Содержание

[ВВЕДЕНИЕ 5](#_Toc28032388)

[1 АНАЛИТИЧЕСКАЯ ЧАСТЬ 6](#_Toc28032389)

[1.1 Анализ предметной области 6](#_Toc28032390)

[1.2 Постановка задач 6](#_Toc28032391)

[2 Разработка логической модели базы данных 8](#_Toc28032392)

[2.1 Построение диаграммы «сущность-связь» в нотации П.Чена 8](#_Toc28032393)

[2.1.1 Построение сложной сетевой структуры 8](#_Toc28032394)

[2.1.2 Построение простой сетевой структуры 9](#_Toc28032395)

[2.1.3 Построение древовидной структуры 10](#_Toc28032396)

[2.1.4 Построение диаграмма «сущность-связь» в нотации П.Чена на основе предыдущих диаграмм 10](#_Toc28032397)

[2.2 Построение модели, основанной на ключах 11](#_Toc28032398)

[2.3. Построение полной̆ атрибутивной̆ модели в нотации IDEF1X 12](#_Toc28032399)

[2.4 Нормализация БД 13](#_Toc28032400)

[2.4.1 Первая нормальная форма 13](#_Toc28032401)

[2.4.2 Вторая нормальная форма 13](#_Toc28032402)

[2.4.3 Третья нормальная форма 14](#_Toc28032403)

[2.4.4 Нормальная форма Бойса-Кодда 14](#_Toc28032404)

[2.5 Результат приведения БД к 3НФ и НФБК 15](#_Toc28032405)

[3 Разработка физической модели базы данных 17](#_Toc28032406)

[3.1 Выбор аппаратной и программной платформы для реализации БД 17](#_Toc28032407)

[3.2 Реализация базы данных 17](#_Toc28032408)

[3.3 Тестирование базы данных 19](#_Toc28032409)

[3.3.1 Заполнение таблиц 19](#_Toc28032410)

[3.3.2 Запросы для тестирования БД 21](#_Toc28032411)

[3.3.2 Проверка целостности 23](#_Toc28032412)

[3.4 Разграничение прав доступа 24](#_Toc28032413)

[3.5 Расчет информационных параметров базы данных 26](#_Toc28032414)

[4 Разработка клиентского приложения 27](#_Toc28032415)

[4.1 Обоснование выбора языка программирования 27](#_Toc28032416)

[4.2 Разработка интерфейса пользователя 28](#_Toc28032417)

[4.3 Алгоритм работы каждого из модулей 32](#_Toc28032418)

[4.4 Тестирование работы приложения 34](#_Toc28032419)

[ЗАКЛЮЧЕНИЕ 39](#_Toc28032420)

[СПИСОК ИСПОЛЬЗОВАННЫХ СОКРАЩЕНИЙ 41](#_Toc28032421)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 42](#_Toc28032422)

[ПРИЛОЖЕНИЕ А Модель в нотации ченa 43](#_Toc28032423)

[ПРИЛОЖЕНИЕ Б Листинг запросов 44](#_Toc28032424)

[ПРИЛОЖЕНИЕ В Листинг программы 47](#_Toc28032425)

ВВЕДЕНИЕ

В современном мире вся информация, хранимая и использующаяся в процессе работы, должна быть жестко структурирована и упорядочена. Это необходимо для того, чтобы упростить поиск, работу и сохранение данных.

В ходе выполнения курсового проекта разрабатывается физическая модель базы данных для учета и хранения заказов транспортной компании (междугородние пассажирские перевозки) в рамках технического задания.

Целями курсового проектирования является закрепление теоретических знаний, полученных в результате изучения курсов «Теория баз данных» и «Управление данными», формирование практических навыков в области информационного моделирования и проектирования баз данных, разработка физической модели базы данных.

Для достижения цели на разных этапах курсового проектирования должны быть решены следующие задачи:

выбор варианта задания детализация постановки задачи и анализ предметной области;

разработка логической модели базы данных; ¾ разработка и тестирование физической модели БД;

разработка клиентского приложения.

Данная пояснительная записка состоит из: содержания, введения, а также других разделов. Раздел «Аналитическая часть», в котором описывается предметная область. Раздел «Разработка логической модели базы данных», в котором отражены логические связи между сущностями. Раздел «Разработка физической модели базы данных» описывает физические свойства базы данных (типы данных, размер полей, индексы). Раздел «Разработка клиентского приложения» в котором находятся сведения о выбранной платформе реализации, описание взаимодействие модулей, а также описание работы пользователя с ней.

1. АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1 Анализ предметной области

Предметная область представляет собой большую ИС автовокзал, направленную на сбор, обработку информации для предоставления услуг автоперевозок. ИС автовокзала является связующим звеном между поставщиком услуг - автокомпанией и их потребителями - пассажиров. Поэтому для оптимального взаимодействия выделенных сторон, существует необходимость автоматизации информационных процессов, что ведёт к их быстродействию и качеству.

1.2 Постановка задач

Необходимо разработать базу данных "Автовокзал". Проектирумая БД предназначена для ИС диспетчеров автовокзала и обслуживающего персонала, для управления и учёта выездов всех автобусов, для автоматизации продажи билетов на междугородние и пригородные автобусные перевозки на уровне отдельного автовокзала. БД должна решать довольно узкий круг задач, связанный сопоставлением расписания и фактических выездов автобусов по различным маршрутам. Выделены следующие объекты: перевозчик, автобус, сотрудники, билет, населенные пункты, рейс, путевой лист.

Связи между объектами:

перевозчику принадлежат автобусы;

автобус указывается в путевых листах;

сотрудник указывается в билетах, которые продал, и в путевых листах, которые составил;

билет указывается в путевом листе;

насленный пункт указывается в рейсах;

рейс указывается в путевом листе.

Таким образом, были сформированы некоторые требования к разработке логической базы данных, которую необходимо будет реализовать в виде клиентского приложения.

Исходя из аналитической части, а именно из анализа предметной области и постановки задачи, получив необходимые данных (в ходе общения со специалистами, выделены основные сущности, определены первичные связи между ними), можно переходить к разработке логической модели БД.

2 Разработка логической модели базы данных

Проектирование баз данных (БД) – процесс создания схемы базы данных и определения необходимых ограничений целостности. Проектирование базы данных сводится к двум последовательным этапам:

1. Логическое проектирование – преобразование требований к данным в структуры данных.

2. Физическое проектирование – определение особенностей хранения данных, методов доступа и т.д. в зависимости от СУБД.

2.1 Построение диаграммы «сущность-связь» в нотации П.Чена

2.1.1 Построение сложной сетевой структуры

Разработка «сущность-связь» основывается на переходе от сложной сетевой структуры к древовидной и построения диаграммы.

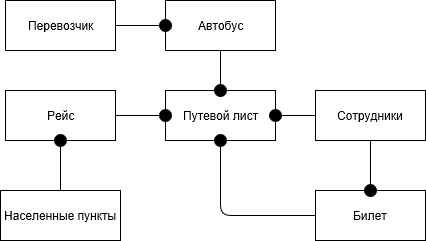


Рисунок 2.1 – Сложная сетевая структура

2.1.2 Построение простой сетевой структуры

Простая сетевая структура создается удалением связей многие ко многим. В данной сетевой структуре таких нет.

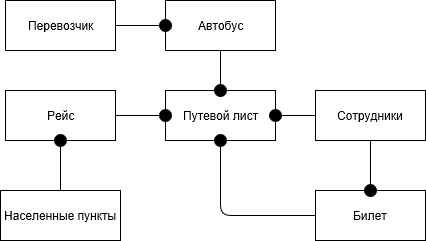


Рисунок 2.2 – Простая сетевая структура

Таким образом, сформированы 7 основных сущностей: «Перевозчик», «Автобус», «Населенные\_пункты», «Рейс», «Путевой\_лист», «Сотрудники», «Билет».

сущность «Перевозчик» содержит код перевозчика, название, адрес, телефон;

сущность «Автобус» содержит государственный номер, марку, модель, количество мест, регистрационный номер, код перевозчика;

сущность «Сотрудники» содержит ИИН, ФИО сотрудника, должность, адрес проживания, контактный телефон, стаж;

сущность «Билет» содержит № билета, дата продажи, стоимость, дата отправления, ИИН сотрудника;

сущность «Рейс» содержит № рейса, время отправления, время прибытия, расстояние, график движения, пункт отправления, код населенного пункта;

сущность «Путевой\_лист» содержит № продажи, № билета, государственный номер, ИИН сотрудника, № рейса, Код населенного пункта, Код перевозчика;

сущность «Населенные\_пункты» содержит Код населенного пункта, название населенного пункта.

Таким образом были выделенны основные сущности БД и их атрибуты.

2.1.3 Построение древовидной структуры

На основе простой сетевой структуры была построена древовидная структура, отображающая взаимосвязь ближайших объектов базы, а также, показывающая будущие внешние ключи.

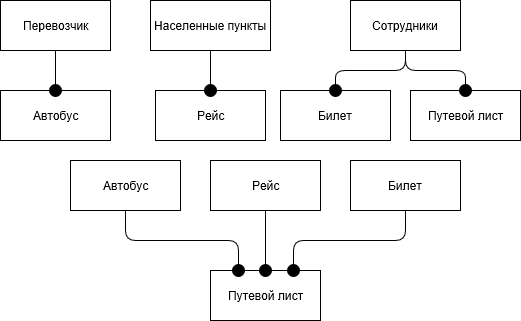


Рисунок 2.3 – Древовидная структура

2.1.4 Построение диаграмма «сущность-связь» в нотации П.Чена на основе предыдущих диаграмм

На основе прошлых таблиц была синтезирована ER диаграмма, отображающая взаимосвязь объектов, а также основные элементы взаимодействующих объектов.

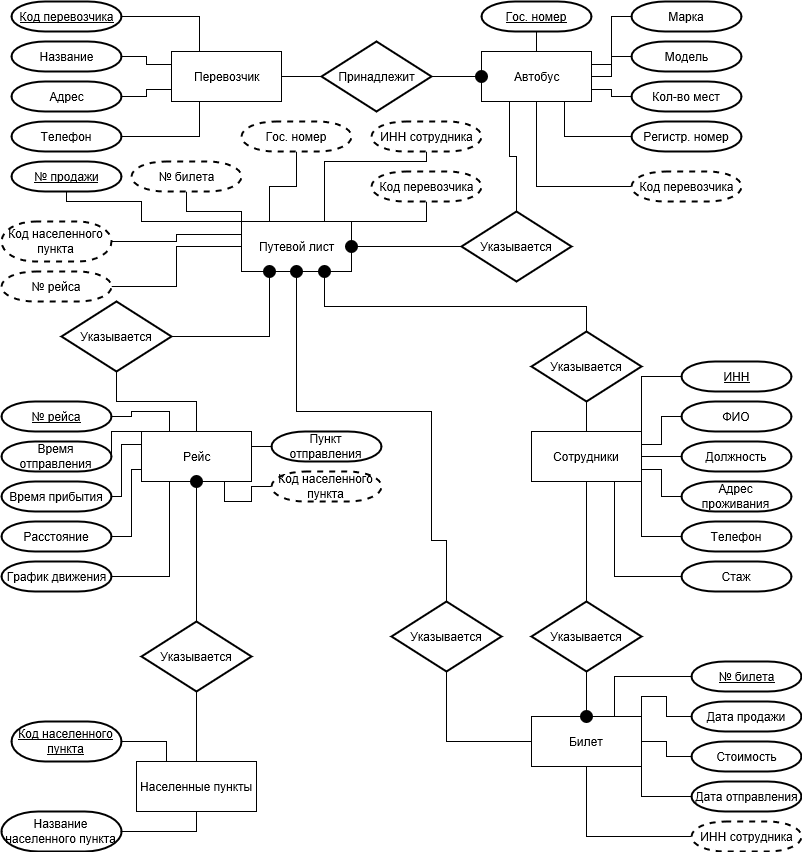


Рисунок 2.4 – Диаграмма «сущность-связь» в нотации П.Чена

2.2 Построение модели, основанной на ключах

Для построения модели основанной на ключах, сущности диаграммы «сущность-связь» были идентифицированы посредствам выбора ключевых атрибутов, которые однозначно определяют каждую из них. Результат - структура, изображенная на рисунке 2.5.

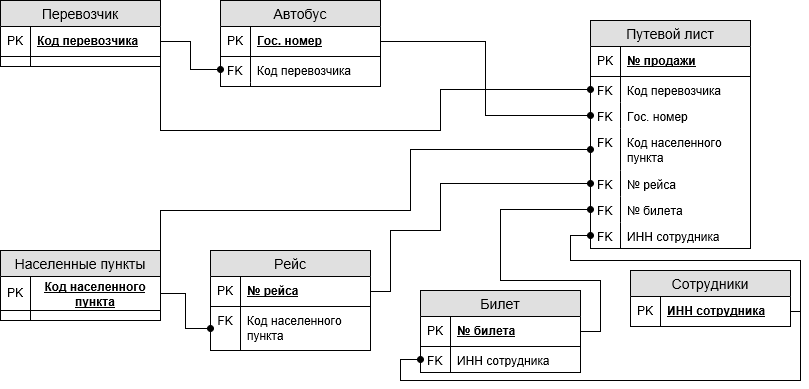


Рисунок 2.5 – Модель, основанная на ключах

Модель, основанная на ключах, строится на основе ERD, но отображает больше деталей: ключевые поля, внешние ключи, таким образом, построение модели, основанной на ключах, позволяет оценить взаимосвязи между сущностями с помощью определение первичных и внешних ключей.

2.3. Построение полной̆ атрибутивной̆ модели в нотации IDEF1X

Полная атрибутивная модель – более подробное представление данных, чем модель, основанная на ключах. Она включает описание всех сущностей, внешних и первичных ключей и предназначена для представления структуры данных и ключей, которые соответствуют предметной области.

На рисунке 2.6 изображена полная атрибутивная модель БД «Автовокзал»

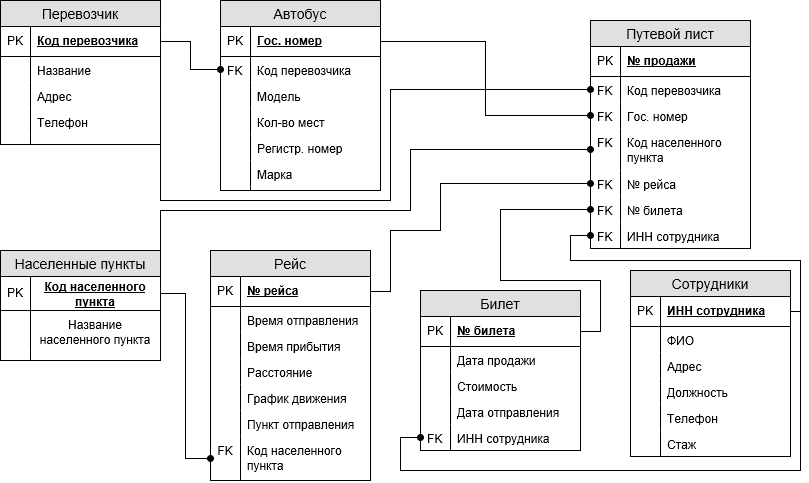


Рисунок 2.6 – Полная атрибутивная модель в нотации IDEFIX

2.4 Нормализация БД

На основе полной̆ атрибутивной̆ модели, можно провести нормализацию отношений. Номализация будет проводится до третьей нормальной , а так же нормальной фомы Бойса-Кодда.

2.4.1 Первая нормальная форма

Чтобы отношение соответствовало первой нормальной форме необходимо привести это отношение в соответствие трем условиям:

1) определить первичные ключи;

2) устранить повторяющиеся группы (одинаковые кортежи);

3) привести поля отношения к атомарности

Исследуя БД можно заметить, что отношение «Перевозчик» не соответствует 1НФ, т. к. поле «Адрес» является составным, оно состоит из «Город», «Улица», «№ дома». Также в отношении «Сотрудники» атрибуты «ФИО» и «Адрес» являются делимыми. Атрибут «Адрес» можно разбить на: «Улица», «Город», «Дом», «Квартира». А «ФИО» - «Фамилия», «Имя», «Отчество». И в отношении «Населенные пункты» «Название населенного пункта» - делимый атрибут, разбиваемый на 2 атрибута «Название страны» и «Название города». Необходима декомпозиция.

