Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение  
 высшего образования Московский авиационный институт

(национальный исследовательский университет)

Филиал «РКТ» МАИ в г. Химки Московской области

**ОТЧЁТ ПО ПРАКТИКЕ**

Специальность 09.02.07 — Информационные системы и программирование

**Выполнила:**

Студентка группы ИСП–41–19

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Кузьмина А.Д./

**Проверил:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Гуров В.В./

Химки, 2023

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc127915496)

[1. Разработка форм для приложения 4](#_Toc127915497)

[2. Проектирование базы данных 7](#_Toc127915498)

[2.1. Концептуальная модель 7](#_Toc127915499)

[2.2. Логическая модель 9](#_Toc127915500)

[2.3. Физическая модель 10](#_Toc127915501)

[2.4. Диаграмма вариантов использования 11](#_Toc127915502)

[3. Разработка и настройка базы данных. 13](#_Toc127915503)

[4. Разработка приложения 17](#_Toc127915504)

[4.1 Разработка авторизации 17](#_Toc127915505)

[4.2 Разработка функционала приложения 18](#_Toc127915506)

[ЗАКЛЮЧЕНИЕ 26](#_Toc127915507)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 27](#_Toc127915508)

[ПРИЛОЖЕННИЕ А. Иллюстрации форм для приложения 28](#_Toc127915509)

[ПРИЛОЖЕННИЕ Б. Запросы на создание таблиц 31](#_Toc127915510)

[ПРИЛОЖЕННИЕ В. Исходный код приложения 33](#_Toc127915511)

# **ВВЕДЕНИЕ**

Учебная практика является важной составной частью учебного процесса подготовки высококвалифицированных специалистов. В период прохождения практики студенты конкретизируют и закрепляют теоретические знания, приобретенные в процессе обучения, овладевают навыками практической работы, стараются применить полученные знания на практике. Практика является подводящим этапом в процессе подготовки будущего специалиста к самостоятельной практической деятельности.

Учебная практика по профессиональному модулю ПМ.02 «Осуществление интеграции программных модулей» проходила в филиале «РКТ» МАИ в городе Химки Московской области с 9 февраля 2023 года по 23 февраля 2023 года. Работа велась в очном формате.

Целью данной практики является разработка приложения для точки проката горнолыжного курорта «Игора».

# **Разработка форм для приложения**

Для визуализации разрабатываемого приложения необходимы формы. Для их разработки использовалось приложение «Qt Designer» (результат разработки форм смотреть в приложении А).

При разработке форм следует соблюдать следующие требования:

1. Все экранные формы пользовательского интерфейса должны иметь заголовок с логотипом (в ресурсах). Не искажайте логотип (не изменяйте изображение, его пропорции, цвет);
2. Должна присутствовать возможность изменения размеров окна, где это необходимо;
3. Увеличение размеров окна должно увеличивать размер контентной части, например, таблицы с данными из БД);
4. Должно присутствовать ограничение на минимальный размер окна;
5. Группировка элементов (в логические категории);
6. Последовательный пользовательский интерфейс, позволяющий перемещаться между существующими окнами в приложении (в том числе обратно, например, с помощью кнопки «Назад»);
7. Соответствующий заголовок на каждом окне приложения (не должно быть значений по умолчанию типа MainWindow, Form1 и тп);
8. Использование шрифта Comic Sans MS;
9. В качестве основного фона используется белый цвет, а в качестве дополнительного: RGB (118, 227, 131). Для акцентирования внимания пользователя на целевое действие интерфейса используйте цвет RGB (73, 140,81) (смотреть рисунок 1).

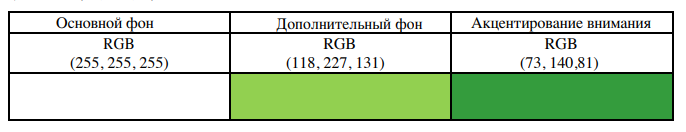


Рисунок 1 — Цветовая схема

Инструкция при разработке форм в «Qt Designer»:

1. Для создания главного окна необходимо выбрать Main Window, а для создания диалогового окна нужно выбрать Dialog without Buttons
2. Для использования виджетов необходимо их перетащить из Widget Box в ваше окно. Рассмотрим основные виджеты:

* Vertical Layout – для группировки виджетов вертикально
* Horizontal Layout – для группировки виджетов горизонтально
* Grid Layout – для группировки виджетов по сетке
* Form Layout – для группировки виджетов по строкам в 2 колонки (смотреть рисунок 2).

