**Содержание**

[Введение 2](#_Toc60322509)

[1.Логическое проектирование структуры базы данных 3](#_Toc60322510)

[2.Физическое проектирование базы данных 6](#_Toc60322511)

[2.1. Описание структуры таблиц 6](#_Toc60322512)

[2.2. Описание создания таблиц базы данных businessTripCounter 8](#_Toc60322513)

[3.Архитектура приложения 10](#_Toc60322514)

[4.Документация по используемым функциям приложения 13](#_Toc60322515)

[5.Руководство пользователя 14](#_Toc60322516)

[5.1Назначение приложения 14](#_Toc60322517)

[5.2Условия выполнения приложения 14](#_Toc60322518)

[5.3Выполнение приложения 14](#_Toc60322519)

[6.Результаты тестирования разработанного приложения 15](#_Toc60322520)

[Заключение 18](#_Toc60322521)

[Список использованных источников 19](#_Toc60322522)

Введение

Эффективность функционирования предприятия или организации любой отрасли и сферы деятельности напрямую зависит от скорости, точности и своевременности обмена данными как внутри этого предприятия между его составляющими частями (отделами, подсистемами и т.д.), так и вне его, то есть взаимодействие и обмен данными этой организации с другими (конкурирующими, предприятиями-партнерами и т.д.). И чем больше, масштабнее предприятие, тем серьезнее перед его управляющими встает проблема организации и контроля потоков огромного количества информации предприятия.

Чтобы обеспечить взаимодействия человека с персональным компьютером в интерактивном режиме, стали разрабатываться автоматизированные информационные системы, которые является совокупностью аппаратных и программных средств, которые обеспечивают взаимодействия человека и компьютера, а обеспечивают следующими функциями: возможность ввода в ПК и возможность вывода информации, как на экран, так и на устройства вывода.

Целью данной курсовой работы является разработка приложения учета командировок сотрудников.

# 1.Логическое проектирование структуры базы данных

Для логического проектирования структуры базы данных возможно использование одной из нескольких стратегий проектирования:

1. Нормализация схем отношений. Нормализация предусматривает идентификацию требуемых атрибутов с последующим созданием из них нормализованных таблиц, основанных на функциональных зависимостях между этими атрибутами. Процесс проектирования представляет собой процесс нормализации.
2. Концепция модели «сущность-связь». Начинается этот подход с разработки моделей данных, которые содержат несколько высокоуровневых сущностей и связей, затем работа продолжается в виде серии нисходящих уточнений низкоуровневых сущностей, связей и относящихся к ним атрибутов.
3. Подход «от общего к частному» напоминает нормализацию, но отличается от него тем, что вначале выявляется набор основных сущностей с последующим расширением круга рассматриваемых сущностей, связей и атрибутов, которые взаимодействуют с первоначально определенными сущностями.
4. «Смешанная» стратегия проектирования предполагает использование первых двух стратегий для создания разных частей модели, после чего все подготовленные фрагменты собираются в единое целое.

Воспользуемся «смешанной» стратегией проектирования.

На первом этапе выделим сущности и опишем их реквизитный состав, на втором – сформируем отношения, уточним количество этих отношений и их атрибутный состав. Затем проверим отношения на соответствие 3 НФ.

* 1. Выделение сущностей и описание их реквизитного состава.

В задаче учета командировок сотрудников можно выделить следующие сущности:

Сущность «Должность», которая характеризуется реквизитами: идентификатор сущности, название должности, заработная плата.

Сущность «Сотрудник», которая характеризуется реквизитами: идентификатор сущности, фио сотрудника, его должность.

Сущность «Тип расходов», которая характеризуется реквизитами: название типа.

Сущность «Расходы», которая характеризуется реквизитами: идентификатор сущности, номер поездки, тип расходов, количество расходов.

Сущность «Командировка», которая характеризуется реквизитами: идентификатор сущности, номер сотрудника, дата начала командировки, количество дней командировки, аванс.

* 1. Подготовка к применению ER-метода логического проектирования.

Использование данного метода возможно тогда, когда в результате выполнения концептуального проектирования БД уже выполнены следующие действия:

выделены все сущности, информация о которых должна содержаться в искомой БД;

определены основные атрибуты для каждой сущности;

назначен ключевой атрибут для каждой сущности;

сформулированы связи между выделенными сущностями.

Итак, для применения ER-метода назначим ключевые атрибуты для каждой сущности и сформулируем связи между выделенными сущностями:

*Должность (Ид, Название, Зарплата);*

*Сотрудник (Ид, Фио, КодДолжности);*

*Командировка (Ид, КодСотрудника, ДатаНачала, КоличествоДней, Аванс);*

*Расход (Ид, КодКомандировки, КодРасхода, Количество);*

*ТипРасхода (Название).*

Сущности *«Тип расхода»* и *«Расход»* соотносятся с помощью *связи Относится*.

