользователю можно будет заходить в БД.

Существует два оператора по работе с пользователями БД, синтаксис которых рассмотрим далее: оператор GRANT, который создает пользователей *MySQL* и позволяет сразу настроить их привилегии, и оператор REVOKE, который удаляет привилегии. Эти два оператора являются своего рода внешним интерфейсом для таблиц разрешений базы данных *MySQL* и обеспечивают альтернативу непосредственному редактированию содержимого этих таблиц.

Операторы GRANT и REVOKE работают с четырьмя таблицами разрешений, представленными в таблице 4.1.

Таблица 4.1 – Таблицы разрешений

Таблица разрешений	Содержимое
user	Подключающиеся к серверу пользователи и все их глобальные привилегии
db	Привилегии уровня базы данных
tables_priv	Привилегии уровня таблицы
columns_priv	Привилегии уровня столбца

Существует еще одна, пятая таблица разрешений (host), однако операторы GRANT и REVOKE не в состоянии ее обрабатывать.

Оператор GRANT имеет следующий синтаксис:

```
GRANT privileges [(columns)]
      ON what
   TO 'user'@'host'
[IDENTIFIED BY "password"]
   [WITH GRANT OPTION];
```

Для успешного его выполнения обязательно нужно правильно определить следующую информацию:

Privileges (привилегии). Привилегии присваиваются определенному пользователю. Если необходимо указать несколько привилегий, то их требуется перечислять через запятую. Используемые в операторе GRANT спецификаторы привилегий описаны в представленной ниже таблице 4.2.

Таблица 4.2 – Список привилегий для пользователя

Спецификатор привилегий	Разрешенная операция
1	2
user	Подключение к серверу пользователей и всех их глобальных привилегий
alter	Изменение таблиц и индексов
create	Создание баз данных и таблиц
delete	Удаление существующих записей из таблиц

1	2
drop	Удаление баз данных и таблиц
index	Создание и удаление индексов
insert	Вставка новых записей в таблицы
references	Не используется
select	Извлечение существующих записей из таблиц
update	Изменение существующих записей таблиц
file	Чтение и запись файлов сервера
process	Просмотр информации о внутренних потоках сервера и их удаление
reload	Перезагрузка таблиц разрешений или обновление журналов, кэша компьютера или кэша таблицы
shutdown	Завершение работы сервера
all	Все операции. Аналог – all privileges
usage	Полное отсутствие привилегий

Спецификаторы привилегий, входящие в первую группу этой таблицы, применяются к базам данных, таблицам и столбцам. Спецификаторы второй группы определяют административные привилегии. Как правило, они применяются довольно редко, поскольку позволяют пользователю влиять на работу сервера (не каждому пользователю, например, необходима привилегия *shutdown*). В третью группу входят два отдельных спецификатора: спецификатор *all* предоставляет «все привилегии», а *usage* означает «полное отсутствие привилегий». В последнем случае создается новый пользователь, не обладающий никакими правами.

Спецификатор *all* и *usage* используется только для базы данных или для таблицы. К столбцам этот спецификатор неприменим.

Columns (столбцы). Столбцы, к которым применяются определенные привилегии. Этот параметр необязателен и используется только при установке привилегий для столбцов. Имена нескольких столбцов отделяются друг от друга запятыми.

Если записать

GRANT SELECT (name_reader) ON lab.reader TO ‘user’@’host’;

то привилегии для созданного пользователя будут распространяться лишь на столбец *name_reader* в таблице *reader* базы данных «*lab»*. Когда необходимо указать столбец, для которого требуется назначить привилегии, всегда нужно не забывать указывать не только БД, к которой мы обращаемся, но и саму таблицу.

What (что). Уровень применения привилегий. Привилегии могут быть глобальными (применяемыми ко всем базам данных и их таблицам), уровня баз данных (применяемыми ко всем таблицам определенной базы данных) или уровня таблицы.

Например, на рисунке 4.2 показано применение привилегий ко всем базам данных. А если вместо части выражения, для которого привилегии будут применяться ко всей БД «*lab»*

GRANT SELECT ON lab. TO 'user'@'host';*

записать

GRANT SELECT ON lab.reader TO 'user'@'host';

то привилегии применятся лишь к конкретной таблице *reader*.

User (пользователь). Пользователь, которому присваиваются привилегии. В некоторых версиях *MySQL* необходимо указывать как имя пользователя, так и компьютер, с которого он сможет подключаться. Такой способ задания легко позволяет определить двух пользователей с одинаковым именем, но подключающихся с разных компьютеров. Возможности *MySQL* позволяют их различать и наделять различными правами.

В операторе GRANT аналогично командам, примеры которых представлены на рисунке 4.1

CREATE USER 'user'@'host';

часть *'user'@'host'* имеет точно такой же фунционал.

Password (пароль). Присвоенный пользователю пароль, который не является обязательным, если пользователь был создан ранее. Если для нового пользователя опустить выражение *IDENTIFIED BY*, то пользователь создан не будет, поскольку пользователя требуется однозначно идентифицировать. Таким образом, при создании нового пользователя необходимо обязательно указывать пароль.

Если же этот оператор задается для уже существующего пользователя, введенный пароль заменит используемый до настоящего момента. Старый пароль останется неизменным, если новый не будет определен в запросе.

Строка пароля, задаваемая с помощью выражения *IDENTIFIED BY*, должна представлять собой буквенную строку, которую при записи зашифрует оператор GRANT. Поэтому не следует применять функцию PASSWORD(), о которой пойдет речь в подразделе 4.4.

Оператор WITH GRANT OPTION является необязательным. С его помощью можно предоставить пользователю все привилегии, существующие в операторе GRANT. Этот оператор можно использовать для делегирования возможностей определенных категорий другим пользователям.

В именах пользователей, баз данных, таблиц и паролях, записываемых в таблицу разрешений, строчные буквы отличаются от заглавных. Регистр в именах компьютеров и столбцов таблиц не учитывается.

Для примера создадим новых пользователей с помощью оператора GRANT и присвоим им какие-либо привилегии и пароль, поскольку, как мы помним, *MySQL* не сможет идентифицировать нового пользователя без задания пароля. Результат представлен на рисунках 4.2–4.4.

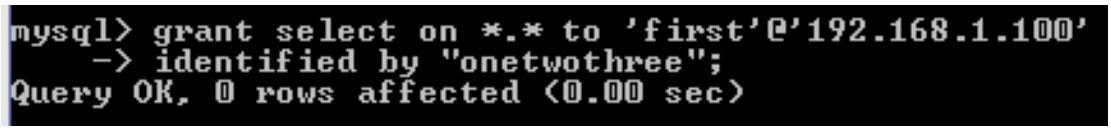

Рисунок 4.2 – Присвоение привилегий и пароля пользователю *first*

Запрос на рисунке 4.2 интересен тем, что вместо привычной базы данных *«lab»* стоит звездочка. Это означает, что пользователю в данном случае предоставлены права на извлечение записей из любой существующей базы данных.

На рисунке 4.3 продемонстрировано предоставление нескольких привилегий для БД *«lab»* и таблицы *reader* для одного пользователя.

```
mysql> grant select, insert, update on lab.reader to 'second'@'192.168.1.105'
    -> identified by "admin";
Query OK, 0 rows affected (0.00 sec)
```

Рисунок 4.3 – Присвоение привилегий и пароля пользователю *second*

На рисунке 4.4 видно, что для уже созданного пользователя вводить пароль необязательно. Однако если пользователя с параметрами *'user'@'host'* не существует, то пользователь создан не будет.

```
mysql> grant select, insert, update on lab.reader to 'second'@'192.168.1.100';
Query OK, 0 rows affected (0.00 sec)

mysql> grant select, insert, update on lab.reader to 'second'@'192.168.1.101';
ERROR 1133 (42000): Can't find any matching row in the user table
```

Рисунок 4.4 – Присвоение привилегий для существующего и несуществующего пользователя в случае отсутствия указания пароля в запросе

Для отмены привилегий пользователя используется оператор REVOKE.

Оператор REVOKE имеет следующий синтаксис:

```
REVOKE privileges [(columns)]
        ON what
    FROM 'user'@'host';
```

Функциональная составляющая параметров оператора REVOKE совпадает с параметрами оператора GRANT.

При этом пользователь, у которого отменили, допустим, все привилегии, не будет удален из базы данных. Ему присвоится спецификатор привилегий USAGE.

Для примера, отменим привилегии пользователя, продемонстрированные на рисунке 4.4, и представим отмену на рисунке 4.5.

```
mysql> revoke select, insert, update on lab.reader from 'second'@'192.168.1.100';
Query OK, 0 rows affected (0.00 sec)
```

Рисунок 4.5 – Отмена привилегий пользователя

Для просмотра привилегий отдельного пользователя необходимо использовать команду

```
SHOW GRANTS [FOR <'user'@'host'>];
```

Команда SHOW GRANTS выводит сведения о привилегиях пользователя в виде набора команд GRANT, с помощью которых можно сформировать текущий набор привилегий пользователя.

Если необязательная часть запроса

[FOR <'user'@'host'>]

не задана, то в командную строку будут выведены привилегии администратора БД (рисунок 4.6).

Рисунок 4.6 – Привилегии администратора

Если же эта часть запроса присутствует, то увидим командную строку, представленную на рисунке 4.7.