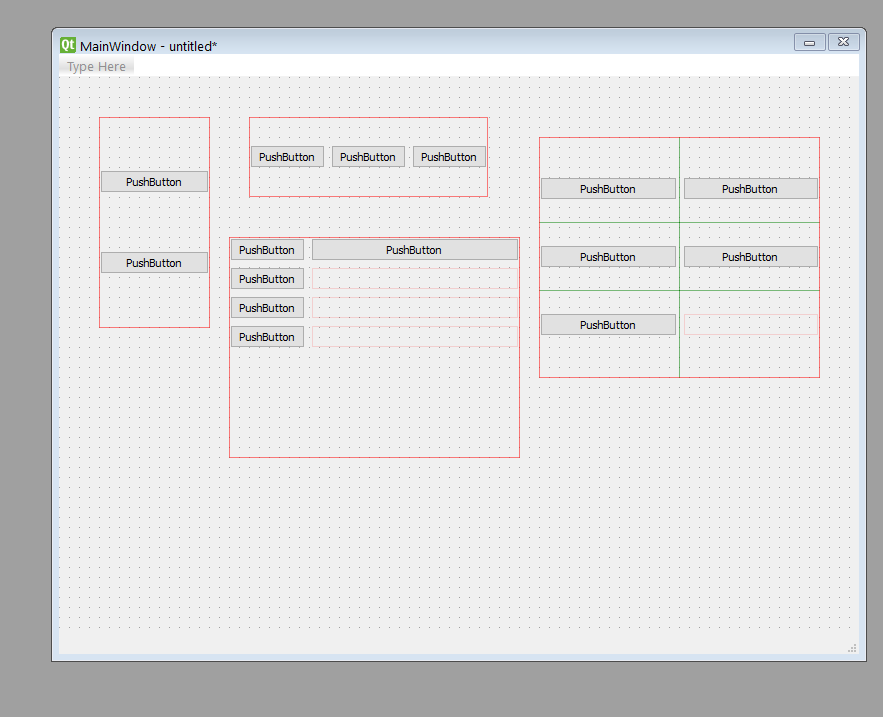


Рисунок 2 — Layout

* Horizontal Spacer и Vertical Spacer нужны, чтобы раздвинуть виджеты, заполнив пространство между ними (смотреть рисунок 3).