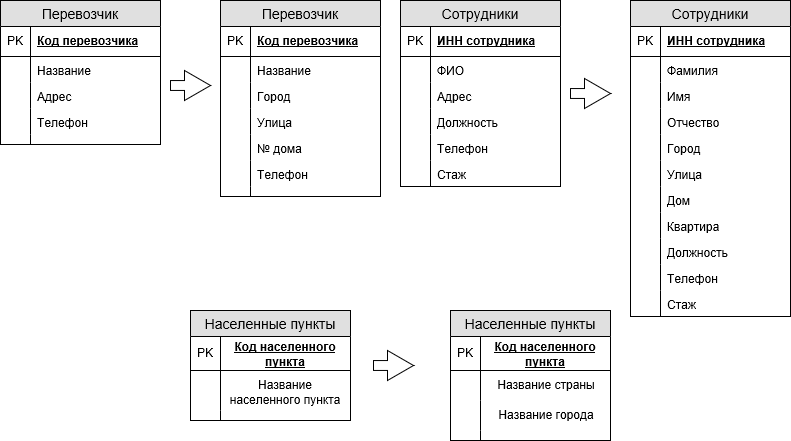


Рисунок 2.7 – Пример декомпозиции для приведения БД к 1НФ

2.4.2 Вторая нормальная форма

Отношение находится во второй нормальной форме, если оно находится в 1НФ, и каждый не ключевой атрибут характеризуется полной функциональной зависимостью от первичного ключа (т.е. все не ключевые атрибуты зависят только от ключа целиком, а не от какой-то его части).

Если посмотреть на БД, то будет видно, что все отношения не имеют составных ключей, следовательно, БД находится в 2НФ.

2.4.3 Третья нормальная форма

Отношение находится в третьей нормальной форме, если оно находится во 2НФ и никакой не ключевой атрибут функционально не зависит от другого не ключевого атрибута, т.е. нет транзитивных зависимостей.

Все неключевые поля таблиц зависят от первичного ключа и не имеют зависимости друг от друга, следовательно, модель приведена к 3НФ

2.4.4 Нормальная форма Бойса-Кодда

Нормальная форма Бойса-Кодда (Далее НФБК) считается уточнением 3НФ. Она учитывает все потенциальные ключи, которые входят в отношения. Если отношение имеет единственный потенциальный ключ, то 3НФ и НФБК – эквивалентны. Считается, что отношение, находящееся в НФБК, если каждый его детерминант является потенциальным ключом. Чтобы убедиться, что отношение находится в НФБК необходимо отыскать все его детерминанты и убедиться, что они являются потенциальными ключами

БД находится в НФБК т. к. в каждом отношении все поля можно объединить в составной первичный ключ, т. е. все поля являются потенциальными ключами.

2.5 Результат приведения БД к 3НФ и НФБК

На рисунке 2.8 изображена полная атрибутивная модель БД в нотации IDEFIX в 3НФ и НФБК.

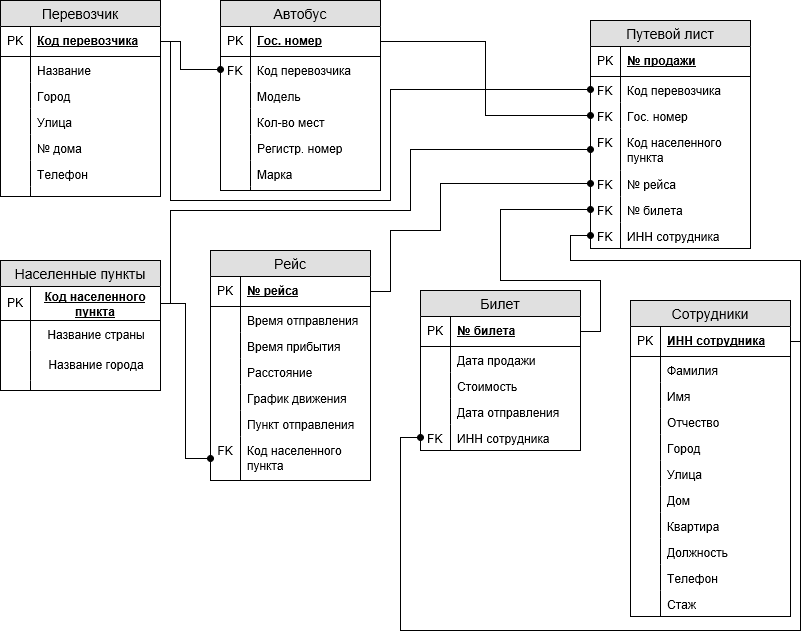


Рисунок 2.8 - Полная атрибутивная модель в нотации IDEFIX в 3НФ и НФБК

В разделе 2 были разработаны сложная сетевая сруктура, на ее основе была спроектирована простая сетевая структура, далее древовидная структура, показывающая внешние ключи, связывающие сущности и, наконец, диграмма «сущность связь», показывающая будужие атрибуты сущностей, и их взамодействия с другими сущностями. Также была подробно рассмотрена нормализация отношений от 1НФ к нормальной форме Бойса-Кодда включительно. Построение логической модели аргументировано приведением к нормальной форме Бойса-Кодда для улучшения структура базы данных перед реализацией.

3 Разработка физической модели базы данных

3.1 Выбор аппаратной и программной платформы для реализации БД

В качестве СУБД служит MySQL, которая была выбрана из-за того, что к этой базе относительно просто подключаться, есть встроенный графический интерфейс, который позволяет ускорить работу с вводом и выводом данных, бесплатная лицензия для некоммерческих проектов. Данная СУБД была выбрана из-за наличия следующих достоинств:

Весьма высокое быстродействие;

Высокий уровень безопасности;

Бесплатное распространение программы для некоммерческих целей;

Открытость кода. Благодаря этому вы сможете сами добавлять в пакет нужные функции, расширяя его функциональность так, как вам требуется;

Переносимость. Существуют версии программы для большинства распространенных компьютерных платформ. Это говорит о том, что не навязывается определенная операционная система;

Бд будет реализована в Open Server Panel - портативная серверная платформа и программная среда, созданная специально для веб-разработчиков

3.2 Реализация базы данных

Полный листинг SQL скриптов представлен в разделе Приложение B. Физическая модель была разработана в приложении MySQL Workbench.

На рисунке 3.1 изображена физическая модель БД.

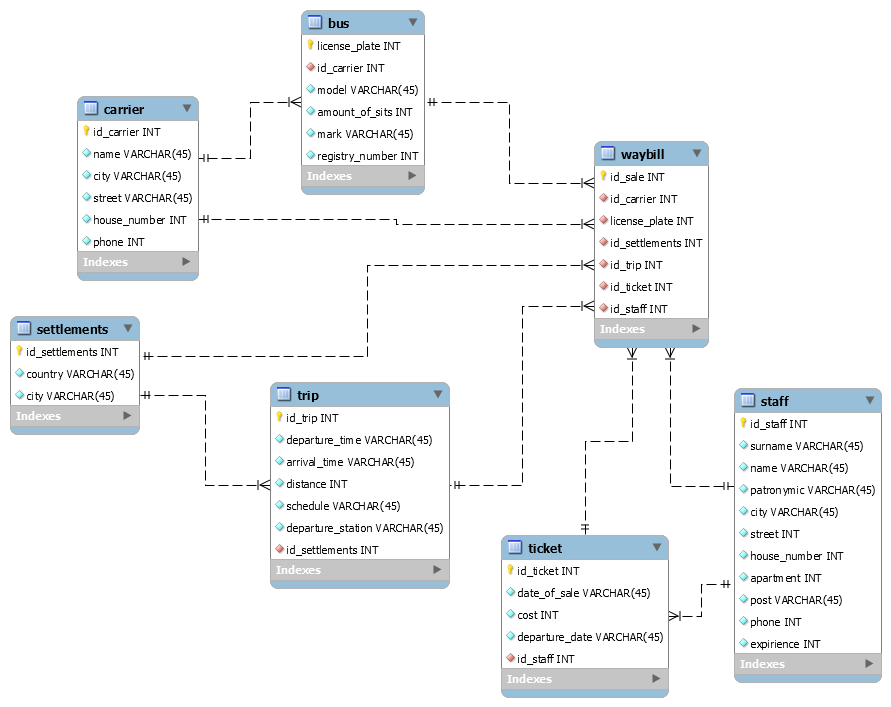


Рисунок 3.1 – Cкриншот представления разработанной базы данных

3.3 Тестирование базы данных

3.3.1 Заполнение таблиц

На рисунке 3.2 изображен результат заполнение таблицы «Перевозчик».

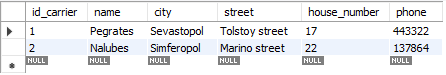


Рисунок 3.2 – Заполнение таблицы «Перевозчик»

На рисунке 3.3 изображен результат заполнение таблицы «Автобус».

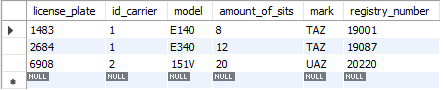


Рисунок 3.3 – Заполнение таблицы «Автобус»

На рисунке 3.4 изображен результат заполнение таблицы «Сотрудники»

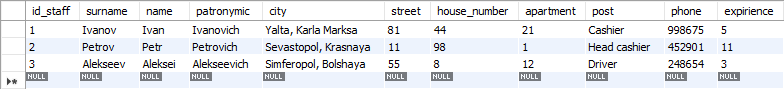


Рисунок 3.4 – Заполнение таблицы «Сотрудники»

На рисунке 3.5 изображен результат заполнение таблицы «Рейс»

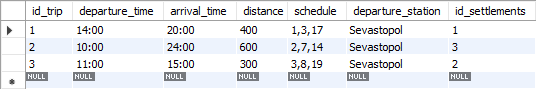


Рисунок 3.5 – Заполнение таблицы «Рейс»

На рисунке 3.6 изображен результат заполнение таблицы «Билет»

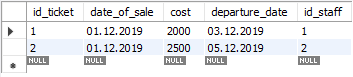


Рисунок 3.6 – Заполнение таблицы «Билет»

На рисунке 3.7 изображен результат заполнение таблицы «Населенные пункты».

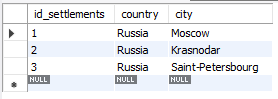


Рисунок 3.7 – Заполнение таблицы «Населенные пункты»

На рисунке 3.8 изображен результат заполнение таблицы «Путевой лист».

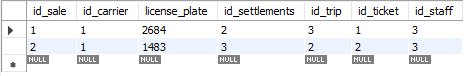


Рисунок 3.8 – Заполнение таблицы «Путевой лист»

3.3.2 Запросы для тестирования БД

Выберем перевозчиков из города «Sevastopol» (рисунок 3.9).

SELECT id\_carrier, name

FROM bus\_station.carrier

WHERE city = "Sevastopol";



Рисунок 3.9 – Результат запроса на выборку клиентов перевозчиков из города «Sevastopol»

Выберем количество автобусов у каждого перевозчика (рисунок 3.10).

SELECT bus\_station.carrier.name, COUNT(bus\_station.bus.license\_plate)

FROM bus\_station.carrier, bus\_station.bus

WHERE bus\_station.carrier.id\_carrier = bus\_station.bus.id\_carrier

GROUP BY bus\_station.carrier.name;



Рисунок 3.10 – Результат запроса на выборку количества автобусов у каждого перевозчика

Выберем населенные пункты, которых нет ни в 1 путевом листе c (рисунок 3.11).

SELECT \* FROM bus\_station.settlements

WHERE bus\_station.settlements.city NOT IN (SELECT city FROM bus\_station.settlements, bus\_station.waybill WHERE bus\_station.settlements.id\_settlements = bus\_station.waybill.id\_settlements);

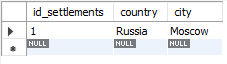


Рисунок 3.11 – Результат запроса на выборку населенных пунктов

Выберем сколько билетов куплено на автобус с самым наименьшим количеством мест (рисунок 3.12).

SELECT COUNT(bus\_station.waybill.id\_ticket)

FROM bus\_station.waybill

WHERE bus\_station.waybill.license\_plate IN (SELECT license\_plate FROM bus\_station.bus WHERE bus\_station.bus.amount\_of\_sits IN (SELECT MIN(bus\_station.bus.amount\_of\_sits) FROM bus\_station.bus));

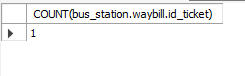


Рисунок 3.12 – Результат запроса на выборку количества билетов

Выберем рейсы водителей со стажем меньше 5 лет (рисунок 3.13).

SELECT bus\_station.staff.surname, bus\_station.staff.name, bus\_station.trip.id\_trip

FROM bus\_station.trip, bus\_station.waybill, bus\_station.staff

WHERE bus\_station.trip.id\_trip = bus\_station.waybill.id\_trip

AND bus\_station.waybill.id\_staff = bus\_station.staff.id\_staff

AND bus\_station.staff.post = "Driver"

AND bus\_station.staff.expirience < 5;

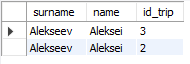


Рисунок 3.13 – Результат запроса на выборку рейсов водителей со стажем меньше 5 лет

Выберем путевые листы, где населенный пункт начинается на букву К (рисунок 3.14).

SELECT bus\_station.waybill.id\_sale

FROM bus\_station.waybill, bus\_station.settlements

WHERE bus\_station.waybill.id\_settlements = bus\_station.settlements.id\_settlements

AND bus\_station.settlements.city LIKE "K%";



Рисунок 3.14 – Результат запроса на выборку путевых листов, где населенный пункт начинается на букву К

3.3.2 Проверка целостности

Проверим целостность БД на основе отношений «Населенные пункты», «Рейс», «Путевой лист».

Удалим одну запись в таблице «Населенные пункты», в результате чего в таблицах «Рейс» и «Путевой лсит» удалятся все записи, кортежи которых ссылаются на удаленного владельца.

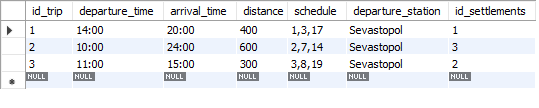


Рисунок 3.15 – Таблица «Рейс» до удаления

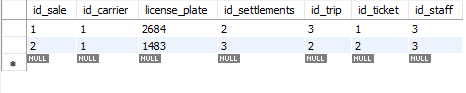


Рисунок 3.16 – Таблица «Путевой лист» до удаления

DELETE FROM bus\_station.settlements

WHERE city = "Krasnodar";

На рисунках 3.17 и 3.18 видно, что поля были удалены, так как внешние ключи, таблиц «Рейс» и «Путевой лист», имеют действие при удалении CASCADE.

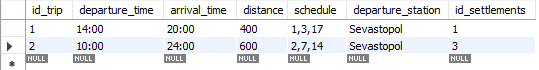


Рисунок 3.17 – Таблица «Рейс» после удаления владельца



Рисунок 3.18 – Таблица «Путевой лист» после удаления владельца

3.4 Разграничение прав доступа

Данная БД создана для работников автовокзала,следовательно в ее права входит: создание, удаление, редактирование полей всех таблиц и запрет на удаление таблиц и базы данных.

3.5 Расчет информационных параметров базы данных

Для переменных длин полей возьмем максимальную длину поля, следовательно, размер типов данных для хранения в памяти будет составлять:

1. integer – 2 байта

2. Varchar(x) – x байт

Были расчитаны длины логических записей:

Таблица «bus»:

Таблица «carrier»:

Таблица «settlements»:

Таблица «staff»:

Таблица «ticket»:

Таблица «trip»:

Таблица «waybill»:

Предположим, что в таблицах «bus» 10 записей (L1 = 98), «carrier» 10 записей (L2 = 141), «settlements » 10 постояльцев (L3 = 92), «staff » 10 записей (L4 = 237), «ticket » 10 записей (L5 = 96), «trip» 10 записей (L6 = 186), «waybill» 10 записей (L7 = 14).

Рассчитаем объем памяти, необходимой для размещения информационного фонда:

Приращение информационного фонда для интенсивности добавления записей в 0.01 [1/байт2]: ΔI = (1/байт)

Время резервного копирования всей структуры занимает:

3.2 (секунды)

В разделе 3, была выбрана программная платформа, на которой будет риализована БД. После чего следовала реализация физической модели БД, а далее ее заполнение. Были проведены тесты БД, проверка её целостноти, ограничений и работы простых запросов на вставку, обновление и удаление записей. Так же в БД реализовано разграничение прав доступа на менеджера и администратора. Окончательным этапом реализации физической модели БД был расчет информационных параметров. Они были рассчитаны с допущениями, что БД включает полную наполненность всех полей атрибутов.