Сущности *«Расход»* и *«Командировка»* соотносятся с помощью *связи Включает*.

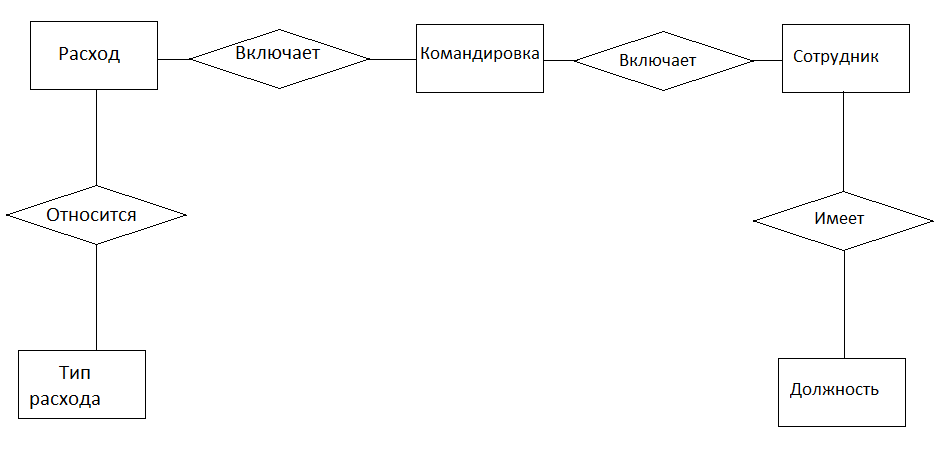
Сущности *«Командировка»* и *«Сотрудник»* соотносятся с помощью *связи Включает*.

Сущности *«Сотрудник»* и *«Должность* соотносятся с помощью *связи имеет*.

* 1. Использование ER-метода логического проектирования.

С помощью указанного метода логического проектирования уточним количество таблиц-отношений и их атрибуты.

Построим ER-диаграмму для проектируемой задачи (рис. 2.1).



**Рисунок 2.1. –Диаграмма ER-типа проектируемой базы**

Таким образом, искомая БД состоит из четырех отношений:

*Должность (****Ид****, Название, Зарплата);*

*Сотрудник (****Ид****, Фио,* ***КодДолжности****);*

*Командировка (****Ид****,* ***КодСотрудника****, ДатаНачала, КоличествоДней, Аванс);*

*Расход (Ид,* ***КодКомандировки****,* ***ТипРасхода****, Количество);*

*ТипРасхода (****Название****).*

У всех таблиц, за исключением типа расхода, первичным ключом является одно поле индивидуальный идентификатор (ид). У таблицы типа расхода – название.

Все получившиеся таблицы удовлетворяют требованиям 3НФ (см. Приложение Б). Эта степень нормализации вполне достаточна для разрабатываемой базы данных, поэтому будем считать, что процесс проектирования заданной БД завершен.

# 2.Физическое проектирование базы данных

## 2.1. Описание структуры таблиц

Опишем структуру таблиц, полученных в ходе логического проектирования, в применении к конкретной СУБД (СУБД MySQL).

Таблица 2.1 – Структура таблиц базы данных

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя таблицы** | **Столбец** | **Описание столбца** | **Комментарий** |
| position | Id | integer(10), unsigned, AUTO\_INCREMENT,  PRIMARY KEY | код сущности, счетчик, первичный ключ |
|  | Name | varchar(255) | название должности |
|  | Salary | float(10) | зарплата должности |
| employee | Id | integer(10), unsigned, AUTO\_INCREMENT,  PRIMARY KEY | код сущности, счетчик, первичный ключ |
|  | Fio | varchar(255) | фио сотрудника |
|  | Position | integer(10), FOREIGN KEY | должность сорудника |
| businesstrip | Id | integer(10), unsigned, AUTO\_INCREMENT,  PRIMARY KEY | код сущности, счетчик, первичный ключ |
|  | Employee | integer(10), FOREIGN KEY | код сотрудника |
|  | StartDate | Date | дата начала командировки |
|  | dayCount | integer(10) | количество дней |
|  | predPaiment | float(10) | аванс |
| expense | Id | integer(10), unsigned, AUTO\_INCREMENT,  PRIMARY KEY | код сущности, счетчик, первичный ключ |
|  | businessTrip | integer(10), FOREIGN KEY | код командировки |
|  | Type | integer(10), FOREIGN KEY | тип растрат |
| expenstype | name | varchar(255), PRIMARY KEY | первичный ключ |

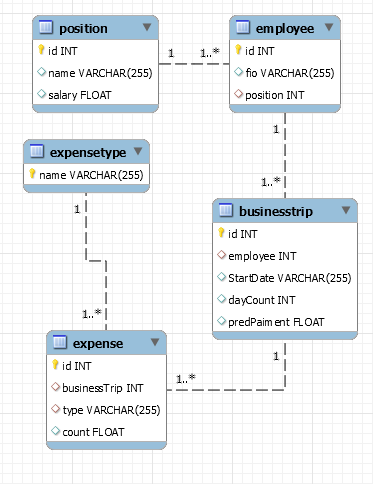
Для создания базы данных businessTripCounter в MySql Workbench возможно использование одного из двух подходов: прямое проектирование (FORVARD INGINEER) и обратное (REVERSE INGINEER).