Рисунок 4.7 – Привилегии отдельно указанного пользователя

На рисунках 4.6 и 4.7 видно, что пароль для каждого пользователя хранится в зашифрованном виде, хотя изначально мы вводим его в обычном виде. Кроме этого, можно увидеть, как должен выглядеть запрос в командной строке с учетом регистра.

4.3 Реализация клиентских запросов

Снова напомним, что база данных, как и приложение, делается в первую очередь для пользователя, который это приложение будет использовать. В данном подразделе необходимо предположить, какие пользователи будут чаще всего обращаться к одним данным, какие пользователи к другим данным и т. д. Для этого, прежде чем начинать делать приложение, следует разобраться в предпочтениях заказчика (клиента) и по мере получения этой информации выдвигать идеи касательно как функционала, так и интерфейса (дизайна) приложения, что не менее важно.

Например, программист получил заказ на создание клиентского приложения для общеобразовательной школы, в которой уже присутствует собственная база данных, однако без взаимодействия с графической оболочкой. Следовательно, такой БД обычный пользователь управлять не сможет. Для этого программисту необходимо знать, что заказчик хочет видеть в итоге, как он это видит и т. п. И самое главное, что он должен знать, к каким данным какие пользователи будут обращаться и иметь доступ. Проанализировав эти данные, можно

выяснить, что из себя будет представлять клиентское приложение и какие запросы буду присутствовать у обычных пользователей и самого заказчика.

Так, учителю в школе необходимо знать, где и какой у него урок (если он ведет несколько предметов), сколько учеников в классах, с которыми учитель контактирует, каков уровень успеваемости этих учеников и т. д. Однако директору школы этого будет недостаточно. Ему необходимо будет знать, кто ушел из школы или закончил ее в этом году, какие мероприятия планируются, какие документы нужны для того или иного мероприятия и многое другое, что поможет директору в более быстрой автоматизации его работы.

Исходя из этой логики и узнав предпочтения будущих пользователей, можно определить степень сложности реализации приложения.

После анализа потребностей пользователя и заказчика идет собственно разработка данного клиентского приложения по проанализированным предпочтениям.

4.4 Обоснование и реализация механизма обеспечения безопасности и сохранности данных

Обеспечение безопасности в общем плане включает в себя наличие хорошего антивирусного обеспечения, шифрование данных, создание паролей с хорошей степенью защиты, чтобы их трудно было взломать (чередование букв в разных регистрах и наличие цифр) и т. д. Для нашего случая в БД предусмотрены функции шифрования и дешифрования данных.

Если необходимо сохранять результаты функции шифрования, которые могут иметь произвольные байтовые значения, применяются столбцы с типом данных BLOB вместо CHAR или VARCHAR, чтобы избежать потенциальных проблем с удалением завершающих пробелов, которые изменяют значения данных.

```
AES_ENCRYPT(строка, строка_ключа)
AES_DECRYPT(зашифрованная_строка, строка_ключа)
```

Эти функции позволяют выполнять шифрование и дешифрование данных с использованием официального алгоритма *AES (Advanced Encryption Standard)*. Применяется кодирование с 128-разрядным ключом, но можно расширить его до 256 разрядов, должным образом изменив исходные тексты. Длина ключа 128 бит выбрана, поскольку при таком показателе он работает намного быстрее и при этом обеспечивает приемлемый уровень безопасности.

Входные аргументы могут иметь любую длину. Если любой из них равен NULL, результатом функции также будет NULL.

Если функция AES_DECRYPT() обнаруживает неверные данные или неправильное дополнение (например, тип данных, отличный от BLOB), она возвращает NULL. Однако существует вероятность, что AES_DECRYPT() вернет

значение, не равное NULL (возможно, «мусор»), если входные данные или ключ не верны.

Продемонстрируем на примере таблицы одну из множества реализаций использования шифрования и дешифрования с помощью алгоритма *AES* (рисунки 4.8–4.11), который содержит первый ключ с типом данных INT и все остальные атрибуты (столбцы) с типом данных BLOB.

```
mysql> select*from blob_test;
+--------------+---------+---------------+----------+
| id_blob_test | name    | email         | password |
+--------------+---------+---------------+----------+
| 1            | alex    | 123@mail.ru   | 12345    |
| 2            | rafaelo | raf@yandex.ru | rafchik  |
+--------------+---------+---------------+----------+
2 rows in set (0.00 sec)
```

Рисунок 4.8 – Исходная таблица для использования шифрования данных

Применим шифрование к столбцу *name*, содержащему тип данных BLOB.

```
mysql> select name from blob_test;
+---------+
| name    |
+---------+
| alex    |
| rafaelo |
+---------+
2 rows in set (0.00 sec)
```

Рисунок 4.9 – Таблица до шифрования данных

В данном случае синтаксис запроса для шифрования будет следующим:

UPDATE <Название таблицы> SET <Название столбца>
= AES_ENCRYPT (<Название колонки>, <'Секретный ключ'>);

Как видно, в запросе дважды указывается название столбца. В первом случае этот параметр используется для идентификации столбца в таблице, а во втором – для применения функции шифрования.

Секретный ключ указывается в одинарных или двойных кавычках. Он будет использоваться при дешифровании данных в таблице.

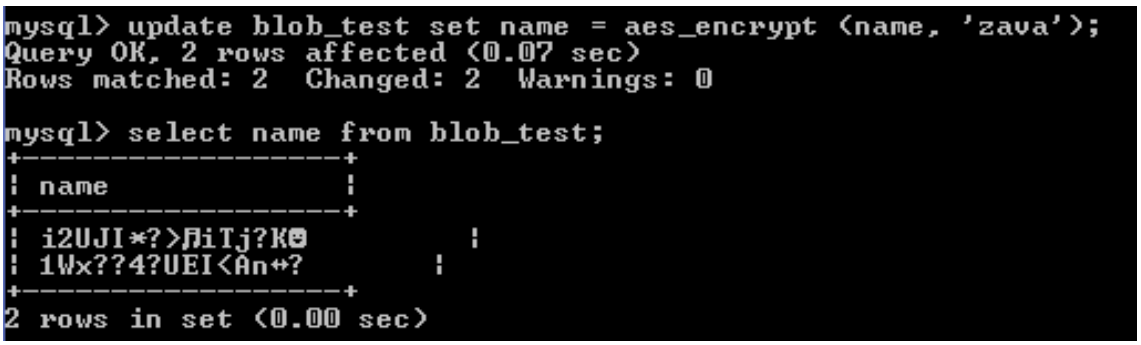

Рисунок 4.10 – Таблица после шифрования данных

Теперь применим операцию дешифрования данных и вернем таблицу в исходное состояние.

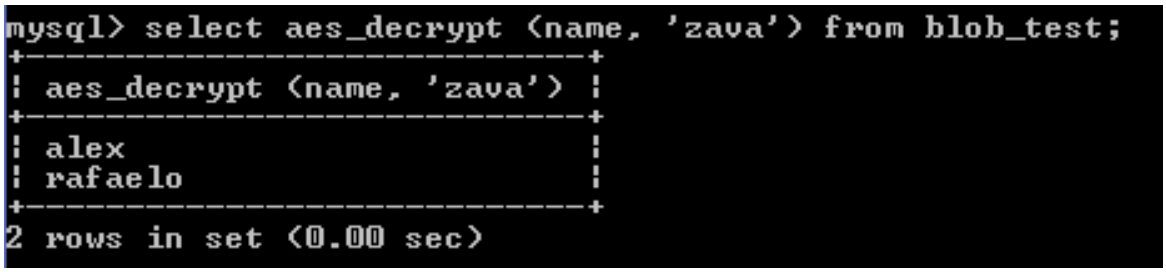

Рисунок 4.11 – Таблица после дешифрования данных

В данном случае синтаксис запроса для шифрования будет следующим:

SELECT AES_ENCRYPT (<Название колонки>, <'Секретный ключ'>)
FROM <Название таблицы>;

Таким образом, была проверена работа алгоритма шифрования *AES* на одном из столбцов в таблице, содержащей необходимый для шифрования тип данных BLOB.

```
ENCODE(строка, строка_пароля)
```

Шифрует строку *строка*, используя значение *строка_пароля* в качестве пароля. Для расшифровки результата применяется функция DECODE(). Результатом является бинарная строка той же длины, что и *строка*. Если нужно сохранить ее в столбце, следует применять тип данных BLOB.

```
DECODE(зашифрованная_строка, строка_пароля)
```

Расшифровывает строку *зашифрованная_строка*, используя значение *строка_пароля* в качестве пароля. Аргумент *зашифрованная_строка* должен быть строкой, ранее возвращенной функцией ENCODE().

```
DES_ENCRYPT(строка[, (номер_ключа | строка_ключа)])
```

Функция ENCODE() шифрует строку с помощью заданного ключа, используя тройной *DES*-алгоритм. В случае ошибки возвращает NULL.

Следует отметить, что эта функция работает, только если *MySQL* настроен на поддержку *SSL*. Ключ шифрования выбирается на базе второго аргумента DES_ENCRYPT(). Если таковой указан, то берется первый ключ из используемого файла *DES*-ключей. Если задан *номер_ключа*, то он берется из используемого файла *DES*-ключей. Если задана *строка_ключа*, то она используется в качестве ключа для шифрования.