4 Разработка клиентского приложения

4.1 Обоснование выбора языка программирования

Программная структура организована с помощью форм – именно в них происходит вся обработка информации.

В качестве языка программирования был выбран Java: язык программирования общего назначения, который следует парадигме объектно-ориентированного программирования. Его плюсами является объектно-ориентированное программирование, высокоуровневость программирования с простым синтаксисом, кроссплатформенность.

Для связи с БД был выбран JDBC API - платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94), реализованный в виде пакета java.sql, входящего в состав [Java SE](https://ru.wikipedia.org/wiki/Java_SE).

4.2 Разработка интерфейса пользователя

Программная структура организована с помощью форм – именно в них происходит вся обработка информации. Полный листинг представлен в разделе «Приложение Б»

На рисунке 4.1 изображено гланое окно «Автосервис», которое содержит таблицу отображающию заказы, а также кнопки «Добавить» для добовления записи, «Изменить» для изменения записи, «Удалить» для удаления записи и «Статистика» бля просмотра статистики автосервиса.

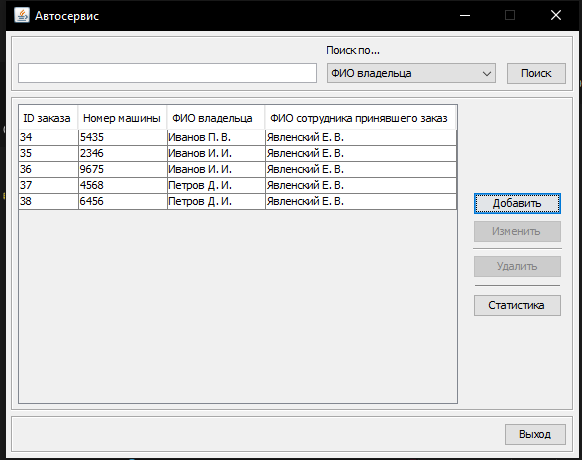


Рисунок 4.1 – Главное окно «Автосервис»

На рисунке 4.2 изображено окно для добовления зака, в нем можно выбрать владельца, после чего его машину, имеется выбор запчастей и услуг, а также в конце указывается менеджер, принявший заказ. Также дааное окно имее таки кнопки как «Добавть владельца» и «Добавить машину», по нажатию на которые откроются диалоговые окна для добавления владельца и машины соответствина.

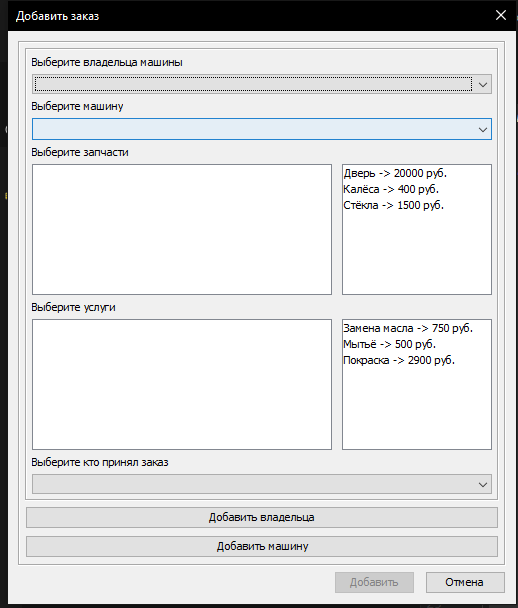


Рисунок 4.2 – Диалоговое окно «Добавить заказ»

На рисунке 4.3 изображено диалоговое окно для добовления владельца, в котором нужно ввести номер паспорта владкльца, его ФИО, дисконтную карту, а также номер телефона.

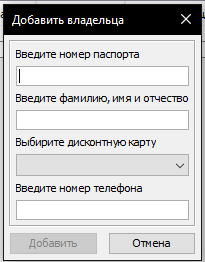


Рисунок 4.3 – Диалоговое окно «Добавить владельца»

На рисунке 4.4 изображено диалоговое окно для добовления машины, в котором можно выбрать владельца машины, а также ввести её номер, марку, цвет и дату выпуска.

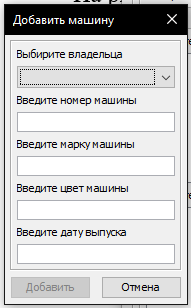


Рисунок 4.4 – Диалоговое окно «Добавить машины»

На рисунке 4.5 изображено диалоговоле окно, которое появляется при добавлении услуги или запчасти, в нем можно указать кто будет выполнять услугу или замену запчасти

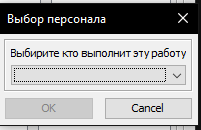


Рисунок 4.5 – Диалоговое окно «Выбор персонала»

На рисунке 4.6 изображено диалоговоле окно, которое позволяет изменить запчасти и услуги в заказе.

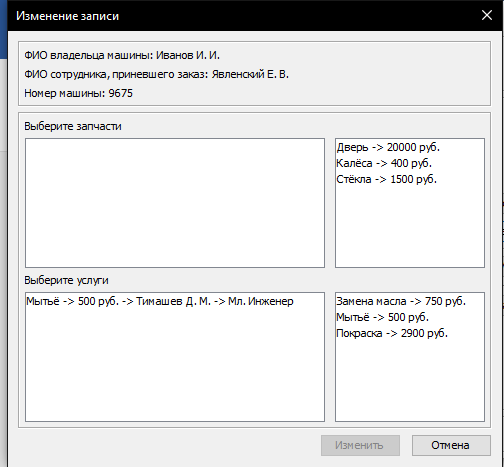


Рисунок 4.6 – Диалоговое окно «Изменение записи»

На рисунке 4.7 изображено диалоговоле окно, которое показывает статистику автосервиса, а именно количество заказов, машин и владельцов, а также количество замен запчастей, замену которых должен выполнять персонал, услуг, которые выполнит персонал, а также сколько принял заказов каждый менеджер.

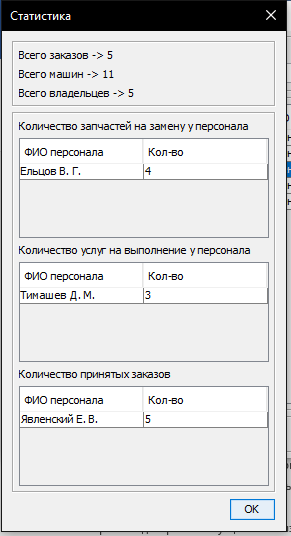


Рисунок 4.7 – Диалоговое окно «Статистика»

4.3 Алгоритм работы каждого из модулей

Самым главным модулем является DB.java, так как он связывается с БД и предоставляет возможность остальным частям программы производить манипуляции с БД.

Далее будут перечислены SQL запросы которые использует приложения для своей работы, стоит уделить внимание тому что для более гибкого использования запросов к ним было применено форматирование строк (метод format класса String стандартной библиотеки языка Java), а именно значения, которые могут меняться были заменены на последовательность символов «%s», в место них будут подставлены необходимые приложению значения, после чего готовый SQL запрос будет отправлен СУБД на обработку.

DB.java использует следующие SQL запросы:

Удаление записи из отношения «Заказ»

DELETE FROM `order` WHERE `id` = %s

Удаление записи из отношения «Заказ\_Услуги»

DELETE FROM `order\_services` WHERE `id\_order` = %s

Удаление записи из отношения «Заказ\_Запчасти»

DELETE FROM `order\_parts` WHERE `id\_order` = %s

Вставка новой записи в отношение «Машина»

INSERT INTO `car`(`num`, `num\_passport\_owner`, `color`, `realeseDate`, `mark`) VALUES ('%s', '%s', '%s', '%s', '%s')

Вставка новой записи в отношение «Владелец»

INSERT INTO `owner`(`num\_passport`, `id\_discount\_card`, `surname`, `name`, `middle\_name`, `num\_telephone`) VALUES ('%s', '%s', '%s', '%s', '%s', '%s')

Вставка новой записи в отношение «Заказ»

INSERT INTO `order`(`num\_car`, `num\_passport\_owner`, `id\_staff`) VALUES ('%s', '%s', '%s')

Выборка из отношения «Заказ» по номеру машины

SELECT \* FROM `order` WHERE `num\_car` = %s

Выборка из отношения «Машина»

SELECT \* FROM `car` WHERE num\_passport\_owner = %s

Выборка из отношения «Запчасть»

SELECT \* FROM `part` WHERE `name` = '%s'

Выборка из отношения «Услуга»

SELECT \* FROM `service` WHERE `name` = '%s'

Выборка из отношения «машина» по номеру паспорта владельца

SELECT \* FROM `car` WHERE num\_passport\_owner = %s

Выборка из отношения «Заказ»

SELECT \* FROM `order`

Выборка из отношения «Владелец»

SELECT \* FROM `owner`

Выборка из отношения «Персонал» по должности

SELECT \* FROM `staff` WHERE position = %s

Выборка из отношения «Запчасть»

SELECT \* FROM `part`

Выборка количества машин

SELECT COUNT(`num`) as count FROM `car`

Выборка количества владельцев

SELECT COUNT(`num\_passport`) as count FROM `owner`

Выборка количества заказов

SELECT COUNT(`id`) as count FROM `order`

Выборка из отношения «Заказ\_Запчасти» по номеру заказа

SELECT \* FROM `order\_parts` WHERE `id\_order` = %s

Выборка из отношения «Заказ\_Услуги» по номеру заказа

SELECT \* FROM `order\_services` WHERE `id\_order` = %s

Выборка из отношения «услуги» по номеру заказа

SELECT \* FROM `service`

Выборка из отношения «Дисконтная катрта» по номеру заказа

SELECT \* FROM `discount\_сard`

Выборка запчастей, которых нет в заказе, из отношения «Запчасть»

SELECT \* FROM `part` as p WHERE p.`name` NOT IN (SELECT `name` FROM `part` JOIN `order\_parts` ON `order\_parts`.`name\_part` = `part`.`name` AND `order\_parts`.`id\_order` = %s)

Выборка запчауслугстей, которых нет в заказе, из отношения «Услуга»

SELECT \* FROM `service` as s WHERE s.`name` NOT IN (SELECT `name`FROM `service` JOIN `order\_services` ON `order\_services`.`name\_service` = `service`.`name` AND `order\_services`.`id\_order` = %s)

Выборка из отношений «Персонал», «Заказ\_Запсасти»

SELECT `second\_name`, `name`, `middle\_name`, COUNT(`order\_parts`.`id\_order`) as count FROM `staff` JOIN `order\_parts` ON `staff`.`id` = `order\_parts`.`id\_performing` GROUP BY `second\_name`

Выборка из отношений «Персонал», «Заказ\_Услуги»

SELECT `second\_name`, `name`, `middle\_name`, COUNT(`order\_services`.`id\_order`) as count FROM `staff` JOIN `order\_services` ON `staff`.`id` = `order\_services`.`id\_performing` GROUP BY `second\_name`

Выборка суммы закзов у персонала

SELECT `second\_name`, `name`, `middle\_name`, COUNT(`order`.`id`) as count FROM `staff` JOIN `order` ON `staff`.`id` = `order`.`id\_staff` GROUP BY `second\_name`

4.4 Тестирование работы приложения

На рисунке 4.8 изображена таблица которая хронит в себе все заказы

Нажмем на кноку «Добавть», после чего откроется диалоговое окно «Добавить зака» в котором нажмем на кнопку «Добавить владельца», заполним поля и подтвердим добовление владельца.

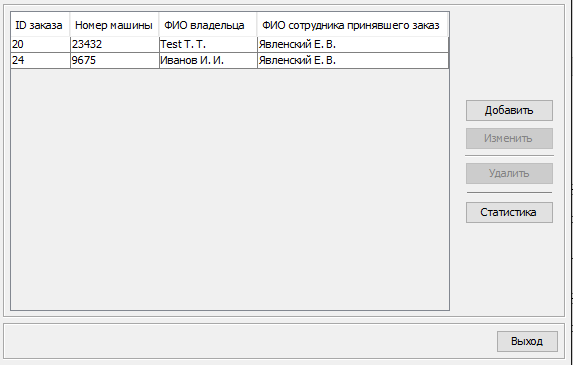


Рисунок 4.8 – Таблица на главном окне «Автомастерская»

На рисунке 4.9 изображен результат заполнение полей диалогового окна

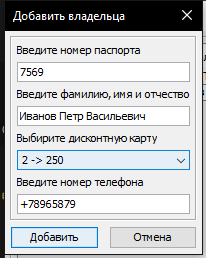


Рисунок 4.9 – Диалоговое окно «Добавление владельца»

Далее нажмем на кнопку «Добавить машину», заполним поля и подтвердим добовление машиный.

На рисунке 4.10 изображен результат заполнение полей диалогового окна

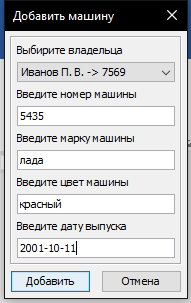


Рисунок 4.10 – Диалоговое окно «Добавление машину»

Теперь вабкрим добавленного владельца и его машину, а также выберим услуги, запчасти и того, кто принемал заказ.

На рисунке 4.12 изображен результат заполнение полей диалогового окна

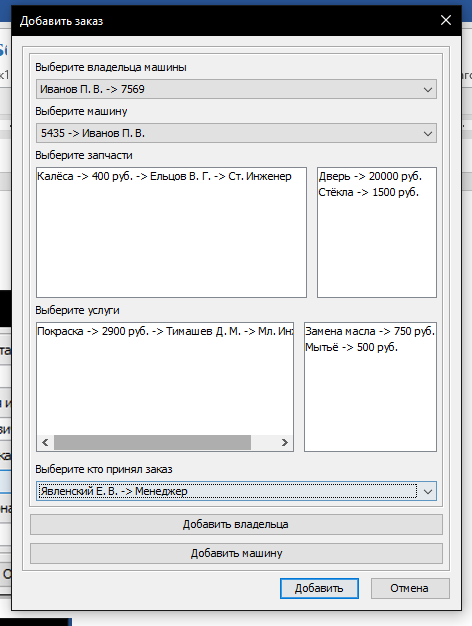


Рисунок 4.12 – Диалоговое окно «Добавить заказ»

На рисунке 4.13 изображен результат добаление заказа

Теперь изменим заказ боавив в него все доступные запчасти и подтвердим доьавление.

На рисунке 4.13 изображен результат добавления запчастей

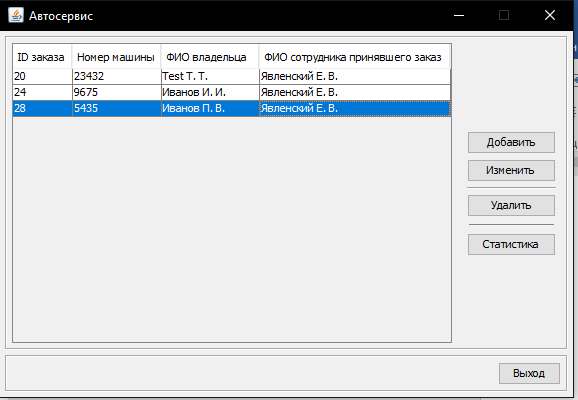


Рисунок 4.13 – Таблица на главном окне «Автомастерская» после добавления заказа

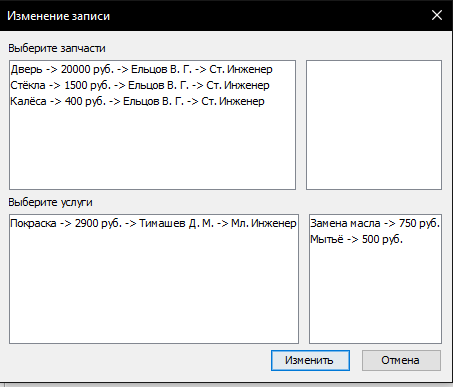


Рисунок 4.14 – Диалогое окно «Изменение записи»

Далее удалим добавленный ранее заказ, для этого выберим его и нажмем на ктнопку «Удалить»

На рисунке 4.15 изображен результат удаления заказа

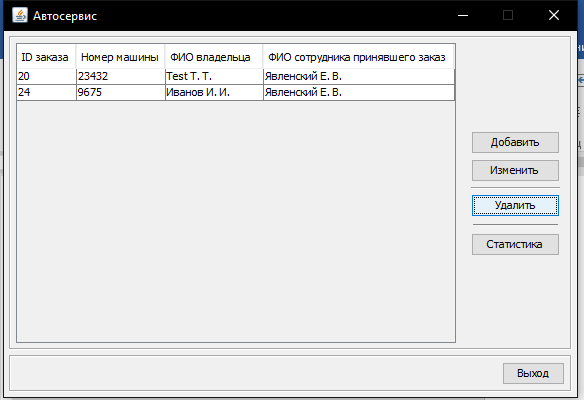


Рисунок 4.15 – Таблица на главном окне «Автомастерская» после удаления заказа

Для удобства работы с базой данных было реализованно приложения, осуществляющий взаимодействие с ней. Самым первым этапом в разработке приложения былы выбор основного языка программирования, а также программной среды. Затем была реализована клиентская часть и структура будущего приложения. Следующим этапом стало написание приложения для взаимодействия с базой данных. В разделе 4 было описана реализация клиентского приложения на платформе Java, с помощью её библиотеки JDBC, алгоритм работы приложения, а также были проведены тесты, проверяющие правильность его работы, а также описание основных модулей.