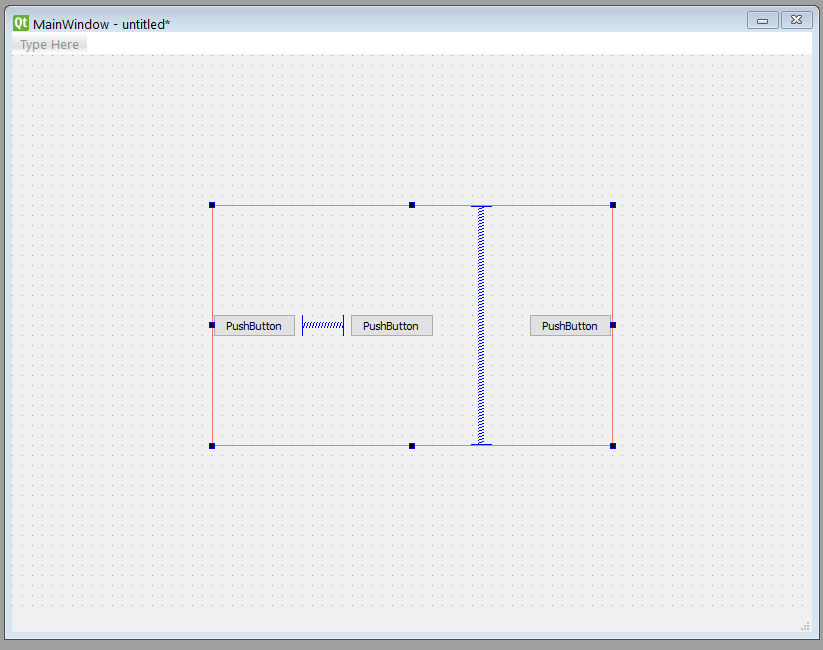


Рисунок 3 — Spacer

* Push Button – кнопка при нажатии, на которую посылается сигнал, который может быть обработан в коде (смотреть рисунок 3).
* Table View – табличное представление элементов для отображения данных из модели (базы данных).
* List Widget — это виджет, который используется для отображения списка элементов.
* StackedWidget предоставляет стек виджетов, в котором одновременно виден только один виджет.
* ComboBox представляет собой комбинированную кнопку и всплывающий список.
* LineEdit – это однострочный текстовый редактор, в который можно записать текст.
* SpinBox разработан для обработки целых чисел, с помощью стрелочек можно уменьшить или увеличить число.
* Label – для вывода текста на экран.
* GraphicsView предоставляет виджет для отображения содержимого QGraphicsScene.

1. Для группировки элементов необходимо разместить их в layout, а также указать layout для основного окна.
2. Для изменения названия виджета необходимо выбрать нужный виджет (лкм по виджету) и в «property editor» изменить поле «objectName».
3. Для изменения цвета и шрифта виджета необходимо нажать на нужный виджет пкм и выбрать «Change Style Sheet…», а после установить необходимые параметры.
4. Для изменения минимального размера виджета необходимо выбрать нужный виджет и в «property editor» изменить параметры «minimumSize» (width, height).
5. Для переключения между окошками основного окна был использован виджет «Stacked Widget». Чтобы создать новую страничку в «Stacked Widget» необходимо нажать по нему пкм и «Insert Page».
6. Для установления изображения в label необходимо изменить поле «pixmap», а для кнопки поле «icon».
7. Чтобы установить иконку окна нужно изменить поле «windowIcon», а для изменения заголовка окна нужно изменить поле «windowTitle».

# **Проектирование базы данных**

Для создания базы данных необходимо определить основные сущности, атрибуты сущностей, а также реализовать связи между сущностями.

Сущности определяются в процессе анализа предметной области. Вычленяются ключевые элементы, определяются их свойства, устанавливаются взаимосвязи между ними.

Предметная область — Горнолыжный курорт «Игора». Сущности и атрибуты:

* Сущность «Employee». Атрибуты: emp\_id; position; emp\_name; login; password; last\_log; log\_type; photo;
* Сущность «Client». Атрибуты: cl\_id; cl\_name; PassportData; Birthdate; address; e–mail; password;
* Сущность «History». Атрибуты: his\_id; logDate; logoutdate; lock; emp\_login;
* Сущность «Services». Атрибуты: serv\_id; serv\_Title; serv\_code; price;
* Сущность «Orders». Атрибуты: ord\_id; order\_code; create\_date; create\_time; status; close\_time; rent\_time.
  1. **Концептуальная модель**

Первым этапом в проектировании базы данных является построение моделей — концептуальной, логической, физической.

Концептуальная модель — это абстрактная модель, определяющая структуру моделируемой системы, свойства её элементов и причинно–следственные связи, присущие системе и существенные для достижения цели моделирования.

Принцип построения концептуальной модели довольно прост: все элементы схемы представлены в виде геометрических фигур. Сущности — прямоугольниками, атрибуты — овалами, связи между сущностями — ромбом. Любую концептуальную модель, или же ER–диаграмму можно создать самым примитивным способом — на бумаге при помощи карандаша.

Рисуется сущность, затем возле неё некое количество атрибутов этой сущности. Таким же образом рисуются еще несколько сущностей с набором атрибутов. Каждая сущность соединяется со своим атрибутом прямой линией. Затем между самими сущностями определяются связи. Рисуется связь, соединяющая линиями две сущности. Определяется тип связи — «один к одному», «один ко многим», «многие ко многим», в зависимости от самих сущностей. Связи получают название в соответствии с модальностью, то есть называются по действию, которое одна сущность совершает по отношению к другой.

Для реализации концептуальной модели (далее — ER–диаграммы) используется выбранный онлайн–сервис ERDPlus.com. Результат реализации концептуальной модели, используя сервис ERDPlus.com, представлен на рисунке 4.