Каждый *номер_ключа* должен быть числом в диапазоне от 0 до 9. Строки в файле могут следовать в любом порядке; *строка_ключа* – это строка, которая будет использоваться для шифрования сообщения. Между номером и ключом должен быть по меньшей мере один пробел. Первый ключ является ключом по умолчанию, который применяется в случае, если не указан аргумент *строка_ключа* в функции DES_ENCRYPT().

```
DES_DECRYPT(зашифрованная_строка [, строка_ключа])
```

Расшифровывает строку *зашифрованная_строка*, которая была зашифрована с помощью DES_ENCRYPT(). В случае ошибки возвращает NULL. Следует отметить, что эта функция работает, только если *MySQL* настроен на поддержку *SSL*. Если не указан аргумент *строка_ключа*, DES_DECRYPT() проверяет первый байт зашифрованной строки для определения номера *DES*-ключа, использованного при шифровании исходной строки, а затем читает ключ из файла *DES*-ключей для расшифровки сообщения. Чтобы это работало, пользователь должен иметь права суперпользователя (администратора) в БД.

Если вы передаете этой функции аргумент *строка_ключа*, он используется в качестве ключа при расшифровке сообщения.

Если аргумент *зашифрованная_строка* не выглядит как зашифрованная строка, *MySQL* вернет строку *зашифрованная_строка* без изменений.

```
ENCRYPT(строка [, нач])
```

Шифрует строку *строка*, используя системный вызов *Unix crypt*(). Необязательный аргумент *нач* должен быть строкой не менее чем из двух символов. Если аргумент *нач* отсутствует, то будет использовано случайное значение *Unix crypt*().

ENCRYPT() игнорирует все, кроме первых восьми символов аргумента *строка*. Это поведение определяется реализацией лежащего в основе системного вызова *crypt*().

Если функция *crypt*() недоступна в вашей системе, ENCRYPT() всегда возвращает NULL. По этой причине следует всегда применять вместо этой функции MD5(), поскольку эта функция представлена на всех платформах:

```
MD5(строка)
```

Вычисляет 128-разрядную контрольную сумму MD5 для аргумента *строка*, которую необходимо зашифровать. Значение возвращается в виде 32-разрядной шестнадцатеричной строки (рисунок 4.12) или же NULL, если аргумент равен NULL. Возвращаемое значение может быть использовано, например, в качестве хэш-ключа.

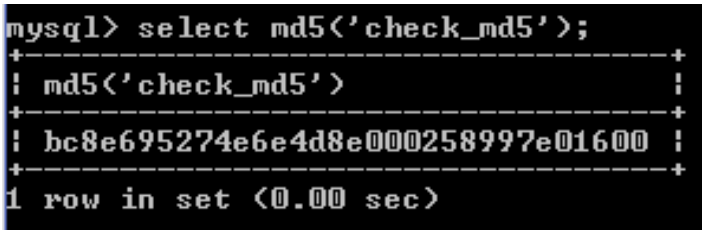

Рисунок 4.12 – Проверка шифрования MD5

В данном случае синтаксис будет следующим:

SELECT MD5(<'строка'>);

И последняя команда, предназначенная для шифрования пароля,

PASSWORD(строка)

Она вычисляет и возвращает строку пароля по значению пароля *строка*, заданному простым текстом (рисунок 4.13). Эта функция применяется для шифрования паролей, сохраняемых в столбце *Password* таблицы привилегий *user*.

Рисунок 4.13 – Шифрование с помощью команды PASSWORD

Синтаксис в этом случае аналогичен шифрованию с помощью MD5, изменена лишь сама команда шифрования:

SELECT PASSWORD(<'строка'>);

Шифрование функцией PASSWORD() является однонаправленным (т. е. необратимым).

Сохранность данных обеспечивается путем резервного копирования данных в конкретный промежуток времени, заданный администратором БД.

Резервная копия – это образ базы данных в конкретный момент времени. К этому образу можно вернуться в случае непредвиденной потери данных. Резервные копии можно создавать сколь угодно часто. Нужно лишь помнить о том, что это достаточно трудоемкий процесс, продолжительность которого зависит от размера базы данных и скоростных характеристик оборудования.

Резервная копия содержит в себе не только сущности и атрибуты, но и записи, которые были внесены в атрибуты.

Создание резервных копий требует от сервера значительных затрат ресурсов, вплоть до того, что работать с другими базами данных станет невозможно. Нужно спланировать этот процесс таким образом, чтобы он приходился на периоды минимальной загруженности сервера.

Действия, связанные с резервным копированием, будут происходить уже в командной строке *Windows*.

Для резервного копирования будем использовать встроенную в *MySQL* утилиту *mysqldump* как один из способов осуществления резервного копирования данных.

На рисунке 4.14 показана исходная таблица, на примере которой мы увидим работу резервного копирования базы данных.

```
mysql> select*from reader;
+-----------+-----------------+-------------+--------------------+----------------------+
| id_reader | last_name_reader | name_reader | middle_name_reader | phone_number_reader |
+-----------+-----------------+-------------+--------------------+----------------------+
|         1 | yaroshenko       | alexander   | leonidovich        | 2749264              |
|         2 | loiko            | evgeni      | ivanovich          | 5820587              |
|         3 | jih              | andrei      | ivanovich          | 2948104              |
|         4 | grishechko       | vladislav   | andreevich         | 9275925              |
|         5 | tolchikova       | anna        | sergeevna          | 1875043              |
+-----------+-----------------+-------------+--------------------+----------------------+
5 rows in set (0.00 sec)
```

Рисунок 4.14 – Исходная таблица перед резервным копированием

Заранее оговорим, что командную строку *Windows* следует запускать от имени администратора. Помимо этого, некоторые действия, осуществленные с помощью операционной системы (ОС), будут происходить на ОС *Windows* 10, поэтому эти действия могут отличаться от других версий ОС.

Перед резервным копированием напишем общую структуру этого запроса в круглых скобках вместо угловых, в отличие от остальных запросов, поскольку здесь уже используются угловые скобки:

```
(Корневой каталог)> mysqldump -u(Имя администратора БД)
        -p(Пароль администратора) (Название БД)
          > (Название файла БД с расширением)
```

Делаем резервное копирование при помощи команды в корневом каталоге (рисунок 4.15):

C:\Program Files\MySQL\MySQL Server 5.6\bin> mysqldump –uroot –p2045 lab > lab.sql

```
c:\Program Files\MySQL\MySQL Server 5.6\bin>mysqldump -uroot -p2045 lab > lab.sq
l
Warning: Using a password on the command line interface can be insecure.
```

Рисунок 4.15 – Осуществление резервного копирования с предупреждением

Однако если пароль к базе данных установлен слишком простой, точнее небезопасный, то в командной строке мы увидим ошибку, продемонстрированную на рисунке 4.15, являющуюся по сути предупреждением, поскольку операция резервного копирования все равно будет проделана.

Чтобы это предупреждение не всплывало, необходимо поле ввода пароля оставить пустым:

C:\Program Files\MySQL\MySQL Server 5.6\bin> mysqldump –uroot –p lab > lab.sql

Вследствие этого после выполнения данной команды будет отдельно запрошен пароль администратора БД, после ввода которого будет произведено резервное копирование базы данных и по указанному в команде пути будет создана резервная копия (дамп) базы данных под названием «*lab.sql*» без строки с пометкой WARNING (рисунок 4.16).

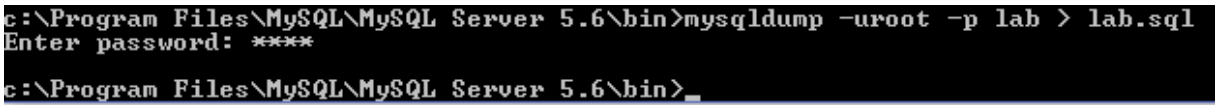

Рисунок 4.16 – Осуществление резервного копирования без ошибки

Видим, что в папке был создан файл «*lab.sql*». На рисунке 4.17 видно, что была создана резервная копия нашей БД (верхний выделенный файл). Кроме этого, в папке присутствует используемая нами утилита *mysqldump* (нижний выделенный файл).