ЗАКЛЮЧЕНИЕ

В данном курсовом проекте была спроектирована реляционная база данных на тему «База данных автомастерской».

Для выбранной предметной области был проведен анализ, построена логическая модель и проведена нормализация, которая позволяет решить проблемы рационального выбора вариантов схем отношений из возможного множества альтернативных решений. В полученной в результате нормализации логической схеме содержатся следующие таблицы: владелец, машина, дисконтная карта, заказ, заказ\_услуга, заказ\_запчасть, запчасть, заказ, персонал, должность.

Была разработана база данных, эмулирующая предметную область «Автосервис» предназначенная для починки. К базе данных в качестве интерфейса было создано приложение для менеджера.

В процессе выполнения были достигнуты следующие задачи:

- описана предметная область;

- выбраны информационные технологии, подходящих для разработки информационной системы;

- разработаны логическая и физическая модель базы данных;

- сделано физическое проектирование базы, данных в выбранной СУБД;

- разработано программное обеспечение, реализующее и формы для базы данных;

- проведено тестирование базы данных и приложения;

- рассчитаны информационные параметры БД.

Выполнение работы включало в себя множество различных этапов и шагов. Сначала была получена задача на курсовое проектирование. Далее было проведено подробное исследование предметной области. На основе полученных данных была спроектирована логическая модель данных, содержащая в себе перечень сущностей предметной области, перечень свойств этих сущностей и связей между ними. Для оптимизации выполнения запросов была проведена нормализация структуры логической модели. Затем на основе проекта была реализована физическая модель данных в СУБД phpMyAdmin. Реализация включала в себя подробное тестирование ограничений целостности базы данных и различные запросы реляционной алгебры.

Для демонстрации работы существующей базы данных был реализован программный продукт, осуществляющий взаимодействие с ней. Создание этого продукта также включало в себя множество этапов. Сначала были выбраны основные языки программирования и разметки, а также программные среды. Затем была реализована клиентская часть и структура будущего приложения с учетом разграничения прав доступа. Следующим этапом стало написание приложения для взаимодействия с базой данных.

Для работы с базой был выбран язык Java, а для подключения к БД библиотека JDBC.

По результатам тестирования разработанного продукта, можно сделать вывод, что задача, поставленная на курсовое проектирование, была выполнена в полном объеме.

СПИСОК ИСПОЛЬЗОВАННЫХ СОКРАЩЕНИЙ

ИС - информационная система.

БД - база данных.

1НФ – первая нормальная форма.

2НФ – вторая нормальная форма.

3НФ – третья нормальная форма.

НФБК – норбальная форма Бойса-Кодда.

БД – база банных.

ERD — Entity-Relationship Diagrams (диаграмма «сущность-связь»)

JDBC API — (Java DataBase Connectivity) соединение с базами данных на Java

[Java SE](https://ru.wikipedia.org/wiki/Java_SE) —  Java Platform Standard Edition (стандартная версия платформы Java)

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Java. Промышленное программирование: практ. пособие / И.Н. Блинов, В.С. Романчик. – Минск: УниверсалПресс, 2007. – 704 с

SQL: Пер. с англ. – М.: ДМК Пресс. – 456 с.: ил. (Серия «Quick Start»)

Swing: Эффектные пользовательские интерфейсы издание 2-ое: Издательство “Лори”, 2011. – 591 c

ПРИЛОЖЕНИЕ А  
Листинг запросов

-- -----------------------------------------------------

-- Schema Bus\_station

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `Bus\_station` DEFAULT CHARACTER SET utf8 ;

USE `Bus\_station` ;

-- -----------------------------------------------------

-- Table `Bus\_station`.`carrier`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Bus\_station`.`carrier` (

`id\_carrier` INT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(45) NOT NULL,

`city` VARCHAR(45) NOT NULL,

`street` VARCHAR(45) NOT NULL,

`house\_number` INT NOT NULL,

`phone` INT NOT NULL,

PRIMARY KEY (`id\_carrier`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `Bus\_station`.`bus`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Bus\_station`.`bus` (

`license\_plate` INT NOT NULL,

`id\_carrier` INT NOT NULL,

`model` VARCHAR(45) NOT NULL,

`amount\_of\_sits` INT NOT NULL,

`mark` VARCHAR(45) NOT NULL,

`registry\_number` INT NOT NULL,

PRIMARY KEY (`license\_plate`),

INDEX `carrier\_idx` (`id\_carrier` ASC) VISIBLE,

CONSTRAINT `carrier`

FOREIGN KEY (`id\_carrier`)

REFERENCES `Bus\_station`.`carrier` (`id\_carrier`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `Bus\_station`.`settlements`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Bus\_station`.`settlements` (

`id\_settlements` INT NOT NULL AUTO\_INCREMENT,

`country` VARCHAR(45) NOT NULL,

`city` VARCHAR(45) NOT NULL,

PRIMARY KEY (`id\_settlements`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `Bus\_station`.`trip`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Bus\_station`.`trip` (

`id\_trip` INT NOT NULL AUTO\_INCREMENT,

`departure\_time` VARCHAR(45) NOT NULL,

`arrival\_time` VARCHAR(45) NOT NULL,

`distance` INT NOT NULL,

`schedule` VARCHAR(45) NOT NULL,

`departure\_station` VARCHAR(45) NOT NULL,

`id\_settlements` INT NOT NULL,

PRIMARY KEY (`id\_trip`),

INDEX `settlements\_idx` (`id\_settlements` ASC) VISIBLE,

CONSTRAINT `settlements`

FOREIGN KEY (`id\_settlements`)

REFERENCES `Bus\_station`.`settlements` (`id\_settlements`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `Bus\_station`.`staff`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Bus\_station`.`staff` (

`id\_staff` INT NOT NULL,

`surname` VARCHAR(45) NOT NULL,

`name` VARCHAR(45) NOT NULL,

`patronymic` VARCHAR(45) NOT NULL,

`city` VARCHAR(45) NOT NULL,

`street` INT NOT NULL,

`house\_number` INT NOT NULL,

`apartment` INT NOT NULL,

`post` VARCHAR(45) NOT NULL,

`phone` INT NOT NULL,

`expirience` INT NOT NULL,

PRIMARY KEY (`id\_staff`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `Bus\_station`.`ticket`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Bus\_station`.`ticket` (

`id\_ticket` INT NOT NULL AUTO\_INCREMENT,

`date\_of\_sale` VARCHAR(45) NOT NULL,

`cost` INT NOT NULL,

`departure\_date` VARCHAR(45) NOT NULL,

`id\_staff` INT NOT NULL,

PRIMARY KEY (`id\_ticket`),

INDEX `staff\_idx` (`id\_staff` ASC) VISIBLE,

CONSTRAINT `staff`

FOREIGN KEY (`id\_staff`)

REFERENCES `Bus\_station`.`staff` (`id\_staff`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `Bus\_station`.`waybill`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `Bus\_station`.`waybill` (

`id\_sale` INT NOT NULL AUTO\_INCREMENT,

`id\_carrier` INT NOT NULL,

`license\_plate` INT NOT NULL,

`id\_settlements` INT NOT NULL,

`id\_trip` INT NOT NULL,

`id\_ticket` INT NOT NULL,

`id\_staff` INT NOT NULL,

PRIMARY KEY (`id\_sale`),

INDEX `carrier\_idx` (`id\_carrier` ASC) VISIBLE,

INDEX `plate\_idx` (`license\_plate` ASC) VISIBLE,

INDEX `settlements\_idx` (`id\_settlements` ASC) VISIBLE,

INDEX `trip\_idx` (`id\_trip` ASC) VISIBLE,

INDEX `ticket\_idx` (`id\_ticket` ASC) VISIBLE,

INDEX `staff\_idx` (`id\_staff` ASC) VISIBLE,

CONSTRAINT `wb\_carrier`

FOREIGN KEY (`id\_carrier`)

REFERENCES `Bus\_station`.`carrier` (`id\_carrier`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `wb\_plate`

FOREIGN KEY (`license\_plate`)

REFERENCES `Bus\_station`.`bus` (`license\_plate`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `wb\_settlements`

FOREIGN KEY (`id\_settlements`)

REFERENCES `Bus\_station`.`settlements` (`id\_settlements`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `wb\_trip`

FOREIGN KEY (`id\_trip`)

REFERENCES `Bus\_station`.`trip` (`id\_trip`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `wb\_ticket`

FOREIGN KEY (`id\_ticket`)

REFERENCES `Bus\_station`.`ticket` (`id\_ticket`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `wb\_staff`

FOREIGN KEY (`id\_staff`)

REFERENCES `Bus\_station`.`staff` (`id\_staff`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

ПРИЛОЖЕНИЕ Б  
Листинг программы

Main.java

import com.data.MainWindow;  
  
import javax.swing.\*;  
  
public class Main {  
 public static void main(String[] args) throws ClassNotFoundException, UnsupportedLookAndFeelException, InstantiationException, IllegalAccessException {  
 UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());  
 MainWindow window = new MainWindow();  
  
 window.setVisible(true);  
 }  
}

MainWindow.java

package com.data;  
  
import com.data.Tables.FIO;  
import com.data.Tables.Order;  
import com.data.Tables.Owner;  
  
import javax.swing.\*;  
import javax.swing.event.ListSelectionEvent;  
import javax.swing.event.ListSelectionListener;  
  
import java.awt.event.\*;  
  
public class MainWindow extends JFrame {  
 private JPanel panelContent;  
 private JButton buttonQuit;  
 private JButton buttonAdd;  
 private JButton buttonChange;  
 private JTable table;  
 private JButton buttonDelete;  
 private JButton buttonStatistic;  
 private JTextField textFieldFind;  
 private JButton buttonFind;  
 private JComboBox <ForFind> comboBoxFind;  
 private TableModel tableModel;  
 private DB db;  
  
 public MainWindow() {  
 initUI();  
  
 db = new DB();  
 tableModel = new TableModel(db);  
  
 table.setModel(tableModel);  
 table.getAutoCreateColumnsFromModel();  
  
 int[] columnsWidth = {60, 90, 100, 200};  
 for (int i = 0; i < tableModel.getColumnCount(); i++) {  
 table.getColumn(table.getColumnName(i)).setMaxWidth(columnsWidth[i]);  
 }  
  
 comboBoxFind.setModel(new DefaultComboBoxModel<>(new ForFind[] {  
 new ForFind("ФИО владельца", new FindAction() {  
 @Override  
 public Owner find() throws NotFoundInDBException {  
 return db.getOwnerOnFIO(new FIO(textFieldFind.getText(), " "));  
 }  
 }, this),  
  
 new ForFind("Номеру паспорта владельца", new FindAction() {  
 @Override  
 public Owner find() throws NotFoundInDBException {  
 return db.getOwner("num\_telephone", textFieldFind.getText());  
 }  
 }, this),  
  
 new ForFind("Номеру машины", new FindAction() {  
 @Override  
 public Owner find() throws NotFoundInDBException {  
 return db.getOwnerOnNumCar(textFieldFind.getText());  
 }  
 }, this)  
 }));  
  
 updateTable();  
 }  
  
 private void initUI() {  
 setContentPane(panelContent);  
 pack();  
 setResizable(false);  
 setDefaultCloseOperation(DISPOSE\_ON\_CLOSE);  
 setTitle("Автосервис");  
  
 addEvent();  
 }  
  
 private void addEvent() {  
 buttonQuit.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 onButtonQuit();  
 }  
 });  
  
 buttonAdd.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 onButtonAddRecord();  
 }  
 });  
  
 buttonDelete.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 onButtonDelete();  
 }  
 });  
  
 buttonChange.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 onButtonChange();  
 }  
 });  
  
 buttonStatistic.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 onButtonStatistic();  
 }  
 });  
  
 table.getSelectionModel().addListSelectionListener(new ListSelectionListener(){  
 public void valueChanged(ListSelectionEvent event) {  
 if (tableModel.getRowCount() > 0 && table.getSelectedRow() != -1) {  
 buttonDelete.setEnabled(true);  
 buttonChange.setEnabled(true);  
 } else {  
 buttonDelete.setEnabled(false);  
 buttonChange.setEnabled(false);  
 }  
 }  
 });  
  
 table.addMouseListener(new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 if (e.getClickCount() == 2) {  
 onButtonChange();  
 }  
 }  
 });  
  
 buttonFind.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 onButtonFind();  
 }  
 });  
  
 textFieldFind.addFocusListener(new FocusAdapter() {  
 @Override  
 public void focusGained(FocusEvent e) {  
 getRootPane().setDefaultButton(buttonFind);  
 }  
  
 @Override  
 public void focusLost(FocusEvent e) {  
 getRootPane().setDefaultButton(buttonChange);  
 }  
 });  
 }  
  
 private void onButtonQuit() {  
 db.close();  
 dispose();  
 }  
  
 private void onButtonAddRecord() {  
 AddOrderWindow dialog = new AddOrderWindow(this);  
 dialog.addComponentToUpdate(new Updater() {  
 @Override  
 public void update() {  
 updateTable();  
 }  
 });  
 dialog.setVisible(true);  
 }  
  
 private void onButtonDelete() {  
 try {  
 db.delete(tableModel.getRow(table.getSelectedRow()));  
 } catch (IndexOutOfBoundsException ex) {  
 JOptionPane.showMessageDialog(this, "Выбирите заказ");  
 }  
  
 updateTable();  
 }  
  
 private void onButtonChange() {  
 try {  
 ChangeRecordWindow dialog = new ChangeRecordWindow(this);  
 dialog.pack();  
 dialog.setVisible(true);  
 } catch (IndexOutOfBoundsException ex) {  
 JOptionPane.showMessageDialog(this, "Выбирите заказ");  
 }  
  
 }  
  
 private void onButtonFind () {  
 try {  
 ((ForFind)comboBoxFind.getSelectedItem()).start();  
 } catch (NotFoundInDBException ex) {  
 JOptionPane.showMessageDialog(this, ex.getMessage());  
 } catch (IndexOutOfBoundsException ex) {  
 JOptionPane.showMessageDialog(this, "Не верное ФИО");  
 }  
 }  
  
 private void onButtonStatistic () {  
 StatisticsWindow dialog = new StatisticsWindow(this);  
 dialog.setVisible(true);  
 }  
  
 public void updateTable() {  
 tableModel.updateTable();  
 if (tableModel.getRowCount() > 0 && table.getSelectedRow() != -1) {  
 buttonDelete.setEnabled(true);  
 buttonChange.setEnabled(true);  
 } else {  
 buttonDelete.setEnabled(false);  
 buttonChange.setEnabled(false);  
 }  
  
 }  
  
 public Order getSelectedOrder () {  
 return tableModel.getRow(table.getSelectedRow());  
 }  
  
 public DB getDb() {  
 return db;  
 }  
}  
  
class ForFind {  
  
 private final String name;  
 private final FindAction action;  
 private final MainWindow root;  
  
 ForFind(String name, FindAction action, MainWindow root) {  
 this.name = name;  
 this.action = action;  
 this.root = root;  
 }  
  
 public void start() throws NotFoundInDBException {  
 OwnerInfWindow dialog = new OwnerInfWindow(  
 root,  
 action  
 );  
 dialog.setVisible(true);  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 @Override  
 public String toString() {  
 return name;  
 }  
}