В первом случае процесс получения структуры базы данных для выбранной СУБД осуществляется на основе построенной ER-модели.

Другими словами, вначале создается графическая модель базы данных и на ее основе строится физическая модель базы данных: структура таблиц и связи между ними.

Во втором случае – выполняется создание ER-модели на основе уже созданной базы данных. Т.е. вначале описывается структура таблиц и связи, а затем создается графическая модель базы данных.

Графическая модель базы данных businessTripCounter представлена на рисунке 2.1.



**Рисунок 2.1 – Графическое представление базы данных в нотации UML**

## 2.2. Описание создания таблиц базы данных businessTripCounter

Для создания базы данных и всех ее таблиц, используется следующий код:

DROP DATABASE IF EXISTS `businessTripCounter`;

CREATE DATABASE IF NOT EXISTS `businessTripCounter`;

USE `businessTripCounter`;

DROP TABLE IF EXISTS `ExpenseType`;

CREATE TABLE `ExpenseType` (

`name` VARCHAR(255) NOT NULL,

PRIMARY KEY (`name`)

) ENGINE=innodb DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `Position`;

CREATE TABLE `Position` (

`id` INTEGER NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(255),

`salary` FLOAT NULL,

PRIMARY KEY (`id`)

) ENGINE=innodb DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `Employee`;

CREATE TABLE `Employee` (

`id` INTEGER NOT NULL AUTO\_INCREMENT,

`fio` VARCHAR(255),

`position` INTEGER,

PRIMARY KEY (`id`),

FOREIGN KEY (`position`) REFERENCES `Position`(`id`)

) ENGINE=innodb DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `BusinessTrip`;

CREATE TABLE `BusinessTrip` (

`id` INTEGER NOT NULL AUTO\_INCREMENT,

`employee` INTEGER,

`StartDate` VARCHAR(255),

`dayCount` INTEGER,

`predPaiment` FLOAT NULL,

PRIMARY KEY (`id`),

FOREIGN KEY (`employee`) REFERENCES `Employee`(`id`)

) ENGINE=innodb DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `Expense`;

CREATE TABLE `Expense` (

`id` INTEGER NOT NULL AUTO\_INCREMENT,

`businessTrip` INTEGER,

`type` VARCHAR(255),

`count` FLOAT NULL,

PRIMARY KEY (`id`),

FOREIGN KEY (`businessTrip`) REFERENCES `BusinessTrip`(`id`),

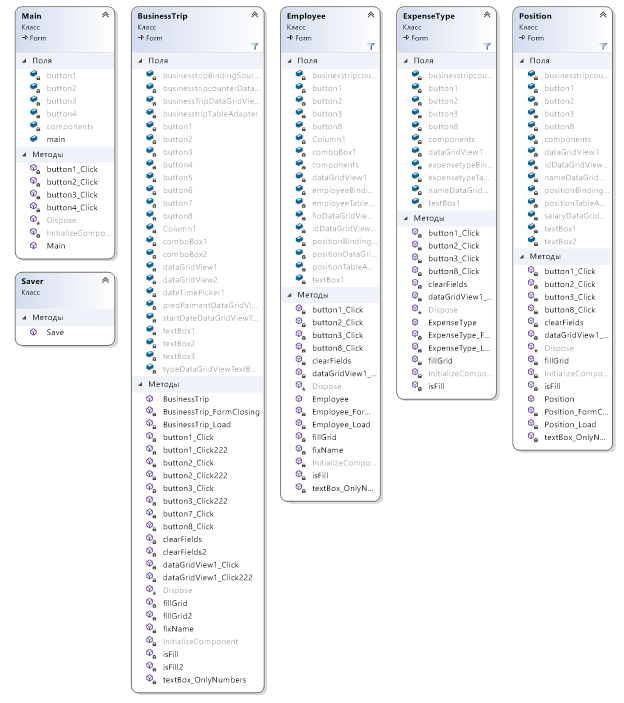
FOREIGN KEY (`type`) REFERENCES `ExpenseType`(`name`)

) ENGINE=innodb DEFAULT CHARSET=utf8;

# 3.Архитектура приложения

Каркас приложения выполнен в архитектуре MVC.

Модель данных представлена в виде набора классов (рисунки 4.1-4.4).



**Рисунок 3.1 – Список классов приложения**

Эти классы выступают в роли контроллеров форм. Таким образом, в них описывается взаимодействие клиента с интерфейсом и ответы на них.