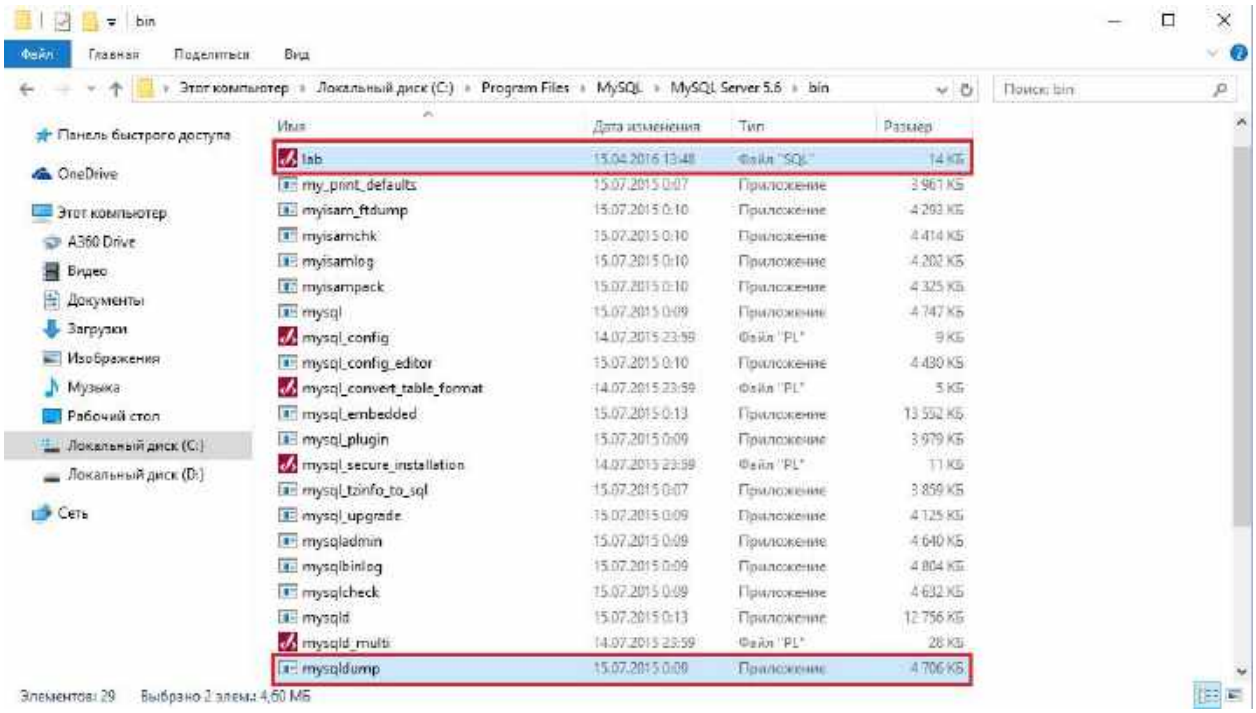

Рисунок 4.17 – Проверка наличия резервной копии

Сразу перейти к этому корневому каталогу в командной строке *Windows*, где бы вы в ней не находились, можно посредством команды, представленной на рисунке 4.18. Регистр символов при этом соблюдать необязательно.

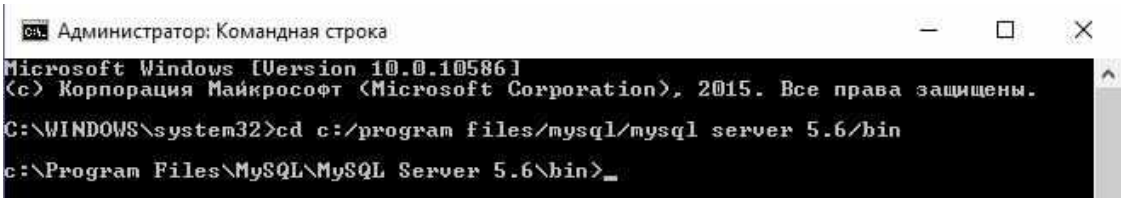

Рисунок 4.18 – Переход к корневому каталогу утилиты

Чтобы не производить таких манипуляций, необходимо внести путь корневого каталога утилиты *mysqldump* в переменные среды *Windows*, если на этапе установки программа этого не было сделано.

Для этого следуем приведенным далее указаниям.

Заходим в «Мой компьютер», щелкаем правой кнопкой мыши и из диалогового окна выбираем пункт «Свойства». Далее щелкаем на выделенной на рисунке 4.19 строке «Дополнительные параметры системы».

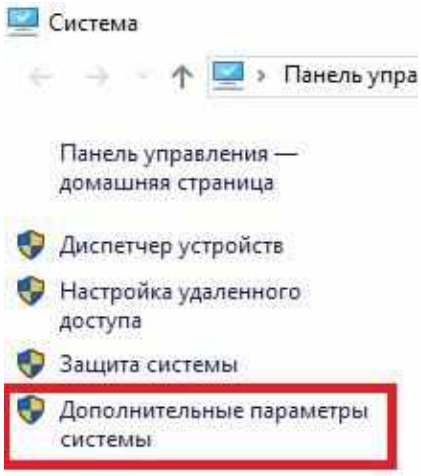

Рисунок 4.19 – Переход в свойства системы *Windows*

В следующем окне, представленном на рисунке 4.20, нажимаем кнопку «Переменные среды…».

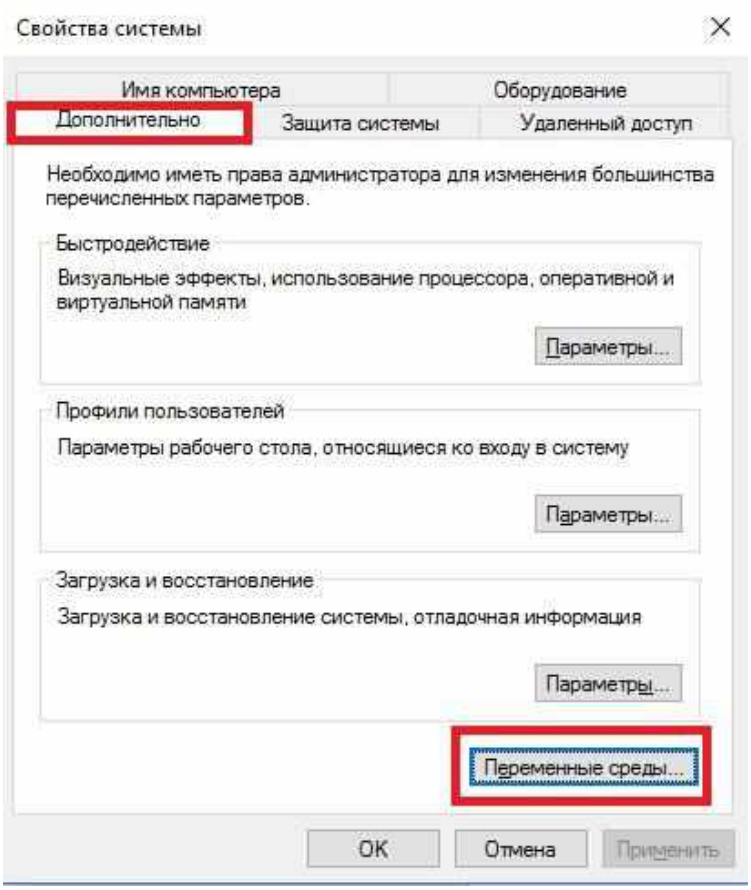

Рисунок 4.20 – Переход в переменные среды *Windows*

В новом окне нам понадобится лишь нижняя часть, в которой мы выделяем строку «*Path*» и нажимаем кнопку «Изменить», как показано на рисунке 4.21.

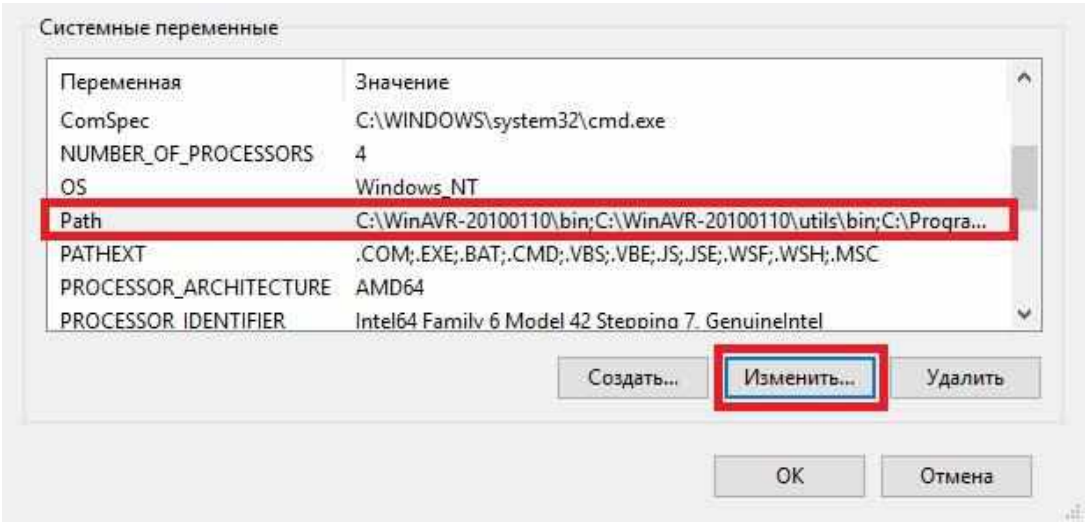

Рисунок 4.21 – Изменение путей переменных среды

В следующем окне сначала нажимаем кнопку «Создать». Появляется строка, в которой необходимо написать любые символы или один символ, что-

бы новая строка сохранилась. Далее нажимаем кнопку «Обзор…» и указываем путь, который показан над созданной строкой на рисунке 4.22 (путь корневого каталога).