AddOrderWindow.java

package com.data;  
  
import com.data.Tables.\*;  
  
import javax.swing.\*;  
import javax.swing.event.ListSelectionEvent;  
import javax.swing.event.ListSelectionListener;  
import java.awt.event.\*;  
import java.util.ArrayList;  
  
  
public class AddOrderWindow extends JDialog {  
 private JPanel contentPane;  
 private JButton buttonAdd;  
 private JButton buttonCancel;  
 private JComboBox comboBoxOwner;  
 private JComboBox comboBoxCar;  
 private JButton buttonAddOwner;  
 private JButton buttonAddCar;  
 private JList listAddPart;  
 private JList listPart;  
 private JList listAddService;  
 private JList listService;  
 private JComboBox comboBoxStaff;  
 private MainWindow parent;  
  
 private DefaultComboBoxModel <Owner> comboBoxOwnerModel = new DefaultComboBoxModel<>();  
 private DefaultComboBoxModel <Car> comboBoxCarModel = new DefaultComboBoxModel<>();  
 private DefaultListModel <Part> listPartModel = new DefaultListModel<>();  
 private DefaultListModel <Service> listServiceModel = new DefaultListModel<>();  
 private DefaultListModel <Order\_Parts> listAddPartModel = new DefaultListModel<>();  
 private DefaultListModel <Order\_Services> listAddServiceModel = new DefaultListModel<>();  
 private DefaultComboBoxModel <Staff> comboBoxStaffModel = new DefaultComboBoxModel<>();  
  
 private final ArrayList <Updater> componentsForUpdate = new ArrayList<>();  
  
 public AddOrderWindow(MainWindow parent) {  
 super(parent, "Добавить заказ", true);  
  
 initUi();  
 this.parent = parent;  
  
 comboBoxOwner.setModel(comboBoxOwnerModel);  
 comboBoxCar.setModel(comboBoxCarModel);  
 comboBoxStaff.setModel(comboBoxStaffModel);  
 listPart.setModel(listPartModel);  
 listService.setModel(listServiceModel);  
 listAddPart.setModel(listAddPartModel);  
 listAddService.setModel(listAddServiceModel);  
  
  
 listService.setValueIsAdjusting(false);  
  
 updateComboBoxOwnerModel();  
 updateListPartModel();  
 updateListServiceModel();  
 updateComboBoxStaffModel("'Менеджер'");  
 }  
  
 public AddOrderWindow(MainWindow parent, Owner owner) {  
 super(parent, "Добавить заказ", true);  
  
 initUi();  
 this.parent = parent;  
  
 comboBoxOwner.setModel(comboBoxOwnerModel);  
 comboBoxCar.setModel(comboBoxCarModel);  
 comboBoxStaff.setModel(comboBoxStaffModel);  
 listPart.setModel(listPartModel);  
 listService.setModel(listServiceModel);  
 listAddPart.setModel(listAddPartModel);  
 listAddService.setModel(listAddServiceModel);  
  
  
 listService.setValueIsAdjusting(false);  
  
 comboBoxOwnerModel.addElement(owner);  
 comboBoxOwnerModel.setSelectedItem(owner);  
 comboBoxOwner.setEnabled(false);  
  
 buttonAddOwner.setEnabled(false);  
  
 updateListPartModel();  
 updateListServiceModel();  
 updateComboBoxStaffModel("'Менеджер'");  
 }  
  
 private void initUi(){  
 setContentPane(contentPane);  
 getRootPane().setDefaultButton(buttonAdd);  
 setDefaultCloseOperation(DISPOSE\_ON\_CLOSE);  
 setResizable(false);  
 pack();  
  
 addEvent();  
 }  
  
 private void addEvent(){  
 buttonCancel.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 dispose();  
 }  
 });  
  
 buttonAddOwner.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 onButtonAddOwner();  
 }  
 });  
  
 buttonAddCar.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 onButtonAddCar();  
 }  
 });  
  
 comboBoxOwner.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 Owner owner = (Owner) comboBoxOwnerModel.getSelectedItem();  
 if (owner != null) updateComboBoxCarModel(owner);  
 }  
 });  
  
 buttonAdd.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 onButtonAdd();  
 }  
 });  
  
 comboBoxOwner.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 checkAllowAdd();  
 }  
 });  
  
 comboBoxCar.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 checkAllowAdd();  
 }  
 });  
  
 comboBoxStaff.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 checkAllowAdd();  
 }  
 });  
  
 listAddPart.addMouseListener(new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) { if (e.getClickCount() == 2) onDeletePart(); }  
 });  
  
 listPart.addMouseListener(new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) { if (e.getClickCount() == 2) onAddParts(); }  
 });  
  
 listAddService.addMouseListener(new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) { if (e.getClickCount() == 2) onDeleteServices(); }  
 });  
  
 listService.addMouseListener(new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) { if (e.getClickCount() == 2) onAddServices(); }  
 });  
 }  
  
 public void updateComboBoxOwnerModel() {  
 comboBoxOwnerModel.removeAllElements();  
 comboBoxOwnerModel.addAll(parent.getDb().getOwners());  
 }  
  
 public void updateComboBoxCarModel(Owner owner) {  
 comboBoxCarModel.removeAllElements();  
 comboBoxCarModel.addAll(parent.getDb().getCars(owner));  
 }  
  
 public void updateComboBoxStaffModel(String position){  
 comboBoxStaffModel.addAll(parent.getDb().getStaffs(position));  
 }  
  
 private void updateListPartModel() {  
 listPartModel.addAll(parent.getDb().getParts());  
 }  
  
 private void updateListServiceModel() {  
 listServiceModel.addAll(parent.getDb().getServices());  
 }  
  
 private void onButtonAddCar(){  
 AddCarWindow dialog = new AddCarWindow(this);  
 dialog.setVisible(true);  
  
 checkAllowAdd();  
 }  
  
 private void onButtonAddOwner(){  
 AddOwnerWindow dialog = new AddOwnerWindow(this);  
 dialog.pack();  
 dialog.setVisible(true);  
  
 checkAllowAdd();  
 }  
  
 private void onButtonAdd(){  
 Owner owner = (Owner) comboBoxOwner.getSelectedItem();  
 Car car = (Car) comboBoxCar.getSelectedItem();  
 Staff staff = (Staff) comboBoxStaff.getSelectedItem();  
 Order order = new Order(  
 "",  
 car,  
 owner,  
 staff  
 );  
  
 String idOrder = String.valueOf(parent.getDb().insert(order));  
 order.setId(idOrder);  
  
 for (Object part: listAddPartModel.toArray()) {  
 Order\_Parts partToAdd = (Order\_Parts) part;  
 partToAdd.setOrder(order);  
 parent.getDb().insert(partToAdd);  
 }  
 for (Object service: listAddServiceModel.toArray()) {  
 Order\_Services serviceToAdd = (Order\_Services) service;  
 serviceToAdd.setOrder(order);  
 parent.getDb().insert(serviceToAdd);  
 }  
  
 parent.updateTable();  
  
 updateComponents();  
  
 dispose();  
 }  
  
 public void onAddParts(){  
 int index = listPart.getSelectedIndex();  
  
 if (index == -1) return;  
  
 AddJobWindow dialog = new AddJobWindow(parent, "'Ст. Инженер'");  
 dialog.pack();  
 dialog.setVisible(true);  
  
 if (dialog.getSelectedStaff() == null) return;  
  
 listAddPartModel.add(0, new Order\_Parts(listPartModel.get(index), dialog.getSelectedStaff()));  
  
 checkAllowAdd();  
 }  
  
 public void onAddServices(){  
 int index = listService.getSelectedIndex();  
  
 if (index == -1) return;  
  
 AddJobWindow dialog = new AddJobWindow(parent, "'Мл. Инженер'");  
 dialog.pack();  
 dialog.setVisible(true);  
 if (dialog.getSelectedStaff() == null) return;  
  
 listAddServiceModel.add(0, new Order\_Services(listServiceModel.get(index), dialog.getSelectedStaff()));  
  
 checkAllowAdd();  
 }  
  
 public void onDeletePart(){  
 int index = listAddPart.getSelectedIndex();  
  
 if (index == -1) return;  
  
 listPartModel.add(0, (Part)listAddPartModel.get(index).getJob());  
 listAddPartModel.remove(index);  
  
 checkAllowAdd();  
 }  
  
 public void onDeleteServices(){  
 int index = listAddService.getSelectedIndex();  
  
 if (index == -1) return;  
  
 listServiceModel.add(0, (Service)listAddServiceModel.get(index).getJob());  
 listAddServiceModel.remove(index);  
  
 checkAllowAdd();  
 }  
  
 private void checkAllowAdd () {  
 if (comboBoxOwner.getSelectedIndex() == -1 ||  
 comboBoxCar.getSelectedIndex() == -1 ||  
 (listAddPartModel.getSize() == 0 &&  
 listAddServiceModel.getSize() == 0) ||  
 comboBoxStaff.getSelectedIndex() == -1) {  
 buttonAdd.setEnabled(false);  
 } else {  
 buttonAdd.setEnabled(true);  
 }  
 }  
  
 @Override  
 public MainWindow getParent() {  
 return parent;  
 }  
  
 private void updateComponents () {  
 for (Updater updater: componentsForUpdate) {  
 updater.update();  
 }  
 }  
  
 public void addComponentToUpdate(Updater updater) {  
 componentsForUpdate.add(updater);  
 }  
}

StatisticWindow.java

package com.data;  
  
import javax.swing.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
  
public class StatisticsWindow extends JDialog {  
 private JPanel contentPane;  
 private JButton buttonOK;  
 private JTable tableCountParts;  
 private JTable tableCountServices;  
 private JTable tableStaffTakedOrders;  
 private JLabel labelCountOrders;  
 private JLabel labelCountOwners;  
 private JLabel labelCountCars;  
  
 private final MainWindow parent;  
 private final TableCountModel tableCountPartsModel;  
 private final TableCountModel tableCountServicesModel;  
 private final TableCountModel tableStaffTakedOrdersModel;  
  
 public StatisticsWindow(MainWindow parent) {  
 initUi();  
  
 this.parent = parent;  
  
 tableCountPartsModel = new TableCountModel(parent.getDb());  
 tableCountServicesModel = new TableCountModel(parent.getDb());  
 tableStaffTakedOrdersModel = new TableCountModel(parent.getDb());  
  
 tableStaffTakedOrdersModel.setColumnHead(new String[] {"ФИО персонала", "Кол-во"});  
 tableCountPartsModel.setColumnHead(new String[] {"ФИО персонала", "Кол-во"});  
 tableCountServicesModel.setColumnHead(new String[] {"ФИО персонала", "Кол-во"});  
  
 tableStaffTakedOrders.setModel(tableStaffTakedOrdersModel);  
 tableCountParts.setModel(tableCountPartsModel);  
 tableCountServices.setModel(tableCountServicesModel);  
  
 tableStaffTakedOrders.setAutoCreateColumnsFromModel(true);  
 tableCountParts.setAutoCreateColumnsFromModel(true);  
 tableCountServices.setAutoCreateColumnsFromModel(true);  
  
 tableStaffTakedOrdersModel.updateModel(parent.getDb().getStaffCountTakeOrder());  
 tableCountPartsModel.updateModel(parent.getDb().getStaffPartsCount());  
 tableCountServicesModel.updateModel(parent.getDb().getStaffServicesCount());  
  
 labelCountOrders.setText(String.format("%s -> %s", labelCountOrders.getText(), parent.getDb().getCountOrders()));  
 labelCountCars.setText(String.format("%s -> %s", labelCountCars.getText(), parent.getDb().getCountCars()));  
 labelCountOwners.setText(String.format("%s -> %s", labelCountOwners.getText(), parent.getDb().getCountOwners()));  
 }  
  
 private void initUi(){  
 setContentPane(contentPane);  
 setModal(true);  
 pack();  
 getRootPane().setDefaultButton(buttonOK);  
 setDefaultCloseOperation(DISPOSE\_ON\_CLOSE);  
 setResizable(false);  
 setTitle("Статистика");  
  
 addEvent();  
 }  
  
 private void addEvent(){  
 buttonOK.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 dispose();  
 }  
 });  
 }  
}

OwnerInfWindow.java

package com.data;  
  
import com.data.Tables.Car;  
import com.data.Tables.Order;  
import com.data.Tables.Owner;  
  
import javax.swing.\*;  
import java.awt.event.\*;  
  
public class OwnerInfWindow extends JDialog {  
 private JPanel contentPane;  
 private JButton buttonOK;  
 private JLabel labelFIOOwner;  
 private JLabel labelNumPassportOwner;  
 private JLabel labelNumPhoneOwner;  
 private JList <Car> listCars;  
 private JList <Order> listOrders;  
 private JButton buttonAddCar;  
 private JButton buttonAddOrder;  
 private JButton buttonDeleteOrder;  
  
 private final MainWindow parent;  
 private final Owner owner;  
 private final DefaultListModel <Car> listCarsModel = new DefaultListModel<>();  
 private final DefaultListModel <Order> listOrdersModel = new DefaultListModel<>();  
 private final DB db;  
  
 public OwnerInfWindow(MainWindow parent, FindAction finder) throws NotFoundInDBException {  
 super(parent, "Инормация о владельце", true);  
 initUI();  
  
 this.parent = parent;  
 owner = finder.find();  
 db = parent.getDb();  
  
 labelFIOOwner.setText("ФИО владельца: " + owner.getFio());  
 labelNumPassportOwner.setText("Номер паспорта владельца: " + owner.getNum\_passport());  
 labelNumPhoneOwner.setText("Номер телефона владельца: " + owner.getNum\_telephone());  
  
 listCars.setModel(listCarsModel);  
 listOrders.setModel(listOrdersModel);  
  
 updateListCarsModel();  
 updateListOrdersModel();  
 }  
  
 private void initUI () {  
 setContentPane(contentPane);  
 pack();  
 getRootPane().setDefaultButton(buttonOK);  
 setDefaultCloseOperation(DISPOSE\_ON\_CLOSE);  
 setResizable(false);  
  
 addEvent();  
 }  
  
 private void addEvent () {  
 buttonOK.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 dispose();  
 }  
 });  
  
 buttonAddCar.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 onButtonAddCar();  
 }  
 });  
  
 buttonAddOrder.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) { onButtonAddOrder(); }  
 });  
 buttonDeleteOrder.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 onButtonDeleteOrder();  
 }  
 });  
 }  
  
 private void onButtonAddCar () {  
 AddCarWindow dialog = new AddCarWindow(parent, owner);  
 dialog.setVisible(true);  
  
 owner.addCar(dialog.getCreatedCar());  
 updateListCarsModel();  
 }  
  
 private void updateListCarsModel () {  
 listCarsModel.removeAllElements();  
 listCarsModel.addAll(owner.getCars());  
 }  
  
 private void updateListOrdersModel () {  
 listOrdersModel.removeAllElements();  
 listOrdersModel.addAll(parent.getDb().getOrders(owner));  
 }  
  
 private void onButtonAddOrder () {  
 AddOrderWindow dialog = new AddOrderWindow(parent, owner);  
 dialog.addComponentToUpdate(new Updater() {  
 @Override  
 public void update() {  
 updateListOrdersModel();  
 }  
 });  
 dialog.setVisible(true);  
 }  
  
 private void onButtonDeleteOrder () {  
 try {  
 db.delete(listOrders.getSelectedValue());  
 updateListOrdersModel();  
 } catch (NullPointerException ex) {  
 JOptionPane.showMessageDialog(this, "Выбирите заказ");  
 }  
 }  
  
 public DB getDb() {  
 return db;  
 }  
}

AddOwnerWindow.java

package com.data;  
  
import com.data.Tables.DiscountCard;  
import com.data.Tables.FIO;  
import com.data.Tables.Owner;  
  
import javax.swing.\*;  
import javax.swing.event.DocumentEvent;  
import javax.swing.event.DocumentListener;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class AddOwnerWindow extends JDialog {  
 private JPanel contentPane;  
 private JButton buttonAdd;  
 private JButton buttonCancel;  
 private JTextField textFieldNumPassport;  
 private JFormattedTextField formattedTextFieldFIO;  
 private JComboBox comboBoxDiscountCard;  
 private JFormattedTextField formattedTextFieldNumPhone;  
  
 private DefaultComboBoxModel <DiscountCard> comboBoxDiscountCardModel = new DefaultComboBoxModel<>();  
 private AddOrderWindow parent;  
 private DB db;  
  
 public AddOwnerWindow(AddOrderWindow parent) {  
 initUI();  
  
 this.parent = parent;  
 db = parent.getParent().getDb();  
  
 comboBoxDiscountCard.setModel(comboBoxDiscountCardModel);  
 updateComboBoxDiscountCard();  
 }  
  
 private void initUI(){  
 setContentPane(contentPane);  
 setModal(true);  
 getRootPane().setDefaultButton(buttonAdd);  
 setResizable(false);  
 setDefaultCloseOperation(DISPOSE\_ON\_CLOSE);  
 setTitle("Добавить владельца");  
  
 addEvent();  
 }  
  
 private void addEvent(){  
 buttonAdd.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 onButtonAdd();  
 }  
 });  
  
 buttonCancel.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 dispose();  
 }  
 });  
  
 textFieldNumPassport.getDocument().addDocumentListener(new DocumentListener() {  
 @Override  
 public void insertUpdate(DocumentEvent e) { checkAllowAdd(); }  
 @Override  
 public void removeUpdate(DocumentEvent e) { }  
 @Override  
 public void changedUpdate(DocumentEvent e) { }  
 });  
 formattedTextFieldFIO.getDocument().addDocumentListener(new DocumentListener() {  
 @Override  
 public void insertUpdate(DocumentEvent e) { checkAllowAdd(); }  
  