Для отображения (View) каждой формы, используются внутренние данные каждого класса, оформленным отдельным классом с именем *НазваниеКласса.Designer.cs.*

В паттерне «модель – представление – контроллер» модель представляет данные приложения и связанную с ними бизнес-логику.

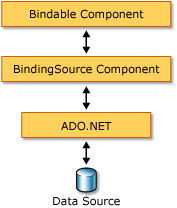
Данные передаются в БД и из нее в объектах передачи данных, и к ним обращаются с помощью объектов доступа к данным.

Представление – это наглядное отображение содержащихся в модели данных. Подмножество модели содержится в отдельном представлении, таким образом, представление действует в качества фильтра для данных модели. Пользователь взаимодействует с данными модели с помощью предлагаемого представлением наглядного отображения и обращается к бизнес логике, которая, в свою очередь, воздействует на данные модели.

Контроллер связывает представление с моделью и управляет потоками данных приложения. Он выбирает, какое представление визуализировать для пользователя в ответ на вводимые им данные и в соответствии с выполняемой бизнес-логикой. Контроллер получает сообщение от представления и пересылает его модели. Модель, в свою очередь, подготавливает ответ и отправляет его обратно контроллеру, где происходит выбор представления и отправка его пользователю.

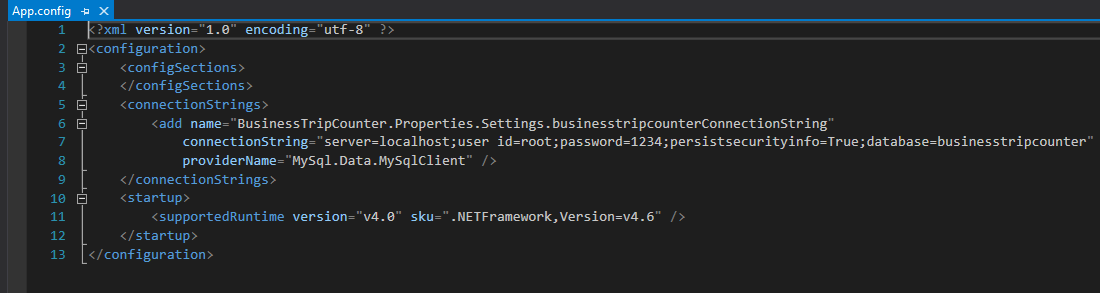
Компонент BindingSource выступает в качестве источника данных для некоторых или всех элементов управления формы. В Visual Studio объект BindingSource можно привязать к элементу управления с помощью DataBindings свойства, доступного в окне Свойства .

Компонент BindingSource можно привязать как к источникам простых данных, например одиночному свойству объекта или базовой коллекции, такому как ArrayList, так и к источникам сложных данных, таким как таблица базы данных. Компонент BindingSource является посредником, обеспечивающим привязку и управление валютой. Во время разработки или во время выполнения компонент BindingSource можно привязать к источнику сложных данных, указав в качестве значений его свойств DataSource и DataMember базу данных и таблицу. На следующем рисунке показано, как компонент BindingSource встраивается в существующую архитектуру привязки данных.



**Рисунок 3.2 – Встраивание компонента в архитектуру**

Доступ к данным в СУБД осуществляться при помощи настраиваемого файла конфигурации.



**Рисунок 3.3 – Файл app.config**

В разрабатываемом приложении обеспечена возможность добавления, редактирования и удаления записей из базы данных, сохранение табличных результатов в excel-файле.

# 4.Документация по используемым функциям приложения

## 4.1Описание класса BusinessTrip

Опишем методы класса BusinessTrip:

* BusinessTrip\_Load – метод, при загрузке формы
* BusinessTrip\_FormClosing – метод, при закрытии формы, возвращает предыдущую на экран
* fixName – метод, исправляющий отображение имен сотрудников, поскольку из базы данных получаем только ид
* textBox\_OnlyNumbers – метод, позволяющий вводить в поле только цифры
* fillGrid – метод, обновляющий данные в таблице командировки
* isFill – метод, проводящий проверку на заполнение полей данных командировки
* clearFields – метод, очищающий все поля командировки
* button1\_Click – метод добавления строки командировки в бд
* button2\_Click – метод редактирование строки командировки в бд
* button3\_Click – метод, удаление строки командировки с бд
* dataGridView1\_Click – метод, заполняющий поля форм при выбора записи в таблице
* fillGrid2 – метод, обновляющий таблицу расходов
* isFill2 – метод, проверяющий заполнение полей расходов
* clearFields2 – метод очистки полей расходов
* button1\_Click222 – метод добавления строки расчета
* button2\_Click222 – метод редактирования строки расчета
* button3\_Click222 – метод удаления строки расчета
* dataGridView1\_Click222 – метод заполняющий поля расчета выбранными данными из таблицы
* button7\_Click – метод сохранения таблицы командировки
* button8\_Click – метод сохранения таблицы расходов

# 5.Руководство пользователя

## 5.1Назначение приложения

Данное приложение предназначено для автоматизации ведения учета командировок сотрудников. Пользователям доступны операции по добавлению, изменению, удалению данных о командировках, сотрудниках, должностях, расходах по командировках и типов расходов.