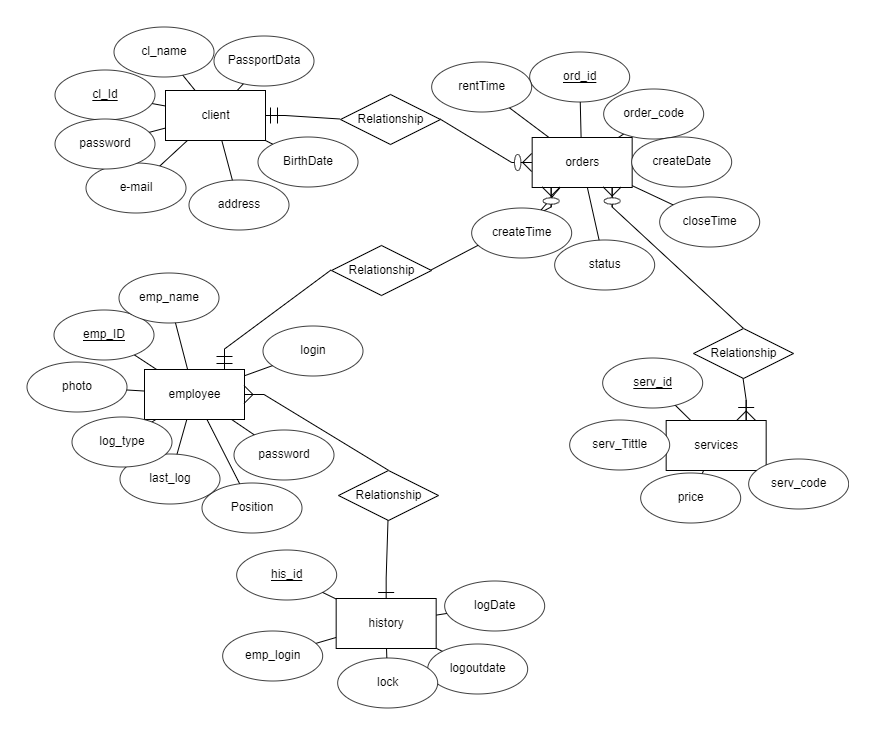


Рисунок 4 — ER–диаграмма.

* 1. **Логическая модель**

Логическая модель – графическое представление структуры базы данных с учетом принимаемой модели данных (иерархической, сетевой, реляционной и так далее), независимое от конечной реализации базы данных и аппаратной платформы.

Иными словами, она показывает, ЧТО хранится в базе данных (объекты предметной области, их атрибуты и связи между ними).

Логическая модель (или реляционная схема) строится следующим образом: создаются сущности (далее — таблицы), в теле которых указываются названия атрибутов (далее — поля). Важные поля назначаются первичными и внешними ключами таблицы. Между таблицами устанавливаются связи по установленным ранее ключам.

Логическую модель можно также реализовать, используя сервис ERDPlus.com.

После завершения работы с концептуальной моделью (ER–диаграммой), необходимо её сохранить и выйти в главное меню, где будет представлен список всех ER–диаграмм и реляционных схем. Затем в области необходимой нам ER–диаграммы необходимо вызвать контекстное меню, нажав на три точки. Далее нажать на кнопку “Convert to Relational Schema” и дать название новой реляционной модели. Нажать кнопку “Create”.

В результате на основе ER–диаграммы будет создана реляционная схема, то есть — логическая модель, представленная на рисунке 5.

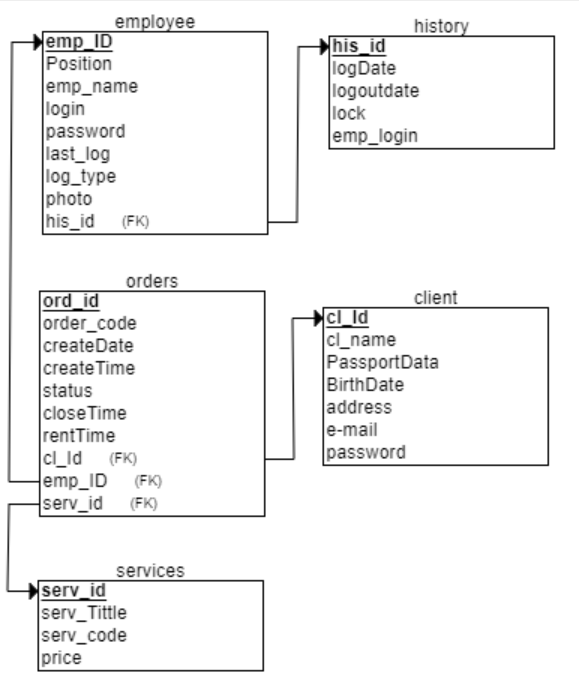


Рисунок 5 — Логическая модель.