 @Override  
 public void removeUpdate(DocumentEvent e) { }  
  
 @Override  
 public void changedUpdate(DocumentEvent e) { }  
 });  
 formattedTextFieldNumPhone.getDocument().addDocumentListener(new DocumentListener() {  
 @Override  
 public void insertUpdate(DocumentEvent e) { checkAllowAdd(); }  
  
 @Override  
 public void removeUpdate(DocumentEvent e) { }  
  
 @Override  
 public void changedUpdate(DocumentEvent e) { }  
 });  
 comboBoxDiscountCard.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 checkAllowAdd();  
 }  
 });  
 }  
  
 public static void main(String[] args) throws ClassNotFoundException, UnsupportedLookAndFeelException, InstantiationException, IllegalAccessException {  
 UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());  
 AddOwnerWindow dialog = new AddOwnerWindow(null);  
 dialog.pack();  
 dialog.setVisible(true);  
 }  
  
 private void onButtonAdd () {  
 if (!checkAllowAdd()) {  
 JOptionPane.showMessageDialog(this, "Заполните все поля");  
 return;  
 }  
  
 FIO fio = new FIO(formattedTextFieldFIO.getText(), " ");  
 Owner owner = new Owner(  
 textFieldNumPassport.getText(),  
 ((DiscountCard)comboBoxDiscountCard.getSelectedItem()).getId(),  
 fio.getSurname(),  
 fio.getName(),  
 fio.getMiddleName(),  
 formattedTextFieldNumPhone.getText().replace("+", ""));  
 db.insert(owner);  
 parent.updateComboBoxOwnerModel();  
 dispose();  
 }  
  
 private boolean checkAllowAdd () {  
 if (textFieldNumPassport.getText().equals("") ||  
 textFieldNumPassport.getText().equals("") ||  
 comboBoxDiscountCard.getSelectedIndex() == -1 ||  
 formattedTextFieldNumPhone.getText().equals("")) {  
 buttonAdd.setEnabled(false);  
 return false;  
 } else {  
 buttonAdd.setEnabled(true);  
 return true;  
 }  
 }  
  
 private void updateComboBoxDiscountCard(){  
 comboBoxDiscountCardModel.addAll(db.getDiscountCards());  
 }  
}

AddCarWindow.java

package com.data;  
  
import com.data.Tables.Car;  
import com.data.Tables.Owner;  
  
import javax.swing.\*;  
import javax.swing.event.DocumentEvent;  
import javax.swing.event.DocumentListener;  
import java.awt.event.\*;  
  
public class AddCarWindow extends JDialog {  
 private JPanel contentPane;  
 private JButton buttonAdd;  
 private JButton buttonCancel;  
 private JComboBox <Owner> comboBoxOwner;  
 private JTextField textFieldNum;  
 private JTextField textFieldMark;  
 private JTextField textFieldColor;  
 private JFormattedTextField formattedTextFieldDateRelease;  
  
 private AddOrderWindow parent;  
 private DB db;  
 private DefaultComboBoxModel <Owner> comboBoxOwnerModel = new DefaultComboBoxModel<>();  
 private Car createdCar;  
  
 public AddCarWindow(AddOrderWindow parent) {  
 initUI();  
  
 this.parent = parent;  
 db = parent.getParent().getDb();  
 comboBoxOwner.setModel(comboBoxOwnerModel);  
 updateComboBoxOwnerModel();  
 }  
  
 public AddCarWindow(MainWindow parent, Owner owner) {  
 initUI();  
  
 db = parent.getDb();  
 comboBoxOwner.setModel(comboBoxOwnerModel);  
 comboBoxOwnerModel.addElement(owner);  
 comboBoxOwner.setSelectedItem(owner);  
 comboBoxOwner.setEnabled(false);  
 }  
  
 private void initUI(){  
 setContentPane(contentPane);  
 pack();  
 setModal(true);  
 setDefaultCloseOperation(DISPOSE\_ON\_CLOSE);  
 getRootPane().setDefaultButton(buttonAdd);  
 setResizable(false);  
 setTitle("Добавить машину");  
  
 addEvent();  
 }  
  
 private void addEvent(){  
 buttonCancel.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 onButtonCancel();  
 }  
 });  
  
 buttonAdd.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 onButtonAdd();  
 }  
 });  
  
 comboBoxOwner.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
  
 }  
 });  
 textFieldNum.getDocument().addDocumentListener(new DocumentListener() {  
 @Override  
 public void insertUpdate(DocumentEvent e) { checkAllowAdd(); }  
  
 @Override  
 public void removeUpdate(DocumentEvent e) { }  
  
 @Override  
 public void changedUpdate(DocumentEvent e) { }  
 });  
 textFieldMark.getDocument().addDocumentListener(new DocumentListener() {  
 @Override  
 public void insertUpdate(DocumentEvent e) { checkAllowAdd(); }  
  
 @Override  
 public void removeUpdate(DocumentEvent e) { }  
  
 @Override  
 public void changedUpdate(DocumentEvent e) { }  
 });  
 textFieldColor.getDocument().addDocumentListener(new DocumentListener() {  
 @Override  
 public void insertUpdate(DocumentEvent e) { checkAllowAdd(); }  
  
 @Override  
 public void removeUpdate(DocumentEvent e) { }  
  
 @Override  
 public void changedUpdate(DocumentEvent e) { }  
 });  
 formattedTextFieldDateRelease.getDocument().addDocumentListener(new DocumentListener() {  
 @Override  
 public void insertUpdate(DocumentEvent e) { checkAllowAdd(); }  
  
 @Override  
 public void removeUpdate(DocumentEvent e) { }  
  
 @Override  
 public void changedUpdate(DocumentEvent e) { }  
 });  
 }  
  
 private boolean checkAllowAdd () {  
 if (comboBoxOwner.getSelectedIndex() == -1 ||  
 textFieldNum.getText().equals("") ||  
 textFieldMark.getText().equals("") ||  
 textFieldColor.getText().equals("") ||  
 formattedTextFieldDateRelease.getText().equals("")) {  
 buttonAdd.setEnabled(false);  
 return false;  
 } else {  
 buttonAdd.setEnabled(true);  
 return true;  
 }  
 }  
  
 private void updateComboBoxOwnerModel () {  
 comboBoxOwnerModel.addAll(db.getOwners());  
 }  
  
 private void onButtonCancel (){  
 dispose();  
 }  
  
 private void onButtonAdd () {  
 if (!checkAllowAdd()) {  
 JOptionPane.showMessageDialog(this, "Заполните все поля");  
 return;  
 }  
  
 Car car = new Car(  
 textFieldNum.getText(),  
 (Owner) comboBoxOwner.getSelectedItem(),  
 textFieldColor.getText(),  
 formattedTextFieldDateRelease.getText(),  
 textFieldMark.getText()  
 );  
 db.insert(car);  
  
 createdCar = car;  
 dispose();  
 }  
  
 public Car getCreatedCar() {  
 return createdCar;  
 }  
}

AddJobWindow.java

package com.data;  
  
import com.data.Tables.Staff;  
  
import javax.swing.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class AddJobWindow extends JDialog {  
 private JPanel contentPane;  
 private JButton buttonAdd;  
 private JButton buttonCancel;  
 private JComboBox comboBoxStaff;  
  
 private DefaultComboBoxModel <Staff> comboBoxStaffModel = new DefaultComboBoxModel<>();  
 private MainWindow parent;  
  
 public AddJobWindow(MainWindow parent, String staffPosition) {  
 initUi();  
  
 this.parent = parent;  
  
 comboBoxStaff.setModel(comboBoxStaffModel);  
 updateComboBoxStaffModel(staffPosition);  
 }  
  
 private void initUi() {  
 setContentPane(contentPane);  
 setModal(true);  
 getRootPane().setDefaultButton(buttonAdd);  
 setResizable(false);  
 setTitle("Выбор персонала");  
  
 addEvent();  
 }  
  
 private void addEvent() {  
 buttonAdd.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 dispose();  
 }  
 });  
  
 buttonCancel.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 dispose();  
 }  
 });  
  
 comboBoxStaff.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 buttonAdd.setEnabled(true);  
 }  
 });  
 }  
  
 public Staff getSelectedStaff(){  
 return (Staff)comboBoxStaffModel.getSelectedItem();  
 }  
  
 private void updateComboBoxStaffModel(String staffPosition) {  
 comboBoxStaffModel.addAll(parent.getDb().getStaffs(staffPosition));  
 }  
}

ChangeRecordWindow.java

package com.data;  
  
import com.data.Tables.\*;  
  
import javax.swing.\*;  
import javax.swing.event.ListSelectionEvent;  
import javax.swing.event.ListSelectionListener;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class ChangeRecordWindow extends JDialog {  
 private JPanel contentPane;  
 private JButton buttonChange;  
 private JButton buttonCancel;  
 private JList listAddPart;  
 private JList listPart;  
 private JList listAddService;  
 private JList listService;  
 private JLabel labelOwnerFio;  
 private JLabel labelStaffFio;  
 private JLabel labelCarNum;  
  
 private final Order order;  
 private final MainWindow parent;  
  
 private final DefaultListModel <Part> listPartModel = new DefaultListModel<>();  
 private final DefaultListModel <Service> listServiceModel = new DefaultListModel<>();  
 private final DefaultListModel <Order\_Parts> listAddPartModel = new DefaultListModel<>();  
 private final DefaultListModel <Order\_Services> listAddServiceModel = new DefaultListModel<>();  
  
 public ChangeRecordWindow (MainWindow parent) {  
 super(parent, "Изменение записи", true);  
 initUI();  
  
 this.order = parent.getSelectedOrder();  
 this.parent = parent;  
  
 listPart.setModel(listPartModel);  
 listService.setModel(listServiceModel);  
 listAddPart.setModel(listAddPartModel);  
 listAddService.setModel(listAddServiceModel);  
  
 updateListAddPartModel();  
 updateListAddServiceModel();  
 updateListPart();  
 updateListService();  
  
 updateLabels();  
 }  
  
 private void initUI () {  
 setContentPane(contentPane);  
 setDefaultCloseOperation(DISPOSE\_ON\_CLOSE);  
 setResizable(false);  
 getRootPane().setDefaultButton(buttonChange);  
  
 addEvent();  
 }  
  
 private void addEvent () {  
 buttonCancel.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 onButtonCancel();  
 }  
 });  
 buttonChange.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 onButtonChange();  
 }  
 });  
 listService.addListSelectionListener( new ListSelectionListener() {  
 public void valueChanged(ListSelectionEvent e) {  
 onAddServices();  
 }  
 });  
  
 listAddService.addListSelectionListener( new ListSelectionListener() {  
 public void valueChanged(ListSelectionEvent e) {  
 onDeleteServices();  
 }  
 });  
  
 listPart.addListSelectionListener( new ListSelectionListener() {  
 public void valueChanged(ListSelectionEvent e) {  
 onAddParts();  
 }  
 });  
  
 listAddPart.addListSelectionListener( new ListSelectionListener() {  
 public void valueChanged(ListSelectionEvent e) {  
 onDeletePart();  
 }  
 });  
 }  
  
 private void onButtonCancel () {  
 dispose();  
 }  
  
 private void onButtonChange () {  
  
 parent.getDb().deleteAllServicesAndPartsInOrder(order);  
  
 for (Object part: listAddPartModel.toArray()) {  
 Order\_Parts partToAdd = (Order\_Parts) part;  
 partToAdd.setOrder(order);  
 parent.getDb().insert(partToAdd);  
 }  
 for (Object service: listAddServiceModel.toArray()) {  
 Order\_Services serviceToAdd = (Order\_Services) service;  
 serviceToAdd.setOrder(order);  
 parent.getDb().insert(serviceToAdd);  
 }  
  
 parent.updateTable();  
 dispose();  
 }  
  
 private void updateListPart () {  
 listPartModel.addAll(parent.getDb().getParts());  
 }  
  
 private void updateListService () {  
 listServiceModel.addAll(parent.getDb().getServices());  
 }  
  
 private void updateListAddPartModel () {  
 listAddPartModel.addAll(parent.getDb().getOrderParts(parent.getSelectedOrder()));  
 }  
  
 private void updateListAddServiceModel () {  
 listAddServiceModel.addAll(parent.getDb().getOrderServices(parent.getSelectedOrder()));  
 }  
  
 private void updateLabels () {  
 labelOwnerFio.setText("ФИО владельца машины: " + order.getOwner().getFio().toString());  
 labelStaffFio.setText("ФИО сотрудника, приневшего заказ: " + order.getStaff().getFio().toString());  
 labelCarNum.setText("Номер машины: " + order.getNum\_car());  
 }  
  
 private void onAddParts () {  
 int index = listPart.getSelectedIndex();  
  
 if (index == -1) return;  
  
 AddJobWindow dialog = new AddJobWindow(parent,"'Ст. Инженер'");  
 dialog.pack();  
 dialog.setVisible(true);  
  
 if (dialog.getSelectedStaff() == null) return;  
  
 listAddPartModel.add(0, new Order\_Parts(listPartModel.get(index), dialog.getSelectedStaff()));  
  
 buttonChange.setEnabled(true);  
 }  
  
 private void onAddServices () {  
 int index = listService.getSelectedIndex();  
  
 if (index == -1) return;  
  
 AddJobWindow dialog = new AddJobWindow(parent,"'Мл. Инженер'");  
 dialog.pack();  
 dialog.setVisible(true);  
  
 if (dialog.getSelectedStaff() == null) return;  
  
 listAddServiceModel.add(0, new Order\_Services(listServiceModel.get(index), dialog.getSelectedStaff()));  
  
 buttonChange.setEnabled(true);  
 }  
  
 private void onDeletePart () {  
 int index = listAddPart.getSelectedIndex();  
  
 if (index == -1) return;  
  
 listPartModel.add(0, (Part)listAddPartModel.get(index).getJob());  
 listAddPartModel.remove(index);  
  
 checkAllowChange();  
 }  
  
 private void onDeleteServices () {  
 int index = listAddService.getSelectedIndex();  
  
 if (index == -1) return;  
  
 listServiceModel.add(0, (Service)listAddServiceModel.get(index).getJob());  
 listAddServiceModel.remove(index);  
  
 checkAllowChange();  
 }  
  
 private void checkAllowChange () {  
 if (listAddPartModel.size() == 0 && listServiceModel.size() == 0) {  
 buttonChange.setEnabled(false);  
 JOptionPane.showMessageDialog(this, "Должна быть выбрана хотя бы одна запчать или услуга");  
 } else {  
 buttonChange.setEnabled(true);  
 }  
 }  
}