## 5.2Условия выполнения приложения

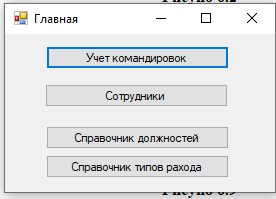
Исполняемый файл BusinessTripCounter занимает 119 КБ (вместе со всеми библиотеками 6.36 МБ) памяти и может выполняться на любом персональном компьютере, на котором установлено Windows 7 и выше, MicrosoftOffice, .NET, MySQL 8.

## 5.3Выполнение приложения

Для получения доступа к приложению требуется запустить файл BusinessTripCounter.exe и дождаться появления главного окна приложения.

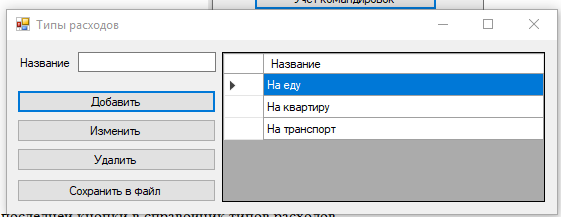
# 6.Результаты тестирования разработанного приложения

При входе в приложения мы видим следующую форму



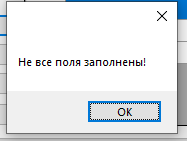
**Рисунок 6.1 – Главная форма**

Первым делом стоит заполнить типы расходов. Для этого перейдем с помощью последней кнопки в справочник типов расходов.



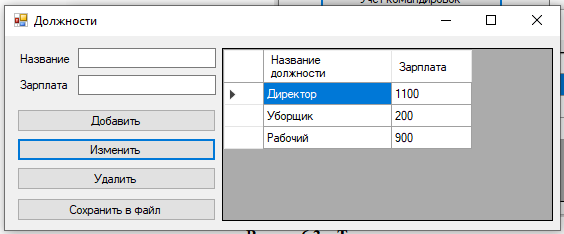
**Рисунок 6.2 – Типы расходов**

На этой форме можно выполнить основные операции с данными таблицы. При отсутствии заполнения поля для добавления, приложение выдаст соответствующую ошибку.



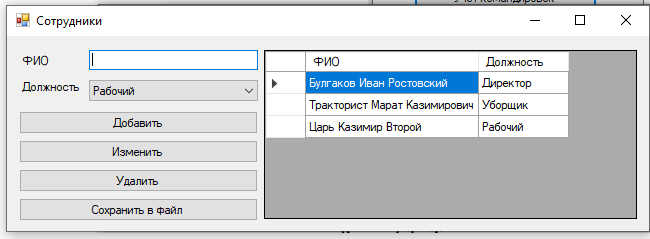
**Рисунок 6.3 – Ошибка при незаполненных полях**

Можно также перейти на страницу должностей, где работа уже пойдет непосредственно с ними.



**Рисунок 6.4 – Должности**

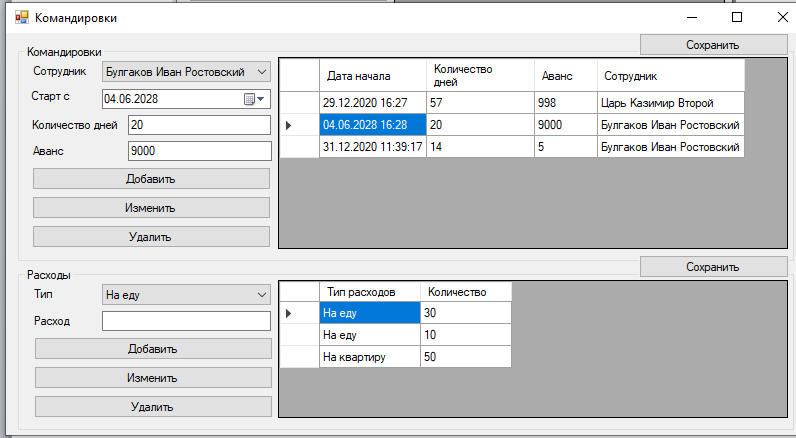
При работе с таблицей сотрудников уже необходимо выбирать его должность из списка, который мы добавляется ранее.



**Рисунокк 6.5 – Сотрудники**

Последняя работа с формой командировки, на которой работаем сразу с двумя таблицами: командировки и расходы.

Сперва создается командировка с выбором определенного сотрудника, количества дней, даты старта и аванса. После добавления командировки, в нижней таблицы отражаются все расходы по данной командировке.