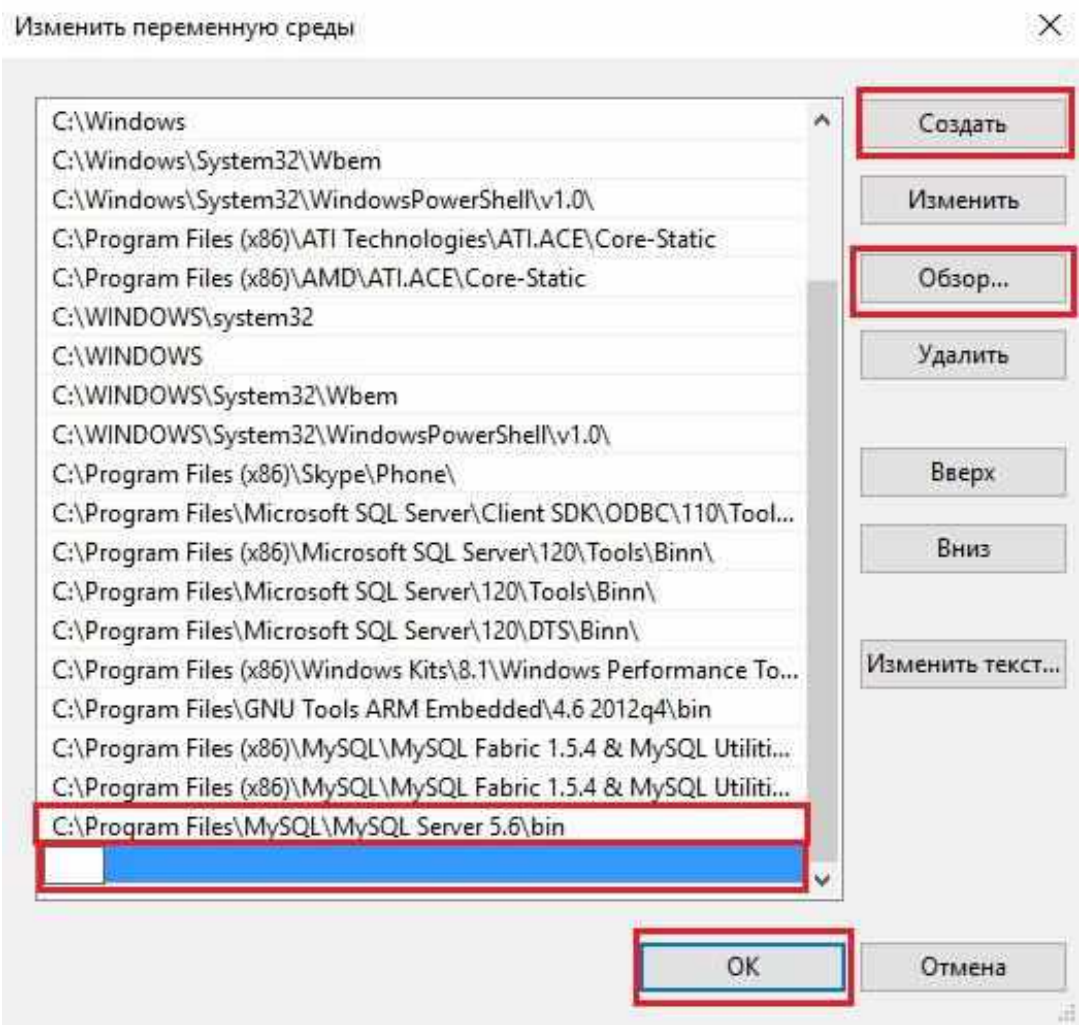

Рисунок 4.22 – Добавлений новой переменной среды

После указания папки все время нажимаем кнопку «Ок». Таким образом, была создана переменная среды *Windows*, позволяющая запускать утилиту *mysqldump* непосредственно после запуска командной строки без указания корневого каталога.

После внесения пути в переменную среды, в случае если в это время командная строка *Windows* была запущена, необходимо будет ее перезапустить.

Написав такую же команду после внесение корневого каталога в переменные среды, увидим, что команда без проблем была принята и выполнена (рисунок 4.23).

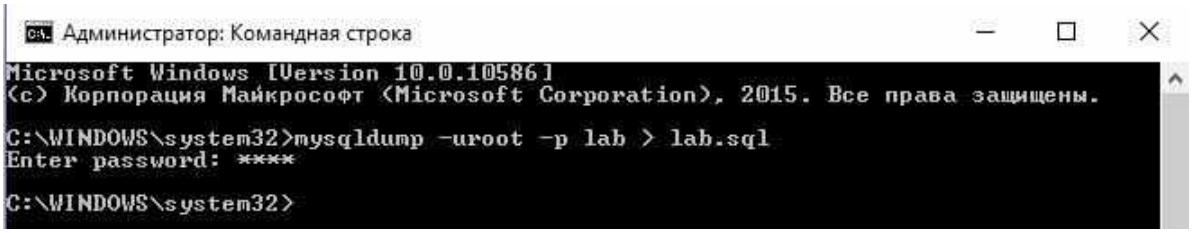

Рисунок 4.23 – Резервное копирование при использовании переменной среды

Без внесения корневого каталога в переменные среды результат выглядел бы таким образом, как показано на рисунке 4.24. При этом резервная копия БД будет создана по тому же корневому каталогу, который представлен на рисунке 4.17.

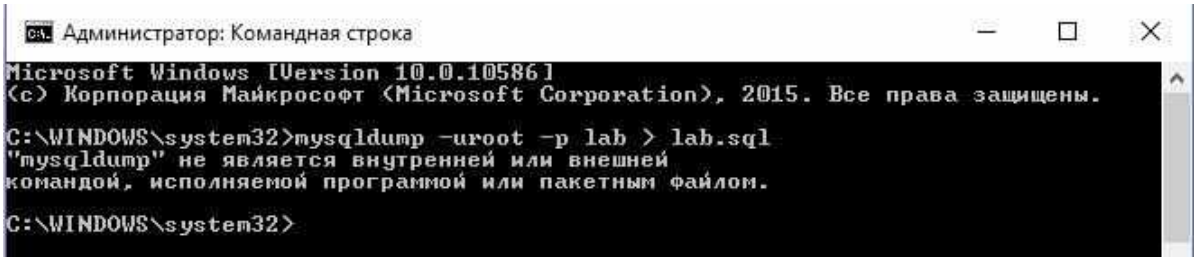

Рисунок 4.24 – Резервное копирование без использования переменной среды

Для проверки того, правильно ли было произведено резервное копирование базы данных, изменим некоторые значения в таблице, как показано на рисунке 4.25, и попробуем ее восстановить до исходного значения.

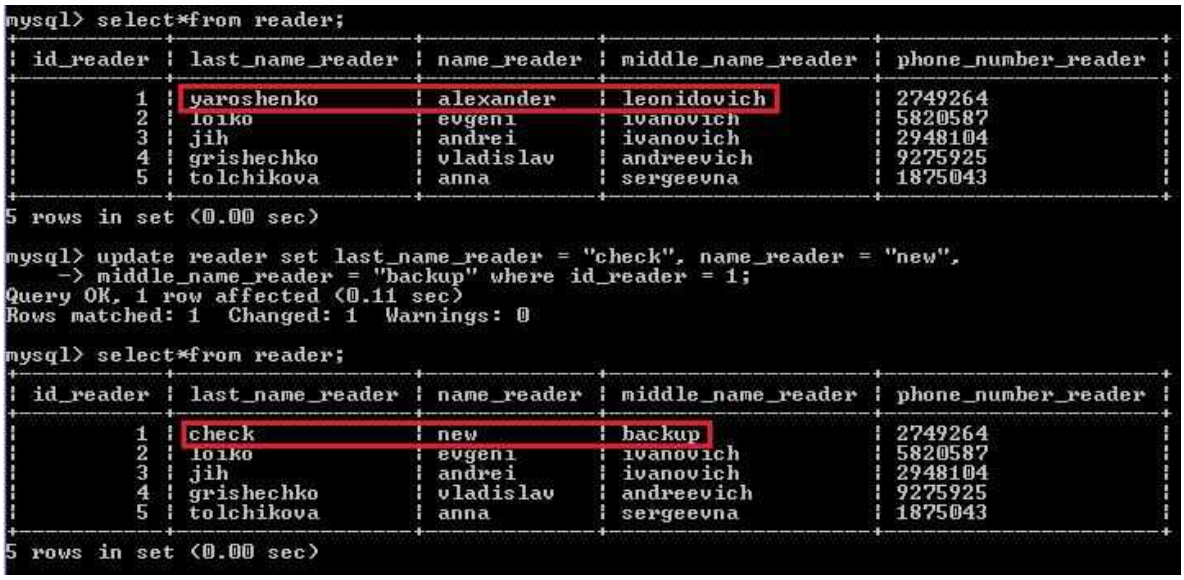

Рисунок 4.25 – Таблица с изменением данных первой строки

Восстанавливаем исходные данные из сделанного заранее дампа с помощью следующей команды, которая дублирована на рисунке 4.23:

C:\WINDOWS\system32> mysql –uroot –p lab < lab.sql

В итоге мы получили исходную таблицу, представленную ранее на рисунке 4.14.

Следовательно, резервное копирование базы данных было сделано корректно, без каких-либо ошибок.

ПРИЛОЖЕНИЕ А
(справочное)
Варианты предметной области

В таблице А.1 приведены варианты предметной области и необходимые исходные данные для разработки.