DB.java

package com.data;  
  
import com.data.Tables.\*;  
  
import java.sql.\*;  
import java.util.ArrayList;  
  
public class DB {  
 private Connection conn;  
 private Statement statement;  
  
 public DB () {  
 try  
 {  
 Class.forName ("com.mysql.cj.jdbc.Driver");  
 String userName = "root";  
 String password = "";  
 String url = "jdbc:mysql://localhost:3306/coursework"+  
 "?verifyServerCertificate=false"+  
 "&useSSL=false"+  
 "&requireSSL=false"+  
 "&useLegacyDatetimeCode=false"+  
 "&amp"+  
 "&serverTimezone=UTC";  
 conn = DriverManager.getConnection (url, userName, password);  
 statement = conn.createStatement();  
 }  
 catch (Exception ex) {  
 ex.printStackTrace();  
 }  
 }  
  
 public ResultSet select (String query) throws SQLException {  
 return statement.executeQuery(query);  
 }  
  
 public void close () {  
 try {  
 statement.close();  
 conn.close ();  
 } catch (SQLException ex){  
 ex.printStackTrace();  
 }  
  
 }  
  
 public void delete (Order order) {  
 try {  
 statement.execute(String.format("DELETE FROM `order` WHERE `id` = %s", order.getId()));  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
  
 public void deleteAllServicesAndPartsInOrder (Order order) {  
 try {  
 statement.execute(String.format("DELETE FROM `order\_services` WHERE `id\_order` = %s", order.getId()));  
 statement.execute(String.format("DELETE FROM `order\_parts` WHERE `id\_order` = %s", order.getId()));  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
  
 public void insert (Car car) {  
 try {  
 statement.executeUpdate(String.format("INSERT INTO `car`(`num`, `num\_passport\_owner`, `color`, `realeseDate`, `mark`) VALUES ('%s', '%s', '%s', '%s', '%s')",  
 car.getNum(),  
 car.getNum\_passport\_owner(),  
 car.getColor(),  
 car.getRealeseDate(),  
 car.getMark()));  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
  
 public void insert (Owner owner) {  
 try {  
 statement.executeUpdate(String.format("INSERT INTO `owner`(`num\_passport`, `id\_discount\_card`, `surname`, `name`, `middle\_name`, `num\_telephone`) VALUES ('%s', '%s', '%s', '%s', '%s', '%s')",  
 owner.getNum\_passport(),  
 owner.getId\_discount\_card(),  
 owner.getFio().getSurname(),  
 owner.getFio().getName(),  
 owner.getFio().getMiddleName(),  
 owner.getNum\_telephone()));  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
  
 public int insert (Order order) {  
  
 try {  
 statement.executeUpdate(String.format("INSERT INTO `order`(`num\_car`, `num\_passport\_owner`, `id\_staff`) VALUES ('%s', '%s', '%s')",  
 order.getNum\_car(),  
 order.getNum\_passport\_owner(),  
 order.getId\_stuff()  
 ), Statement.RETURN\_GENERATED\_KEYS);  
 ResultSet id = statement.getGeneratedKeys();  
 if (id.next()) return id.getInt(1);  
  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 return 0;  
 }  
  
 public void insert (Order\_job o\_j) {  
 try {  
 statement.executeUpdate(o\_j.getInsertRequest());  
 } catch (SQLException ex) {  
 ex.printStackTrace();  
 }  
 }  
  
 public Staff getStaff (Order order) throws NotFoundInDBException {  
 return getStaffOnRequest(Staff.getRequest(order.getId\_stuff()));  
 }  
  
 public Staff getStaff (Order\_job job) throws NotFoundInDBException {  
 return getStaffOnRequest(Staff.getRequest(job.getId\_performing()));  
 }  
  
 private Staff getStaffOnRequest (String request) throws NotFoundInDBException {  
 Staff staff = null;  
 try {  
 ResultSet res = select(request);  
 if (res.next()) {  
 staff = createStaff(res);  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 if (staff == null) throw new NotFoundInDBException("Персонал не обнаружен");  
 return staff;  
 }  
  
 public Owner getOwner (Order order) throws NotFoundInDBException {  
 return getOwnerWithCars(Owner.getRequest(order.getNum\_passport\_owner()));  
 }  
  
 public Owner getOwnerOnFIO (FIO fio) throws NotFoundInDBException {  
 return getOwnerWithCars(String.format(  
 "SELECT \* FROM `owner`" +  
 "WHERE `surname` = '%s' AND " +  
 "`name` = '%s' AND " +  
 "`middle\_name` = '%s'", fio.getSurname(), fio.getName(), fio.getMiddleName()));  
 }  
  
 public Owner getOwnerOnNumCar (String num\_car) throws NotFoundInDBException {  
 return getOwnerWithCars(String.format(  
 "SELECT \* " +  
 "FROM `owner` JOIN `car` " +  
 "ON `car`.`num\_passport\_owner` = `owner`.`num\_passport` AND " +  
 "`car`.`num` = '%s'", num\_car));  
 }  
  
 public Owner getOwner (String columnName, String value) throws NotFoundInDBException {  
 return getOwnerWithCars(String.format("SELECT \* FROM `owner` WHERE `%s` = '%s'", columnName, value));  
 }  
  
 private Owner getOwnerWithCars (String request) throws NotFoundInDBException {  
 Owner owner = null;  
 try {  
 ResultSet res = select(request);  
 if (!res.isClosed() && res.next()){  
 owner = createOwner(res);  
 }  
 if (owner == null) throw new NotFoundInDBException("Владелец не обнаружен");  
 owner.setCars(getCars(owner));  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 return owner;  
 }  
  
 public Car getCar (Owner owner) throws NotFoundInDBException {  
 Car car = null;  
 try {  
 ResultSet res = select(String.format("SELECT \* FROM `car` WHERE num\_passport\_owner = %s", owner.getNum\_passport()));  
 if (!res.isClosed() && res.next()){  
 car = createCar(res);  
 car.setOwner(owner);  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 if (car == null) throw new NotFoundInDBException("Машина не обнаружена");  
  
 return car;  
 }  
  
 public Part getPart (Order\_Parts partInOrder) throws NotFoundInDBException {  
 Part part = null;  
 try {  
 ResultSet res = select(String.format("SELECT \* FROM `part` WHERE `name` = '%s'", partInOrder.getName\_service\_part()));  
 if (!res.isClosed() && res.next()){  
 part = createPart(res);  
  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 if (part == null) throw new NotFoundInDBException("Запчасть не обнаружена");  
  
 return part;  
 }  
  
 public Service getService(Order\_Services servicesInOrder) throws NotFoundInDBException {  
 Service service = null;  
 try {  
 ResultSet res = select(String.format("SELECT \* FROM `service` WHERE `name` = '%s'", servicesInOrder.getName\_service\_part()));  
 if (!res.isClosed() && res.next()){  
 service = createService(res);  
  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 if (service == null) throw new NotFoundInDBException("Услуга не обнаружена");  
  
 return service;  
 }  
  
 public ArrayList <Car> getCars (Owner owner) {  
 ArrayList <Car> cars = new ArrayList<>();  
 try {  
 ResultSet res = select(String.format("SELECT \* FROM `car` WHERE num\_passport\_owner = %s", owner.getNum\_passport()));  
 while (res.next()) {  
 Car car = createCar(res);  
 car.setOwner(owner);  
 cars.add(car);  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 return cars;  
 }  
  
 public ArrayList <Order> getOrders(Owner owner) {  
 return getOrders(String.format(  
 "SELECT \* FROM `order` WHERE `order`.`num\_passport\_owner` = '%s'",  
 owner.getNum\_passport()  
 ));  
 }  
  
 public ArrayList <Order> getOrders() {  
 return getOrders("SELECT \* FROM `order`");  
 }  
  
 private ArrayList <Order> getOrders (String request) {  
 ArrayList <Order> orders = new ArrayList<>();  
 try {  
 ResultSet res = select(request);  
 while (res.next()){  
 Order order = createOrder(res);  
 orders.add(order);  
 }  
 for (Order order: orders){  
 order.setStaff(getStaff(order));  
 order.setOwner(getOwner(order));  
 order.setCar(getCar(order.getOwner()));  
 }  
  
 } catch (SQLException | NotFoundInDBException e) {  
 e.printStackTrace();  
 }  
 return orders;  
 }  
  
 public ArrayList <Owner> getOwners () {  
 ArrayList <Owner> owners = new ArrayList<>();  
 try {  
 ResultSet res = select("SELECT \* FROM `owner`");  
 while (!res.isClosed() && res.next()){  
 owners.add(createOwner(res));  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return owners;  
 }  
  
 public ArrayList <Staff> getStaffs (String position) {  
 return getStaffsByRequest(String.format("SELECT \* FROM `staff` WHERE position = %s", position));  
 }  
  
 private ArrayList <Staff> getStaffsByRequest (String request) {  
 ArrayList <Staff> staffs = new ArrayList<>();  
 try {  
 ResultSet res = select(request);  
 while (!res.isClosed() && res.next()){  
 staffs.add(createStaff(res));  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return staffs;  
 }  
  
 public ArrayList <Part> getParts () {  
 ArrayList <Part> parts = new ArrayList<>();  
 try {  
 ResultSet res = select("SELECT \* FROM `part`");  
 while (!res.isClosed() && res.next()){  
 parts.add(createPart(res));  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return parts;  
 }  
  
 public ArrayList <Part> getPartsNotInOrder (Order order) {  
 ArrayList <Part> parts = new ArrayList<>();  
 try {  
 ResultSet res = select(String.format  
 ("SELECT \* " +  
 "FROM `part` as p " +  
 "WHERE p.`name` NOT IN (" +  
 "SELECT `name` " +  
 "FROM `part` JOIN `order\_parts` ON `order\_parts`.`name\_part` = `part`.`name` AND " +  
 "`order\_parts`.`id\_order` = %s)", order.getId()));  
 while (!res.isClosed() && res.next()){  
 parts.add(createPart(res));  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return parts;  
 }  
  
 public ArrayList <Service> getServicesNotInOrder (Order order) {  
 ArrayList <Service> services = new ArrayList<>();  
 try {  
 ResultSet res = select(String.format  
 ("SELECT \* " +  
 "FROM `service` as s " +  
 "WHERE s.`name` NOT IN (" +  
 "SELECT `name` " +  
 "FROM `service` JOIN `order\_services` ON `order\_services`.`name\_service` = `service`.`name` AND " +  
 "`order\_services`.`id\_order` = %s)", order.getId()));  
 while (!res.isClosed() && res.next()){  
 services.add(createService(res));  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return services;  
 }  
  
 public ArrayList <Service> getServices () {  
 ArrayList <Service> services = new ArrayList<>();  
 try {  
 ResultSet res = select("SELECT \* FROM `service`");  
 while (!res.isClosed() && res.next()){  
 services.add( new Service(  
 res.getString("name"),  
 res.getString("cost")  
 ));  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return services;  
 }  
  
 public ArrayList <DiscountCard> getDiscountCards () {  
 ArrayList <DiscountCard> cards = new ArrayList<>();  
 try {  
 ResultSet res = select("SELECT \* FROM `discount\_сard`");  
 while (!res.isClosed() && res.next()){  
 cards.add( new DiscountCard(  
 res.getString("ID"),  
 res.getString("discount")  
 ));  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return cards;  
 }  
  
 public ArrayList <StaffCountWork> getStaffPartsCount () {  
 ArrayList <StaffCountWork> count = new ArrayList<>();  
 try {  
 ResultSet res = select(  
 "SELECT `second\_name`, `name`, `middle\_name`, COUNT(`order\_parts`.`id\_order`) as count FROM `staff` JOIN `order\_parts` ON `staff`.`id` = `order\_parts`.`id\_performing` GROUP BY `second\_name`");  
 while (!res.isClosed() && res.next()){  
 count.add(createStaffCountWork(res));  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return count;  
 }  
  
 public ArrayList <StaffCountWork> getStaffServicesCount () {  
 ArrayList <StaffCountWork> count = new ArrayList<>();  
 try {  
 ResultSet res = select(  
 "SELECT `second\_name`, `name`, `middle\_name`, COUNT(`order\_services`.`id\_order`) as count FROM `staff` JOIN `order\_services` ON `staff`.`id` = `order\_services`.`id\_performing` GROUP BY `second\_name`");  
 while (!res.isClosed() && res.next()){  
 count.add(createStaffCountWork(res));  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return count;  
 }  
  
 public ArrayList <StaffCountWork> getStaffCountTakeOrder () {  
 ArrayList <StaffCountWork> count = new ArrayList<>();  
 try {  
 ResultSet res = select(  
 "SELECT `second\_name`, `name`, `middle\_name`, COUNT(`order`.`id`) as count FROM `staff` JOIN `order` ON `staff`.`id` = `order`.`id\_staff` GROUP BY `second\_name`");  
 while (!res.isClosed() && res.next()){  
 count.add(createStaffCountWork(res));  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return count;  
 }  
  
 public String getCountCars () {  
 return getCount("SELECT COUNT(`num`) as count FROM `car`");  
 }  
  
 public String getCountOwners () {  
 return getCount("SELECT COUNT(`num\_passport`) as count FROM `owner`");  
 }  
  
 public String getCountOrders () {  
 return getCount("SELECT COUNT(`id`) as count FROM `order`");  
 }  
  
 private String getCount (String query) {  
 String count = null;  
 try {  
 ResultSet res = select(query);  
 if (res.next()){  
 count = res.getString("count");  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return count;  
 }  
  
 public ArrayList <Order\_Parts> getOrderParts (Order order) {  
 ArrayList <Order\_Parts> partsInOrder = new ArrayList<>();  
 try {  
 ResultSet res = select(String.format("SELECT \* FROM `order\_parts` WHERE `id\_order` = %s", order.getId()));  
 while (res.next()){  
 partsInOrder.add( new Order\_Parts(  
 order,  
 res.getString("name\_part"),  
 res.getString("id\_performing")  
 ));  
 }  
 for (Order\_Parts order\_parts: partsInOrder) {  
 order\_parts.setJob(getPart(order\_parts));  
 order\_parts.setPerforming(getStaff(order\_parts));  
 }  
 } catch (SQLException | NotFoundInDBException e) {  
 e.printStackTrace();  
 }  
 return partsInOrder;  
 }  
  
 public ArrayList <Order\_Services> getOrderServices(Order order) {  
 ArrayList <Order\_Services> servicesInOrder = new ArrayList<>();  
 try {  
 ResultSet res = select(String.format("SELECT \* FROM `order\_services` WHERE `id\_order` = %s", order.getId()));  
 while (res.next()){  
 servicesInOrder.add( new Order\_Services(  
 order,  
 res.getString("name\_service"),  
 res.getString("id\_performing")  
 ));  
 }  
 for (Order\_Services order\_services: servicesInOrder) {  
 order\_services.setJob(getService(order\_services));  
 order\_services.setPerforming(getStaff(order\_services));  
 }  
 } catch (SQLException | NotFoundInDBException e) {  
 e.printStackTrace();  
 }  
 return servicesInOrder;  
 }  
  
 private Order createOrder (ResultSet res) throws SQLException {  
 return new Order(  
 res.getString("id"),  
 res.getString("num\_car"),  
 res.getString("num\_passport\_owner"),  
 res.getString("id\_staff")  
 );  
 }  
  
 private Car createCar (ResultSet res) throws SQLException {  
 return new Car(  
 res.getString("num"),  
 res.getString("num\_passport\_owner"),  
 res.getString("color"),  
 res.getString("realeseDate"),  
 res.getString("mark")  
 );  
 }  
  
 private Owner createOwner (ResultSet res) throws SQLException {  
 return new Owner(  
 res.getString("num\_passport"),  
 res.getString("id\_discount\_card"),  
 res.getString("surname"),  
 res.getString("name"),  
 res.getString("middle\_name"),  
 res.getString("num\_telephone")  
 );  
 }  
  
 private Staff createStaff (ResultSet res) throws SQLException {  
 return new Staff(  
 res.getString("id"),  
 res.getString("position"),  
 res.getString("second\_name"),  
 res.getString("name"),  
 res.getString("middle\_name"),  
 res.getString("num\_passport"),  
 res.getString("num\_phone")  
 );  
 }  
  
 private Part createPart (ResultSet res) throws SQLException {  
 return new Part(  
 res.getString("name"),  
 res.getString("cost"),  
 res.getString("guarantee")  
 );  
 }  
  
 private Service createService (ResultSet res) throws SQLException {  
 return new Service(  
 res.getString("name"),  
 res.getString("cost")  
 );  
 }  
  
 private StaffCountWork createStaffCountWork (ResultSet res) throws SQLException {  
 return new StaffCountWork(  
 res.getString("name"),  
 res.getString("second\_name"),  
 res.getString("middle\_name"),  
 res.getString("count")  
 );  
 }  
}