**Рисунок 6.6 – Командировки и расходы**

# Заключение

В ходе выполнения данной курсовой работы было разработано приложение по учетe командировок сотрудников, закреплены теоретические знания и отработаны практические навыки объектно-ориентированного проектирования и программирования на языке C#, изучена литература по объектно-ориентированному программированию, проведено обучение работе в интегрированной среде программирования Visual Studio 2019..

В рамках разработки приложения было проведено логическое проектирование структуры базы данных, физическое проектирование базы данных. Была рассмотрена архитектура MVC.

В практической части работы было разработано приложение для учёта командировок сотрудников при помощи языка программирования высокого уровня С#.

# Список использованных источников

1. Коваленко, Е.А. Базы и банки данных: практическое пособие для студентов специальности I – 53 01 02 «Автоматизированные системы обработки информации» / Е.А.Коваленко, В.Н.Леванцов. – Гомель: учреждение образования «Гомельский государственный университет имени Франциска Скорины», 2007. – 95 с.

2. Лекция №4 – Проектирование баз данных Подходы к проектированию базы данных // StudFiles [Электронный ресурс]. – 2019. – Режим доступа: https://studfile.net/preview/1705211/. – Дата доступа: 31.12.2020.3. MySQL/Руководство для начинающих [Электронный ресурс]. – Режим доступа: https://wiki.gentoo.org/wiki/MySQL/Startup\_Guide/ru. – Дата доступа: 31.12.2020.

4. Руководство по MVC [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/mvc5/. – Дата доступа: 31.12.2020.

**ПРИЛОЖЕНИЕ А**

**Правила генерации отношений по диаграммам ER-типа**

**ПРАВИЛО 1.** Если показатель кардинальности бинарной связи равен 1:1 и класс принадлежности обеих сущностей является обязательным, то требуется только **одно** отношение. Первичным ключом этого отношения может быть ключ любой из двух сущностей.

**ПРАВИЛО 2.** Если показатель кардинальности бинарной связи равен 1:1 и класс принадлежности одной сущности является обязательным, а другой – необязательным, то необходимо построение двух отношений. Под каждую сущность необходимо выделение одного отношения, при этом ключ сущности должен служить первичным ключом для соответствующего отношения. Кроме того, ключ сущности, для которой класс принадлежности является необязательным, добавляется в качестве атрибута в отношение, выделенное для сущности с обязательным классом принадлежности.

**ПРАВИЛО 3.** Если показатель кардинальности бинарной связи равен1:1 и класс принадлежности ни одной сущности не является обязательным, то необходимо использовать три отношения: по одному для каждой сущности, ключи которых служат в качестве первичных в соответствующих отношениях, и одного для связи. Среди своих атрибутов отношение, выделяемое связи, будет иметь по одному ключу сущности от каждой сущности.

**ПРАВИЛО 4.** Если показатель кардинальности бинарной связи равен 1:n и класс принадлежности n-связной сущности является обязательным, то достаточным является использование двух отношений, по одному на каждую сущность, при условии, что ключ сущности каждой сущности служит в качестве первичного ключа для соответствующего отношения. Дополнительно ключ односвязной сущности должен быть добавлен как атрибут в отношение, отводимое n-связной сущности.

**ПРАВИЛО 5.** Если показатель кардинальности бинарной связи равен 1:n и класс принадлежности n‑связной сущности является необязательным, то необходимо формирование трех отношений: по одному для каждой сущности, причем ключ каждой сущности служит первичным ключом соответствующего отношения, и одного отношения для связи. Связь должна иметь среди своих атрибутов ключ сущности от каждой сущности.

**ПРАВИЛО 6.** Если показатель кардинальности бинарной связи равен m:n, то для хранения данных необходимо три отношения: по одному для каждой сущности, причем ключ каждой сущности используется в качестве первичного ключа соответствующего отношения, и одного отношения для связи. Последнее отношение должно иметь в числе своих атрибутов ключ сущности каждой сущности.

**ПРИЛОЖЕНИЕ Б**

**Описание нормальных форм**

В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм:

* первая нормальная форма (1NF);
* вторая нормальная форма (2NF);
* третья нормальная форма (3NF);
* нормальная форма Бойса-Кодда (BCNF);
* четвертая нормальная форма (4NF);
* пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

1НФ – таблица находится в первой нормальной форме (1НФ) тогда и только тогда, когда ни одна из ее строк не содержит в любом своем поле более одного значения и ни одно из ее ключевых полей не пусто. (Любое поле таблицы содержит неделимую информацию и в таблице определен первичный ключ).

2НФ – Таблица находится во второй нормальной форме (2NF) в том и только в том случае, когда находится в 1NF, и каждый ее не ключевой атрибут полностью зависит от первичного ключа.

Если таблица в 1 НФ имеет простой первичный ключ, то она автоматически находится и во 2 НФ.

3НФ – Таблица находится в третьей нормальной форме (3NF) в том и только в том случае, если находится в 2NF и каждый не ключевой атрибут не транзитивно зависит от первичного ключа. (Иными словами, таблица должна находиться во второй нормальной форме и ни одно из ее не ключевых полей не должно однозначно идентифицироваться значением другого не ключевого поля (полей)).