Таблица А.1 – Варианты предметной области

Вариант предметной области	Исходные данные для разработки
1	2
1 Поликлиника	Поликлиника оказывает платные услуги. Необходимо вести список услуг, список врачей, их исполняющих, списки клиентов. При обращении для клиента формируется листок посещения с датой, врачами и перечнем услуг. После посещения врача перечень услуг может быть изменен. После посещения выполняется окончательный расчет стоимости заказа
2 Налоговая инспекция	Налоговая инспекция ведет учет юридических лиц по видам деятельности, причем юридические лица могут иметь несколько видов деятельности. В налоговую инспекцию поступают отчеты юридических лиц о расчете налогов, а из банка поступает информация об уплате налогов. Необходимо получать информацию о задолженностях и расчете пени, а также иметь возможность просматривать как списки юридических лиц по каждому виду деятельности, так и перечень видов деятельности по каждому юридическому лицу
3 Школа	В школе требуется вести учет успеваемости учащихся (выполнение планов) и хранить их анкетные данные. Первичными документами являются учебные планы. Ученики состоят в классах. Возможно зачисление учеников в классы, указание сведений об окончании школы и переводы между классами. Требуется формировать отчеты об успеваемости и приложения к аттестату
4 Абитуриент	Следует вести списки поступающих на различные факультеты и специальности, формировать группы и графики сдачи вступительных экзаменов, выполнять ввод результатов сдачи экзаменов и, как итог, определять проходной балл и формировать списки поступивших
5 Кафедра	Кафедре учреждения высшего образования требуется вести списки преподавателей с указанием анкетных данных и читаемых ими дисциплин, а также групп студентов, в которых проводят занятия преподаватели кафедры. Каждый преподаватель может читать несколько дисциплин, а одна дисциплина может читаться несколькими преподавателями. Надо иметь возможность просматривать как список преподавателей по каждой дисциплине, так и перечень дисциплин каждого преподавателя, выполнять расчет и вести учет выполненной нагрузки

1	2
6 Деканат	Деканату учреждения высшего образования требуется вести учет успеваемости студентов (выполнение планов) и хранить их анкетные данные. Первичными документами являются учебные планы специальности. Студенты обучаются на нескольких специальностях, на разных кафедрах, состоят в группах. Возможны зачисления студентов в группы, отчисления, восстановления и переводы. Требуется формировать отчеты об успеваемости, академические справки и приложения к диплому
7 Учебный отдел	Учебному отделу учреждения высшего образования требуется вести планирование и учет выполнения почасовой нагрузки преподавателями. Первичными документами являются учебные планы специальности и распределение нагрузки. Выполнение нагрузки и расчеты производятся за любой период, который выбирается преподавателем при оформлении акта. Требуется формировать договора, акты выполнения нагрузки за период, отчеты о выполненной нагрузке, остатках и т. п
8 Отдел кадров	Отдел кадров ведет списки принятых и уволенных работников, а также пенсионеров и работников, состоящих на воинском учете. Надо иметь возможность просматривать списки работников всего предприятия и по подразделениям, осуществлять различные выборки с итоговыми расчетами по количественному и качественному составу работников. Требуется также распечатывать эти списки и выборки в виде отчетов
9 Штатное расписание	Необходимо предусмотреть построение структуры штатного расписания «с нуля», заполнение структуры должностями, окладами и т. п. Размер штатного расписания зависит от структуры предприятия (руководство, управление, отделы, сектора и т. п.). Также необходимо предусмотреть процедуру изменения штатного расписания (введение, выведение из штатного расписания должностей, отделов, управлений и т. п.)
10 Основные средства	Необходимо вести учет движения основных средств (отражать поступления, выбытия, перемещения) внутри счета, осуществлять контроль за наличием и сохранностью счета в местах эксплуатации, предусмотреть начисление амортизационных отчислений. Необходимо иметь в виду, что учет основных средств ведется по однородным группам, которые подразделяются по принадлежности, а также в разрезе отдельных инвентарных номеров объектов
11 Зарплата по сдельно-премиальной схеме	Начисление заработной платы выполняется по сдельно-премиальной схеме. Список выполняемых работ за текущий месяц формируется мастерами участков в соответствии с прейскурантом цен и их объемом. Необходимо формировать расчетную и платежную ведомости, расчет налогов, уплачиваемых от фонда заработной платы по действующему законодательству Республики Беларусь

1	2
12 Зарплата по почасовой премиальной схеме	Начисление заработной платы выполняется по почасовой премиальной схеме. Начисление заработной планы производится исходя из количества отработанного времени, должностного оклада и установленных премиальных (при отсутствии лишения таковых). Необходимо формировать расчетную и платежную ведомости, расчет налогов, уплачиваемых от фонда заработной платы по действующему законодательству Республики Беларусь
13 Депозит	Требуется вести учет депозитов физических лиц по технологии какого-либо банка, включая прием денег, начисления процентов, планирование возврата сумм на следующий день за месяц в целом и по каждому вкладчику отдельно, а также реализовывать формирование договоров, приход-расход за текущий день, месяц, год, пятилетие
14 Кредиты физическим лицам	Требуется вести учет выдачи кредитов физическим лицам, начисление процентов по кредитам, учет данных поручителей, начисление пени в случае задержки выплат по кредиту в установленный срок в размере, установленном договором, также для определенных видов кредитов предусмотреть возможность досрочного погашения без штрафов, а для других – со штрафами. Необходимо формировать договора получения кредитов и поручительства
15 Кредиты юридическим лицам	Требуется вести учет выдачи кредитов юридическим лицам, начисление процентов по кредитам, учет залогов имущества, начисление пени в случае задержки выплат по кредиту в установленный срок в размере, установленном договором, также для определенных видов кредитов предусмотреть возможность досрочного погашения без штрафов, а для других – со штрафами. Необходимо формировать договора закладных при получении кредитов
16 Рекламное агентство	Рекламное агентство собирает заявки от рекламодателей и публикует их в печатных изданиях (газетах, журналах). При этом требуется вести списки печатных изданий с их расценками на рекламу, списки рекламодателей, заявок. Заявка от рекламодателя может запрашивать публикацию рекламы в нескольких печатных изданиях и в различные даты выхода. Необходимо обеспечить оперативный просмотр списка заявок (печатные издания, рекламодатель, стоимость) на любую вводимую дату
17 Агентство по трудоустройству	Агентство по трудоустройству ведет списки лиц, ищущих работу, и списки вакансий, поступающих от организаций, с указанием должности, зарплаты и количества требуемых работников. В заявках претендентов кроме анкетных данных указываются желаемые должности и зарплата. Заявка на вакансию заполняется несколькими претендентами, согласно их анкетным данным, распечатывается и передается работодателю. Работодатель независимо от агентства отбирает одного из претендентов (или исключает всех), который и должен занять вакансию в базе данных агентства, после чего вакансия и претендент «аннулируются» в списке агентства

1	2
18 Автомагазин	Фирма по продаже автомобилей производит их доукомплектование по желанию покупателя. При этом требуется вести учет заказов с перечнем дополнительно устанавливаемых деталей, производить расчет общей суммы и суммы продаж за определенный период времени, а также печать заказа
19 Автосервис	Для станции технического обслуживания нужно формировать заказ на выполнение работ, используя прейскурант цен. Необходимо предусмотреть хранение заказов, их группировку по месяцам и работникам для получения данных об объемах выполненных работ за определенный период, выводить информацию для расчета заработной платы в зависимости от выполненных заказов
20 Букинистический магазин	Требуется вести учет продавцов и покупателей книг, регистрировать предложения продавцов и заказы покупателей в базе данных, информировать продавцов и покупателей о поступлении заказа на интересующие их книги, вести учет проданных книг и расчетов с продавцами
21 Техобслуживание оборудования	Предприятие осуществляет наладку и техническую поддержку оборудования, установленного на других предприятиях. Обслуживание оборудования может проходить как на месте нахождения, так и на предприятии. Необходимо вести учет обслуживаемого оборудования, выполненных работ по его обслуживанию, расходных материалов, затрат на обслуживание. Необходимо формировать текущие планы работ по заявкам и регламенту обслуживания, извещать о необходимости пополнения склада деталями и расходными материалами, оформлять отчеты по обслуживанию
22 Производство персональных компьютеров	Фирма выполняет сборку персональных компьютеров (ПК) из комплектующих деталей и их продажу по индивидуальным заказам и сопровождает гарантийное обслуживание. Необходимо вести учет произведенных ПК за определенный период, случаев гарантийного ремонта, выбраковки деталей и т. п.
23 Грузоперевозки	Следует вести учет заявок на грузоперевозки, учитывать реальные отгрузки, отслеживать оплату услуг и поступления денежных средств. Необходимо формировать отчет по произведенным перевозкам подекадно, отслеживать остаток денежных средств
24 Грузоперевозки промышленного предприятия	Промышленное предприятие осуществляет доставку продукции конечным потребителям через сеть собственных торговых точек собственными транспортными единицами различной грузоподъемности. Выполнение грузоперевозок контролируется по товарно-транспортным накладным. Необходимо вести учет объема перевозок по торговым точкам, водителям и транспортным средствам, контролировать своевременность доставки, рассчитывать суммарный объем и стоимость перевозки
25 Гостиница	Отель расположен в нескольких зданиях, имеет номера различных категорий, набор дополнительных услуг (телевизор, сейф, бар и т. п.). Требуется вести учет загруженности номеров, клиентов и сумм по оплате за заданный период, выполнять бронирование номеров