FindAction.java

package com.data;  
  
import com.data.Tables.Owner;  
  
public interface FindAction {  
 Owner find () throws NotFoundInDBException;  
}

Formatter.java

package com.data;  
  
import javax.swing.text.DefaultFormatter;  
  
public class Formatter extends DefaultFormatter {  
 private java.util.regex.Matcher matcher;  
  
 public Formatter(java.util.regex.Pattern regex) {  
 setOverwriteMode(false);  
 matcher = regex.matcher("");  
 }  
  
 public Object stringToValue(String string) throws java.text.ParseException {  
 if (string == null) return null;  
 matcher.reset(string);  
  
 if (! matcher.matches())  
 throw new java.text.ParseException("does not match regex", 0);  
  
 return super.stringToValue(string);  
 }  
}

NotFioundInDBException.java

package com.data;  
  
public class NotFoundInDBException extends Exception {  
 public NotFoundInDBException(String msg) {  
 super(msg);  
 }  
}

StaffCountWork.java

package com.data;  
  
import com.data.Tables.FIO;  
  
public class StaffCountWork {  
  
 private final String name;  
 private final String surname;  
 private final String middleName;  
 private final String count;  
 private final FIO fio;  
  
 public StaffCountWork (  
 String name,  
 String surname,  
 String middleName,  
 String count  
 ) {  
 this.name = name;  
 this.surname = surname;  
 this.middleName = middleName;  
 this.count = count;  
  
 fio = new FIO(name, middleName, surname);  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public String getSurname() {  
 return surname;  
 }  
  
 public String getCount() {  
 return count;  
 }  
  
 public String getMiddleName() {  
 return middleName;  
 }  
  
 public FIO getFio() {  
 return fio;  
 }  
}

TableCountModel.java

package com.data;  
  
import javax.swing.table.AbstractTableModel;  
import java.lang.reflect.Array;  
import java.util.ArrayList;  
  
public class TableCountModel extends AbstractTableModel {  
 private ArrayList <StaffCountWork> countParts = new ArrayList<>();  
 private String[] columnsHead;  
 private DB db;  
  
 public TableCountModel(DB db) {  
 this.db = db;  
 }  
  
 @Override  
 public int getRowCount() {  
 return countParts.size();  
 }  
  
 @Override  
 public int getColumnCount() {  
 return 2;  
 }  
  
 @Override  
 public String getColumnName(int column) {  
 return columnsHead[column];  
 }  
  
 @Override  
 public Object getValueAt(int rowIndex, int columnIndex) {  
 StaffCountWork selected = countParts.get(rowIndex);  
 switch (columnIndex) {  
 case 0: return selected.getFio().toString();  
 case 1: return selected.getCount();  
 default: return null;  
 }  
 }  
  
 public void updateModel ( ArrayList <StaffCountWork> newCount) {  
 countParts.clear();  
 countParts = newCount;  
 fireTableRowsUpdated(0, getRowCount());  
 }  
  
 public void setColumnHead (String[] columnsHead) {  
 this.columnsHead = columnsHead;  
 }  
}

TabelModel.java

package com.data;  
  
import com.data.Tables.Order;  
  
import javax.swing.table.AbstractTableModel;  
import java.util.ArrayList;  
  
public class TableModel extends AbstractTableModel {  
 private ArrayList<Order> orders = new ArrayList<>();  
 private DB db;  
  
 public TableModel(DB db){  
 this.db = db;  
 }  
  
 @Override  
 public int getRowCount() {  
 return orders.size();  
 }  
  
 @Override  
 public int getColumnCount() {  
 return 4;  
 }  
  
 @Override  
 public Object getValueAt(int rowIndex, int columnIndex) {  
 Order selectedOrder = orders.get(rowIndex);  
 switch (columnIndex){  
 case 0: return selectedOrder.getId();  
 case 1: return selectedOrder.getNum\_car();  
 case 2: return selectedOrder.getOwner().getFio();  
 case 3: return selectedOrder.getStaff().getFio();  
 default: return null;  
 }  
 }  
  
 @Override  
 public String getColumnName(int column) {  
 switch (column){  
 case 0: return "ID заказа";  
 case 1: return "Номер машины";  
 case 2: return "ФИО владельца";  
 case 3: return "ФИО сотрудника принявшего заказ";  
 default: return null;  
 }  
 }  
  
 void addOrder(Order order) {  
 orders.add(order);  
 fireTableRowsInserted(0, getRowCount());  
 }  
  
 public Order getRow (int rowIndex) {  
 return orders.get(rowIndex);  
 }  
  
 public void updateTable() {  
 orders.clear();  
 orders = db.getOrders();  
 fireTableRowsInserted(0, orders.size());  
 }  
}

Updater.java

package com.data;  
  
public interface Updater {  
 public void update ();  
}

Car.java

package com.data.Tables;  
  
public class Car {  
 private String num, num\_passport\_owner, color, realeseDate, mark;  
  
 private Owner owner;  
  
 public Car(String num, String num\_passport\_owner, String color, String realeseDate, String mark){  
 this.num = num;  
 this.num\_passport\_owner = num\_passport\_owner;  
 this.color = color;  
 this.realeseDate = realeseDate;  
 this.mark = mark;  
 }  
  
 public Car(String num, Owner owner, String color, String realeseDate, String mark){  
 this.owner = owner;  
 this.num = num;  
 this.num\_passport\_owner = owner.getNum\_passport();  
 this.color = color;  
 this.realeseDate = realeseDate;  
 this.mark = mark;  
 }  
  
 public static String getRequestByPassport(String num\_passport\_owner){  
 return String.format("SELECT \* FROM car WHERE num\_passport\_owner = %s", num\_passport\_owner);  
 }  
  
 public void setOwner(Owner owner) {  
 this.owner = owner;  
 }  
  
 public static String getRequestByNum(String num){  
 return String.format("SELECT \* FROM car WHERE num = %s", num);  
 }  
  
 public String getNum\_passport\_owner() {  
 return num\_passport\_owner;  
 }  
  
 public String getColor() {  
 return color;  
 }  
  
 public String getMark() {  
 return mark;  
 }  
  
 public String getNum() {  
 return num;  
 }  
  
 public String getRealeseDate() {  
 return realeseDate;  
 }  
  
 @Override  
 public String toString() {  
 if(owner == null){  
 return "1";  
 }  
 return String.format("Номер машины %s", num);  
 }  
}

DiscountCard.java

package com.data.Tables;  
  
public class DiscountCard {  
 String id, discount;  
  
 public DiscountCard(String id, String discount){  
 this.id = id;  
 this.discount = discount;  
 }  
  
 @Override  
 public String toString() {  
 return String.format("%s -> %S", id, discount);  
 }  
  
 public String getId() {  
 return id;  
 }  
}

FIO.java

package com.data.Tables;  
  
public class FIO {  
 private String surname, name, middleName, fio;  
  
 public FIO(String name, String middleName, String surname){  
 this.surname = surname;  
 this.name = name;  
 this.middleName = middleName;  
  
 fio = String.format("%s %s. %s.", surname, name.charAt(0), middleName.charAt(0));  
 }  
  
 public FIO(String str, String sep){  
 String[] res = str.split(sep);  
 this.surname = res[0];  
 this.name = res[1];  
 this.middleName = res[2];  
  
 fio = String.format("%s %s. %s.", surname, name.charAt(0), middleName.charAt(0));  
 }  
  
 public String getFio() {  
 return fio;  
 }  
  
 public String getMiddleName() {  
 return middleName;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public String getSurname() {  
 return surname;  
 }  
  
 @Override  
 public String toString() {  
 return fio;  
 }  
}

Job.java

package com.data.Tables;  
  
public abstract class Job {  
 protected String name, cost;  
  
 Job(String name, String cost){  
 this.name = name;  
 this.cost = cost;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public String getCost() {  
 return cost;  
 }  
  
 @Override  
 public String toString() {  
 return String.format("%s -> %s руб.", getName(), getCost());  
 }  
  
}

Order.java

package com.data.Tables;  
  
public class Order {  
 private String id, num\_car, num\_passport\_owner, id\_stuff;  
 private Car car;  
 private Owner owner;  
 private Staff staff;  
  
 public Order(String id, String num\_car, String num\_passport\_owner, String id\_stuff){  
 this.id = id;  
 this.num\_car = num\_car;  
 this.num\_passport\_owner = num\_passport\_owner;  
 this.id\_stuff = id\_stuff;  
 }  
  
 public Order(String id, Car car, Owner owner, Staff staff){  
 this.car = car;  
 this.owner = owner;  
 this.staff = staff;  
 this.id = id;  
 this.num\_car = car.getNum();  
 this.num\_passport\_owner = owner.getNum\_passport();  
 this.id\_stuff = staff.getId();  
 }  
  
 public String getId() {  
 return id;  
 }  
  
 public String getId\_stuff() {  
 return id\_stuff;  
 }  
  
 public String getNum\_car() {  
 return num\_car;  
 }  
  
 public String getNum\_passport\_owner() {  
 return num\_passport\_owner;  
 }  
  
 public Car getCar() {  
 return car;  
 }  
  
 public Owner getOwner() {  
 return owner;  
 }  
  
 public Staff getStaff() {  
 return staff;  
 }  
  
 public void setStaff(Staff staff) {  
 this.staff = staff;  
 }  
  
 public void setCar(Car car) {  
 this.car = car;  
 }  
  
 public void setOwner(Owner owner){  
 this.owner = owner;  
 }  
  
 public void setId(String id) {  
 this.id = id;  
 }  
  
 @Override  
 public String toString() {  
 return String.format("Номер.Машины. %s - ФИО влад. %s", num\_car, owner);  
 }  
}

Order\_Job.java

package com.data.Tables;  
  
public abstract class Order\_job {  
 protected String id\_performing, name\_service\_part;  
 protected Order order;  
 protected Staff performing;  
 protected Job job;  
  
  
 public Order\_job(Order order, String name\_service\_part, String id\_performing) {  
 this.name\_service\_part = name\_service\_part;  
 this.order = order;  
 this.id\_performing = id\_performing;  
 }  
  
 public Order\_job(Order order, Job job, Staff performing) {  
 this.order = order;  
 this.name\_service\_part = job.getName();  
 this.performing = performing;  
 this.id\_performing = performing.getId();  
 this.job = job;  
 }  
  
 public Order\_job(Job job, Staff performing) {  
 this.performing = performing;  
 this.name\_service\_part = job.getName();  
 this.performing = performing;  
 this.id\_performing = performing.getId();  
 this.job = job;  
 }  
  
 public Order getOrder() {  
 return order;  
 }  
  
 public void setOrder(Order order) {  
 this.order = order;  
 }  
  
 public Staff getPerforming() {  
 return performing;  
 }  
  
 public String getId\_performing() {  
 return id\_performing;  
 }  
  
 public String getName\_service\_part() {  
 return name\_service\_part;  
 }  
  
 public Job getJob() {  
 return job;  
 }  
  
 public abstract String getInsertRequest ();  
  
 public abstract String getTable ();  
  
 public void setJob(Job job) {  
 this.job = job;  
 }  
  
 public void setPerforming(Staff performing) {  
 this.performing = performing;  
 }  
  
 @Override  
 public String toString() {  
 return String.format("%s -> %s" , job, performing);  
 }  
}

Order\_parts.java

package com.data.Tables;  
  
public class Order\_Parts extends Order\_job {  
  
 public Order\_Parts(Order order, String name\_service\_part, String id\_performing) {  
 super(order, name\_service\_part, id\_performing);  
  
 }  
  
 @Override  
 public String getInsertRequest() {  
 return String.format("INSERT INTO `order\_parts`(`id\_order`, `name\_part`, `id\_performing`) VALUES ('%s', '%s', '%s')",  
 order.getId(), job.getName(), performing.getId());  
 }  
  
 public Order\_Parts(Order order, Part part, Staff performing) {  
 super(order, part, performing);  
 }  
  
 public Order\_Parts(Part part, Staff performing) {  
 super(part, performing);  
  
 }  
  
 public static String getRequest(String order\_id){  
 return String.format("SELECT \* FROM order\_parts WHERE id\_order = %s", order\_id);  
 }  
  
 public static String getRequest(Order order){  
 return String.format("SELECT \* FROM order\_parts WHERE id\_order = %s", order.getId());  
 }  
  
 @Override  
 public String getTable() {  
 return "order\_parts";  
 }  
}

Order\_services.java

package com.data.Tables;  
  
public class Order\_Services extends Order\_job {  
  
 public Order\_Services(Order order, String name\_service\_part, String id\_performing) {  
 super(order, name\_service\_part, id\_performing);  
  
 }  
  
 @Override  
 public String getInsertRequest() {  
 return String.format("INSERT INTO `order\_services`(`id\_order`, `name\_service`, `id\_performing`) VALUES ('%s', '%s', '%s')",  
 order.getId(), job.getName(), performing.getId());  
 }  
  
 public Order\_Services(Order order, Service service, Staff staff) {  
 super(order, service, staff);  
 }  
  
 public Order\_Services(Service service, Staff staff) {  
 super(service, staff);  
 }  
  
 public static String getRequest(String order\_id){  
 return String.format("SELECT \* FROM order\_services WHERE id\_order = %s", order\_id);  
 }  
  
 public static String getRequest(Order order){  
 return String.format("SELECT \* FROM order\_services WHERE id\_order = %s", order.getId());  
 }  
  
 @Override  
 public String getTable() {  
 return "order\_services";  
 }  
}

Owner.java

package com.data.Tables;  
  
import java.util.ArrayList;  
  
public class Owner extends Person {  
  
 private String id\_discount\_card;  
 private ArrayList <Car> cars = new ArrayList<>();  
  
 public Owner(String num\_passport, String id\_discount\_card, String surname, String name, String middle\_name, String num\_telephone){  
 super(surname, name, middle\_name, num\_telephone, num\_passport);  
  
 this.id\_discount\_card = id\_discount\_card;  
 }  
  
 public static String getRequest(String num\_passport){  
 return String.format("SELECT \* FROM owner WHERE num\_passport = %s", num\_passport);  
 }  
  
 public String getId\_discount\_card() {  
 return id\_discount\_card;  
 }  
  
 public ArrayList <Car> getCars () {  
 return cars;  
 }  
  
 public void setCars (ArrayList<Car> cars) {  
 this.cars = cars;  
 }  
  
 public void addCar (Car car) {  
 cars.add(car);  
 }  
  
 @Override  
 public String toString() {  
 return String.format("%s -> %s", getFio(), getNum\_passport());  
 }  
}

Part.java

package com.data.Tables;  
  
public class Part extends Job {  
 private String guarantee;  
  
  
 public Part(String name, String cost, String guarantee){  
 super(name, cost);  
  
 this.guarantee = guarantee;  
 }  
  
 public String getGuarantee() {  
 return guarantee;  
 }  
  
 public static String getRequest(String name){  
 return String.format("SELECT \* FROM `part` WHERE name = %s", name);  
 }  
}

Person.java

package com.data.Tables;  
  
public abstract class Person {  
 protected String num\_passport, num\_telephone;  
  
 protected FIO fio;  
  
 public Person(String surname, String name, String middle\_name, String num\_telephone, String num\_passport){  
 this.num\_passport = num\_passport;  
 this.num\_telephone = num\_telephone;  
  
 fio = new FIO(name, middle\_name, surname);  
 }  
  
 public FIO getFio() {  
 return fio;  
 }  
  
 public String getNum\_passport() {  
 return num\_passport;  
 }  
  
 public String getNum\_telephone() {  
 return num\_telephone;  
 }  
}

Services.java

package com.data.Tables;  
  
public class Service extends Job {  
  
 public Service(String name, String cost){  
 super(name, cost);  
 }  
  
 public static String getRequest(String name){  
 return String.format("SELECT \* FROM `service` WHERE name = %s", name);  
 }  
}

Staff.java

package com.data.Tables;  
  
public class Staff extends Person{  
  
 private String id, position;  
  
 public Staff(String id, String position, String surname, String name, String middle\_name, String num\_passport, String num\_telephone){  
 super(surname, name, middle\_name, num\_telephone, num\_passport);  
  
 this.id = id;  
 this.position = position;  
 }  
  
 public static String getRequest(String id) {  
 return String.format("SELECT \* FROM staff WHERE id = %s", id);  
 }  
  
 public String getId() {  
 return id;  
 }  
  
 public String getPosition() {  
 return position;  
 }  
  
 @Override  
 public String toString() {  
 return String.format("%s -> %s", fio, position);  
 }  
}