Теоретики реляционных систем Кодд и Бойс обосновали и предложили более строгое определение для 3НФ, которое учитывает, что в таблице может быть несколько возможных ключей. Таблица, соответствующая этому определению называется таблицей в улучшенной третьей форме или таблицей в нормальной форме Бойса-Кодда.

Таблица находится в нормальной форме Бойса-Кодда (НФБК), если и только если любая функциональная зависимость между его полями сводится к полной функциональной зависимости от возможного ключа.

В следующих нормальных формах (4НФ и 5НФ) учитываются не только функциональные, но и многозначные зависимости между полями таблицы. Для их описания познакомимся с понятием полной декомпозиции таблицы.

Полной декомпозицией таблицы называют такую совокупность произвольного числа ее проекций, соединение которых полностью совпадает с содержимым таблицы.

Теперь можно дать определения высших нормальных форм. И сначала будет дано определение для последней из предложенных - 5НФ.

5НФ – Таблица находится в пятой нормальной форме тогда и только тогда, когда в каждой ее полной декомпозиции все проекции содержат возможный ключ. Таблица, не имеющая ни одной полной декомпозиции, также находится в 5НФ.

4НФ – Четвертая нормальная форма является частным случаем 5НФ, когда полная декомпозиция должна быть соединением ровно двух проекций. Весьма не просто подобрать реальную таблицу, которая находилась бы в 4НФ, но не была бы в 5НФ.

Третья нормальная форма считается оптимальной для небольших баз данных, при проведении дальнейшей нормализации следует учитывать, что при увеличении количества связанных таблиц увеличивается время обработки информации, хранящейся в них. Поэтому разработчик должен сам принимать решение о том, какая ступень нормализации будет оптимальной для проектируемой базы данных.

ПРИЛОЖЕНИЕ В

**Содержание файла BusinessTripCounter.java**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace BusinessTripCounter

{

public partial class BusinessTrip : Form

{

public BusinessTrip()

{

InitializeComponent();

}

/// <summary>

/// Загрузка формы

/// </summary>

/// <param name="sender">входящий объект</param>

/// <param name="e">Входящее событие</param>

private void BusinessTrip\_Load(object sender, EventArgs e)

{

this.employeeTableAdapter.Fill(this.businesstripcounterDataSet.employee);

this.expensetypeTableAdapter.Fill(this.businesstripcounterDataSet.expensetype);

this.expenseTableAdapter.Fill(this.businesstripcounterDataSet.expense);

this.businesstripTableAdapter.Fill(this.businesstripcounterDataSet.businesstrip);

fixName();

}

/// <summary>

/// Метод закрытия формы

/// </summary>

/// <param name="sender">входящий объект</param>

/// <param name="e">Входящее событие</param>

private void BusinessTrip\_FormClosing(object sender, FormClosingEventArgs e)

{

Main.main.Show();

}

/// <summary>

/// Метод изменение имен на форме

/// </summary>

private void fixName()

{

for (int i = 0; i < dataGridView1.RowCount; i++)

{

comboBox1.SelectedItem = comboBox1.Items[

employeeBindingSource.Find(

"id",

int.Parse(dataGridView1[1, i].Value.ToString())

)

];

dataGridView1[5, i].Value = comboBox1.Text;

}

}

/// <summary>

/// Вставка только цифр

/// </summary>

/// <param name="sender">входящий объект</param>

/// <param name="e">Входящее событие</param>

private void textBox\_OnlyNumbers(object sender, KeyPressEventArgs e)

{

char number = e.KeyChar;

if (!Char.IsDigit(number) && number != 8) // цифры и клавиша BackSpace

{

e.Handled = true;

}

}

/// <summary>

/// заполнение грида

/// </summary>

private void fillGrid()

{

this.businesstripTableAdapter.Fill(this.businesstripcounterDataSet.businesstrip);

fixName();

}

/// <summary>

/// проверка на заполнение

/// </summary>

/// <returns></returns>

private bool isFill()

{

if (textBox1.Text.Length < 1 || textBox2.Text.Length < 1 || comboBox1.Items.Count < 1)

{

MessageBox.Show("Не все поля заполнены!");

return false;

}

return true;

}

/// <summary>

/// очистка полей

/// </summary>

private void clearFields()

{

textBox1.Text = "";

textBox2.Text = "";

}

//add

/// <summary>

/// добавление строки

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void button1\_Click(object sender, EventArgs e)

{

if (isFill())

try

{

DataRowView row = (DataRowView)businesstripBindingSource.AddNew();

row[1] = comboBox1.SelectedValue;

row[2] = dateTimePicker1.Value;

row[3] = textBox1.Text;

row[4] = textBox2.Text;

businesstripBindingSource.EndEdit();

this.businesstripTableAdapter.Update(businesstripcounterDataSet);

clearFields();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

fillGrid();