1	2
26 Сервис-центр	Предприятие оказывает услуги по установке и ремонту бытовой техники. Нужно формировать заказы на выполнение работ, используя прейскурант цен, вести планирование и учет работ, в том числе по гарантии. Необходимо предусмотреть хранение заказов, их группировку по месяцам; выводить список заказов, выполнение которых запланировано на определенный день или за определенный период, список неоплаченных заказов, объем выполненных работ за текущий месяц и т. п.; производить вывод на печать оформленных заказов, а также других необходимых отчетов
27 Учет оргтехники	Требуется вести учет оргтехники, регистрировать заказы подразделений на поставку оборудования и комплектующих, вести учет расходных материалов и замены комплектующих, учет оргтехники, подлежащей списанию, а также учет содержания драгметаллов во всей оргтехнике. Необходимо получать отчеты о расположении оборудования в подразделениях по инвентарным номерам и подразделениям, его состоянии (гарантийный срок, элементы, требующие замены, и др.), а также отчеты по моделям принтеров для заправки и замены картриджей
28 Ремонт оргтех-ники	Требуется вести учет ремонтируемой оргтехники, регистрировать заказы на ремонт, формировать заказы на поставку комплектующих, вести учет расхода материалов и комплектующих. Необходимо получать отчеты об объемах выполненных работ, затратах и доходах, наличии на складе запасных частей, сроках выполнения заказов и т. п.
29 Автопарк	Требуется вести учет сельскохозяйственной техники, регистрировать приход, закрепление за работниками, формировать заказы на поставку новых видов техники и комплектующих, выполнять списание. Необходимо получать отчеты о степени загруженности сельскохозяйственных машин и их техническом состоянии, вести стоимостный учет и рассчитывать амортизацию
30 Склад запчастей	Необходимо вести учет запасных частей для сельхозмашин и агрегатов, поступающих (в соответствии с накладной) в мастерскую и используемых для ремонта сельхозмашин и агрегатов. Учет нужно вести как по ассортименту, так и по стоимости. Необходимо реализовать возможность извещения заведующего мастерской о необходимости пополнения склада деталями
31 Отдел охраны труда	Отдел охраны труда курирует: отдел кадров (ведет списки принятых и уволенных работников, а также списки офицеров и солдат), медицинскую службу (проведение периодического медицинского осмотра гражданского персонала) и вещевую службу (проведение заказа и выдачи средств индивидуальной защиты гражданского персонала). Требуется осуществлять различные выборки с итоговыми расчетами по количественному и качественному составу работников, а также распечатывать эти списки и выборки в виде отчетов

1	2
32 Ремонтное подразделение завода	В подразделении ведется учет времени для выполнения планового ремонта оборудования (в зависимости от его типа), а также обслуживание заявок при поломке оборудования в ходе эксплуатации. Необходимо планировать график ремонта, выполнять расчет затраченного на обслуживание и ремонт времени, формировать итоговую стоимость ремонта в зависимости от его вида и существующих расценок
33 Прокат товаров	Фирма предоставляет на прокат различные товары. Необходимо вести учет прокатных товаров, оформлять оплату, заключать договора, планировать график возвратов, выполнять расчет доходов, полученных от проката товаров, планировать спрос и др.
34 Аренда автомобилей	Фирма предоставляет в аренду автомобили. Необходимо вести учет автомобилей, оформлять оплату, заключать договора, планировать график возвратов арендованных автомобилей, выполнять расчет доходов, полученных от проката товаров, планировать спрос и др.
35 Учет информационных ресурсов	Необходимо вести учет информационных ресурсов, которые курирует предприятие, отслеживать сроки и этапы разработки, соответствие работ поставленным планам, учитывать внеплановые работы, документы-основания выполнения работ. Требуется учитывать сотрудников, входящих в состав рабочей группы по конкретному проекту, а также учитывать расположение ресурса на конкретном сервере, обеспечить возможность получения отчетов по составам рабочих групп, периодам выполнения работ, количеству внеплановых работ и т. д.

ПРИЛОЖЕНИЕ Б
(обязательное)
Пример оформления задания по курсовому проекту

Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»
Факультет компьютерного проектирования
Кафедра проектирования информационно-компьютерных систем

УТВЕРЖДАЮ
Заведующий кафедрой ПИКС
_____ И. Н. Цырельчук
«____» _____ 20__

ЗАДАНИЕ
по курсовому проекту

Группа *313801*

Студенту *Иванову Ивану Ивановичу*

1 Тема проекта: *База данных для поддержки интернет-магазина строительных товаров.*

2 Сроки сдачи студентом законченного проекта: 08.12.2015–09.12.2015.

3 Исходные данные к проекту

3.1 Общие требования: проектируемая база данных должна отвечать требованиям надежности, минимальной избыточности, целостности данных. База данных должна поддерживать основные современные средства для работы и администрирования.

3.2 Требования к программным средствам: _____.

3.3 Требования к языку программирования на стороне клиента: _____.

3.4 Требования, предъявляемые к предметной области и информационным потребностям пользователей: _____.

3.5 Специальные требования: _____.

3.6 Другие требования и нормативные источники:

3.6.1 Положение о курсовом проектировании в БГУИР.

3.6.2 Стандарт предприятия. Дипломные проекты (работы). Общие требования: СТП 01–2013.

4 Содержание расчетно-пояснительной записки

Титульный лист. Реферат. Задание по курсовому проекту. Содержание. Перечень условных обозначений, символов и терминов.

Введение (с указанием цели и основных задач для ее достижения).

4.1 Анализ предметной области и ее формализация для проектирования базы данных.

4.1.1 Описание предметной области.

4.1.2 Анализ информационных потребностей пользователей и предварительное описание запросов.

4.1.3 Определение требований и ограничений к базе данных с точки зрения предметной области.

4.1.4 Постановка решаемой задачи.

4.2 Проектирование базы данных для основного вида деятельности рассматриваемой предметной области.

4.2.1 Разработка инфологической модели предметной области базы данных.

4.2.2 Выбор и обоснование используемых типов данных и ограничений (доменов).

4.2.3 Проектирование запросов к базе данных.

4.2.4 Программная реализация и документирование базы данных.

4.3 Применение разработанной базы данных.

4.3.1 Руководство пользователя.

4.3.2 Администрирование базы данных.

4.3.3 Реализация клиентских запросов.

4.3.4 Обоснование и реализация механизма обеспечения безопасности и сохранности данных.

Заключение. Список использованных источников. Приложения (листинги программного кода).

5 Перечень графического материала

5.1 Структура базы данных (1 лист формата А1).

5.2 Графическая часть, отражающая результаты проектирования (2 листа формата А1).

6 Консультанты по проекту (с указанием разделов): ассистент Богатко Иван Николаевич (ауд. 415а – 1 к.).

7 Дата выдачи задания: 01.09.20__ г.

8 Календарный график работы над проектом на весь период проектирования

Наименование этапов курсового проекта	Срок выполнения этапов проекта	Примечание
1-я опроцентовка (4.1, 5.1)	06.10.20__–07.10.20__	30 %
2-я опроцентовка (4.2)	03.11.20__–04.11.20__	60 %
3-я опроцентовка (введение, 4.3, 5.2, заключение)	01.12.20__–02.12.20__	80 %
Сдача законченного курсового проекта на проверку	08.12.20__–09.12.20__	100 %
Защита курсового проекта	12.12.20__–15.12.20__	Согласно графику

Руководитель _____ (_____)

Задание принял к исполнению 01.09.20__ _____ (_____)

(*подпись студента*) *(расшифровка подписи)*

ПРИЛОЖЕНИЕ В
(справочное)
Пример листинга программного кода

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

-- -----------------------------------------------------
-- Schema music_internet_library
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema music_internet_library
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `music_internet_library` DEFAULT CHARACTER SET utf8
COLLATE utf8_general_ci ;
USE `music_internet_library` ;

-- -----------------------------------------------------
-- Table `music_internet_library`.`site_admin`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `music_internet_library`.`site_admin` (
  `site_admin_id` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `nickname_admin` VARCHAR(32) NULL COMMENT '',
  `country_admin` VARCHAR(32) NULL COMMENT '',
  `city_admin` VARCHAR(32) NULL COMMENT '',
  `phone_number_admin` VARCHAR(32) NULL COMMENT '',
  PRIMARY KEY (`site_admin_id`)  COMMENT '',
  UNIQUE INDEX `site_admin_id_UNIQUE` (`site_admin_id` ASC)  COMMENT '')
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `music_internet_library`.`author`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `music_internet_library`.`author` (
  `author_id` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `last_name_author` VARCHAR(32) NULL COMMENT '',
  `name_author` VARCHAR(32) NULL COMMENT '',
  `middle_name_author` VARCHAR(32) NULL COMMENT '',
  `year_birth_author` DATE NULL COMMENT '',
  `year_death_author` DATE NULL COMMENT '',
  PRIMARY KEY (`author_id`)  COMMENT '',
  UNIQUE INDEX `author_id_UNIQUE` (`author_id` ASC)  COMMENT '')
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `music_internet_library`.`genre`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `music_internet_library`.`genre` (
  `genre_id` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `name_genre` VARCHAR(32) NULL COMMENT '',
  `age_linit` INT UNSIGNED NULL COMMENT '',
  PRIMARY KEY (`genre_id`)  COMMENT '',
  UNIQUE INDEX `genre_id_UNIQUE` (`genre_id` ASC)  COMMENT '')
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `music_internet_library`.`cost`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `music_internet_library`.`cost` (
  `cost_id` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `cost_size` INT UNSIGNED NOT NULL COMMENT '',
  `discount` VARCHAR(8) NULL COMMENT '',
  PRIMARY KEY (`cost_id`)  COMMENT '',
  UNIQUE INDEX `cost_id_UNIQUE` (`cost_id` ASC)  COMMENT '')
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `music_internet_library`.`address`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `music_internet_library`.`address` (
  `address_id` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `country` VARCHAR(32) NULL COMMENT '',
  `area` VARCHAR(32) NULL COMMENT '',
  `district` VARCHAR(32) NULL COMMENT '',
  `city` VARCHAR(32) NULL COMMENT '',
  `street` VARCHAR(32) NULL COMMENT '',
  `house` INT UNSIGNED NOT NULL COMMENT '',
  `housing` INT UNSIGNED NOT NULL COMMENT '',
  `apartment` INT UNSIGNED NOT NULL COMMENT '',
  PRIMARY KEY (`address_id`)  COMMENT '',
  UNIQUE INDEX `address_id_UNIQUE` (`address_id` ASC)  COMMENT '')
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `music_internet_library`.`publisher`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `music_internet_library`.`publisher` (
  `publisher_id` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `address_id` INT UNSIGNED NOT NULL COMMENT '',
  `name_publisher` VARCHAR(32) NULL COMMENT '',
  `phone_number_publisher` VARCHAR(32) NULL COMMENT '',
  PRIMARY KEY (`publisher_id`)  COMMENT '',
  INDEX `address_id_idx` (`address_id` ASC)  COMMENT '',
  UNIQUE INDEX `publisher_id_UNIQUE` (`publisher_id` ASC)  COMMENT '',
  CONSTRAINT `address_id`
    FOREIGN KEY (`address_id`)
    REFERENCES `music_internet_library`.`address` (`address_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `music_internet_library`.`audiobooks`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `music_internet_library`.`audiobooks` (
  `audiobooks_id` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `name_audiobooks` VARCHAR(64) NULL COMMENT '',
  `length` INT UNSIGNED NOT NULL COMMENT '',
  `bitrate` INT UNSIGNED NOT NULL COMMENT '',
  PRIMARY KEY (`audiobooks_id`)  COMMENT '',
  UNIQUE INDEX `audiobooks_id_UNIQUE` (`audiobooks_id` ASC)  COMMENT '')
ENGINE = InnoDB;
-- -----------------------------------------------------
-- Table `music_internet_library`.`book`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `music_internet_library`.`book` (
  `book_id` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
```

```
  `author_id` INT UNSIGNED NOT NULL COMMENT '',
  `genre_id` INT UNSIGNED NOT NULL COMMENT '',
  `cost_id` INT UNSIGNED NOT NULL COMMENT '',
  `publisher_id` INT UNSIGNED NOT NULL COMMENT '',
  `audiobooks_id` INT UNSIGNED NOT NULL COMMENT '',
  `name_book` VARCHAR(64) NULL COMMENT '',
  `book_type` VARCHAR(32) NULL COMMENT '',
  `book_format` VARCHAR(16) NULL COMMENT '',
  `release_date` DATE NULL COMMENT '',
  `number_pages` INT UNSIGNED NOT NULL COMMENT '',
  `book_tome` INT UNSIGNED NOT NULL COMMENT '',
  `language_book` VARCHAR(32) NULL COMMENT '',
  `book_numbers` INT UNSIGNED NULL COMMENT '',
  PRIMARY KEY (`book_id`)  COMMENT '',
  INDEX `author_id_idx` (`author_id` ASC)  COMMENT '',
  INDEX `genre_id_idx` (`genre_id` ASC)  COMMENT '',
  INDEX `cost_id_idx` (`cost_id` ASC)  COMMENT '',
  INDEX `publisher_id_idx` (`publisher_id` ASC)  COMMENT '',
  UNIQUE INDEX `book_id_UNIQUE` (`book_id` ASC)  COMMENT '',
  INDEX `audiobooks_id_idx` (`audiobooks_id` ASC)  COMMENT '',
  CONSTRAINT `author_id`
    FOREIGN KEY (`author_id`)
    REFERENCES `music_internet_library`.`author` (`author_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `genre_id`
    FOREIGN KEY (`genre_id`)
    REFERENCES `music_internet_library`.`genre` (`genre_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `cost_id`
    FOREIGN KEY (`cost_id`)
    REFERENCES `music_internet_library`.`cost` (`cost_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `publisher_id`
    FOREIGN KEY (`publisher_id`)
    REFERENCES `music_internet_library`.`publisher` (`publisher_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `audiobooks_id`
    FOREIGN KEY (`audiobooks_id`)
    REFERENCES `music_internet_library`.`audiobooks` (`audiobooks_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `music_internet_library`.`user`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `music_internet_library`.`user` (
  `user_id` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `book_id` INT UNSIGNED NOT NULL COMMENT '',
  `site_admin_id` INT UNSIGNED NOT NULL COMMENT '',
  `history_pay_id` INT UNSIGNED NOT NULL COMMENT '',
  `country_user` VARCHAR(32) NULL COMMENT '',
  `city_user` VARCHAR(32) NULL COMMENT '',
  `nickname_user` VARCHAR(32) NULL COMMENT '',
  `email_user` VARCHAR(64) NULL COMMENT '',
  `last_name_user` VARCHAR(32) NULL COMMENT '',
  `name_user` VARCHAR(32) NULL COMMENT '',
  `middle_name_user` VARCHAR(32) NULL COMMENT '',
```

```
  `phone_number_user` VARCHAR(32) NULL COMMENT '',
  `register_date_user` DATE NULL COMMENT '',
  PRIMARY KEY (`user_id`)  COMMENT '',
  INDEX `site_admin_id_idx` (`site_admin_id` ASC)  COMMENT '',
  INDEX `book_id_idx` (`book_id` ASC)  COMMENT '',
  UNIQUE INDEX `user_id_UNIQUE` (`user_id` ASC)  COMMENT '',
  CONSTRAINT `site_admin_id`
    FOREIGN KEY (`site_admin_id`)
    REFERENCES `music_internet_library`.`site_admin` (`site_admin_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `book_id`
    FOREIGN KEY (`book_id`)
    REFERENCES `music_internet_library`.`book` (`book_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `music_internet_library`.`history_pay`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `music_internet_library`.`history_pay` (
  `history_pay_id` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `user_id` INT UNSIGNED NOT NULL COMMENT '',
  `balance` INT UNSIGNED NULL COMMENT '',
  PRIMARY KEY (`history_pay_id`)  COMMENT '',
  INDEX `user_id_idx` (`user_id` ASC)  COMMENT '',
  UNIQUE INDEX `history_pay_id_UNIQUE` (`history_pay_id` ASC)  COMMENT '',
  CONSTRAINT `user_id`
    FOREIGN KEY (`user_id`)
    REFERENCES `music_internet_library`.`user` (`user_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `music_internet_library`.`for_example_to_delete`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `music_internet_library`.`for_example_to_delete` (
  `for_example_to_delete_id` INT NOT NULL COMMENT '',
  PRIMARY KEY (`for_example_to_delete_id`)  COMMENT '')
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `music_internet_library`.`blob_test`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `music_internet_library`.`blob_test` (
  `blob_test_id` INT UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '',
  `encrypt_data` VARCHAR(64) NULL COMMENT '',
  PRIMARY KEY (`blob_test_id`)  COMMENT '',
  UNIQUE INDEX `blob_test_id_UNIQUE` (`blob_test_id` ASC)  COMMENT '')
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1 Гольцман, В. MySQL 5.0. Библиотека программиста / В. Гольцман. – СПб. : Санкт-Петербург, 2010. – 253 с.

2 Карпова, И. П. Базы данных : учеб. пособие / И. П. Карпова. – СПб. : Питер, 2013. – 240 с.

3 Хомоненко, А. Д. Базы данных : учебник / А. Д. Хомоненко, В. М. Цыганков, М. Г. Мальцев ; под ред. проф. А. Д. Хомоненко. – 6-е изд., доп. – СПб. : КОРОНА-Век, 2009. – 736 с.

4 Голицына, О. Л. Базы данных : учеб. пособие / О. Л. Голицына, Н. В. Максимов, И. И. Попов. – 3-е изд., перераб. и доп. – М. : ФОРУМ : ИНФРА-М, 2006. – 352 с.

5 Ramez Elmasri. Fundamentals of Database Systems / Ramez Elmasri, Shamkant B. Navathe. – the 6-th edition. – Addison-Wesley, 2010. – 1172 с.

6 Дейт, К. Дж. Введение в системы баз данных / К. Дж. Дейт ; пер. с англ. – 8-е изд.– М. : Изд. дом «Вильямс», 2005. – 1328 с.

7 Роб, П. Системы баз данных : проектирование, реализация и управление / П. Роб, К. Коронел ; пер. с англ. – 5-е изд., перераб. и доп. – СПб. : БХВ-Петербург, 2004. – 1040 с.

8 Блинов, И. Н. Java. Методы программирования : учеб.-метод. пособие / И. Н. Блинов, В. С. Романчик. – Минск : Четыре четверти, 2013. – 896 с.

9 Горшков, Д. А. Исследование современных методов проектирования базы данных / Д. А. Горшков, Л. А. Кутепова // Успехи современного естествознания. – 2011. – №7 – С. 98.

10 СТП 01–2013. Дипломные проекты (работы). Общие технические требования [Электронный ресурс]. – 2013. – Режим доступа : https://www.bsuir.by/m/12_100229_1_80040.pdf.

Св. план 2016, поз. 6

Учебное издание

Алексеев Виктор Федорович
Богатко Иван Николаевич
Пискун Геннадий Адамович

СТРУКТУРЫ И БАЗЫ ДАННЫХ.

ПОСОБИЕ ДЛЯ КУРСОВОГО ПРОЕКТИРОВАНИЯ

ПОСОБИЕ

Редактор *М. А. Зайцева*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *В. М. Задоля*

Подписано в печать 20.10.2017. Формат 60x84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 5,0. Уч.-изд. л. 4,8. Тираж 80 экз. Заказ 418.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6