}

//edit

/// <summary>

/// редактиирование строки

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void button2\_Click(object sender, EventArgs e)

{

if (isFill())

try

{

dataGridView1.CurrentRow.Cells[1].Value = comboBox1.SelectedValue;

dataGridView1.CurrentRow.Cells[2].Value = dateTimePicker1.Value;

dataGridView1.CurrentRow.Cells[3].Value = textBox1.Text;

dataGridView1.CurrentRow.Cells[4].Value = textBox2.Text;

businesstripBindingSource.EndEdit();

this.businesstripTableAdapter.Update(((DataRowView)dataGridView1.CurrentRow.DataBoundItem).Row);

clearFields();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

fillGrid();

}

//delete

/// <summary>

/// удаление строки

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void button3\_Click(object sender, EventArgs e)

{

if (dataGridView1.Rows.Count > 0 && dataGridView1.CurrentRow != null)

{

try

{

businesstripcounterDataSet.AcceptChanges();

businesstripBindingSource.RemoveAt(dataGridView1.CurrentRow.Index);

businesstripBindingSource.EndEdit();

businesstripTableAdapter.Update(businesstripcounterDataSet.businesstrip);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

fillGrid();

}

}

/// <summary>

/// отображение информации о выбранном

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void dataGridView1\_Click(object sender, EventArgs e)

{

if (dataGridView1.Rows.Count > 0 && dataGridView1.CurrentRow != null)

{

comboBox1.Text = dataGridView1.CurrentRow.Cells[1].Value.ToString();

dateTimePicker1.Value = DateTime.Parse(dataGridView1.CurrentRow.Cells[2].Value.ToString());

textBox1.Text = dataGridView1.CurrentRow.Cells[3].Value.ToString();

textBox2.Text = dataGridView1.CurrentRow.Cells[4].Value.ToString();

}

}

// расходы на командировку

/// <summary>

/// заполнение грида

/// </summary>

private void fillGrid2()

{

this.expenseTableAdapter.Fill(this.businesstripcounterDataSet.expense);

}

/// <summary>

/// проверка на зполнение

/// </summary>

/// <returns></returns>

private bool isFill2()

{

if (textBox3.Text.Length < 1 || comboBox2.Items.Count < 1 || dataGridView1.CurrentRow==null)

{

MessageBox.Show("Не все поля заполнены или не выбрана командировка!");

return false;

}

return true;

}

/// <summary>

/// очистка полей

/// </summary>

private void clearFields2()

{

textBox3.Text = "";

}

//add

/// <summary>

/// добавление строки

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void button1\_Click222(object sender, EventArgs e)

{

if (isFill2())

try

{

DataRowView row = (DataRowView)expenseibfk1BindingSource.AddNew();

row[1] = dataGridView1.CurrentRow.Cells[0].Value;

row[2] = comboBox2.SelectedValue;

row[3] = textBox3.Text;

expenseibfk1BindingSource.EndEdit();

this.expenseTableAdapter.Update(businesstripcounterDataSet);

clearFields2();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

fillGrid2();

}

//edit

/// <summary>

/// редактирование строки

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void button2\_Click222(object sender, EventArgs e)

{

if (isFill2())

try

{

dataGridView2.CurrentRow.Cells[2].Value = comboBox2.SelectedValue;

dataGridView2.CurrentRow.Cells[3].Value = textBox3.Text;

expenseibfk1BindingSource.EndEdit();

this.expenseTableAdapter.Update(((DataRowView)dataGridView2.CurrentRow.DataBoundItem).Row);

clearFields2();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

fillGrid2();

}

//delete

/// <summary>

/// удаление строки

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void button3\_Click222(object sender, EventArgs e)

{

if (dataGridView2.Rows.Count > 0 && dataGridView2.CurrentRow != null)

{

try

{

businesstripcounterDataSet.AcceptChanges();

expenseibfk1BindingSource.RemoveAt(dataGridView2.CurrentRow.Index);

expenseibfk1BindingSource.EndEdit();

expenseTableAdapter.Update(businesstripcounterDataSet.expense);

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

fillGrid2();

}

}

/// <summary>

/// отображение информации о выбранном

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void dataGridView1\_Click222(object sender, EventArgs e)

{

if (dataGridView2.Rows.Count > 0 && dataGridView2.CurrentRow != null)

{

comboBox2.Text = dataGridView2.CurrentRow.Cells[2].Value.ToString();

textBox3.Text = dataGridView2.CurrentRow.Cells[3].Value.ToString();

}

}

/// <summary>

/// сохранене командировок

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void button7\_Click(object sender, EventArgs e)

{

Saver.Save(dataGridView1);

}

/// <summary>

/// сохранение расходов

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void button8\_Click(object sender, EventArgs e)

{

Saver.Save(dataGridView2);

}

}

}