* 1. **Физическая модель**

Физическая модель базы данных — это модель данных, которая определяет, каким образом представляются данные, и содержит все детали, необходимые СУБД для создания базы данных**.**

Фактически, это заключительный этап проектирования базы данных. В процессе реализации физической модели определяются типы данных полей в таблицах с учетом работы конкретной выбранной СУБД.

Реализовать физическую модель можно средствами СУБД MySQL Workbench. После создания базы данных и добавления всех необходимых таблиц в неё, во вкладке “Database” по нажатию пункта “Reverse Engineer”, выбрав нужную базу данных из списка, будет создана схема физической модели. Созданная физическая модель базы данных представлена на рисунке 6.

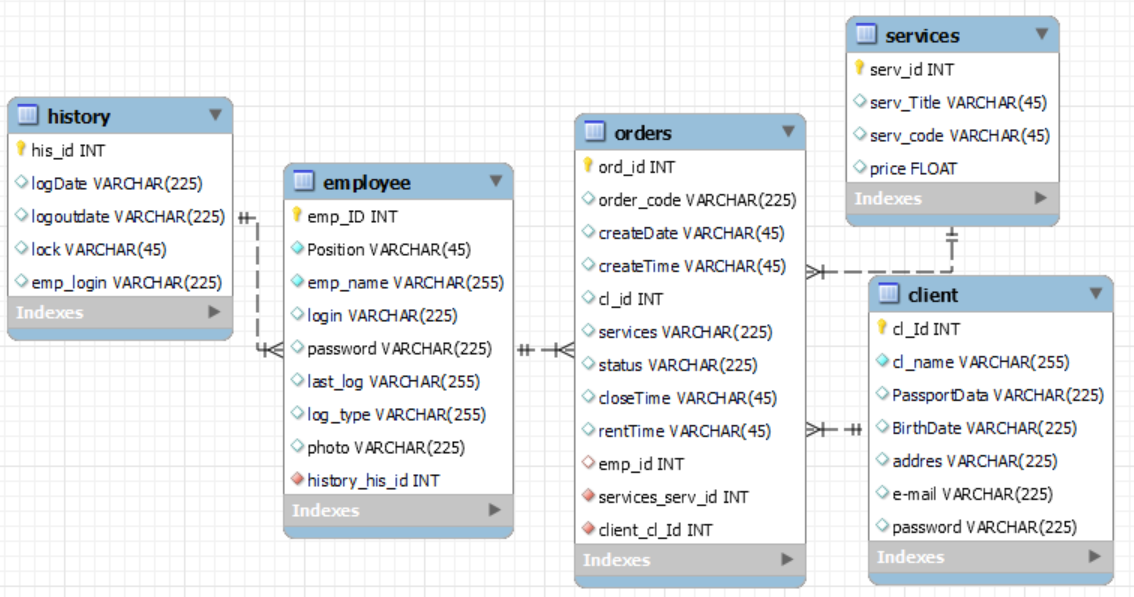


Рисунок 6 — Физическая модель.

* 1. **Диаграмма вариантов использования**

UML (Universal Modeling Language) — универсальный язык моделирования, который был разработан компанией Rational Software с целью создания наиболее оптимального и универсального языка для описания как предметной области, так и конкретной задачи в программировании.

Диаграмма вариантов использования (англ. use–case diagram) – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей.

Реализовать диаграмму вариантов использования можно с помощью сайта DiagramEditor.

Перейти по ссылке: <https://www.diagrameditor.com/>

1. В меню нажать на подпункт «General».
2. Перетащить фигуры, которые необходимо включить в схему, из окна «General» на страницу. Чтобы изменить подписи, необходимо в всплывающем меню справа выбрать вкладку «Text».
3. Для создания нужных типов связи выбирается любой тип соединения из доступных. Нужным образом связь настраивается во вкладке «Style». Альтернативный способ создания связи: переход в подпункт меню «UML» и выбор нужной связи там.
4. Для сохранения созданной диаграммы нужно кликнуть на кнопку «File» в левом верхнем углу. Во всплывающем меню кликнуть на кнопку «Save as…». Дать диаграмме имя и нажать кнопку «Download».

На точке проката работают администратор, старший смены и продавец. Продавец и старший смены имеют одинаковый функционал: добавление клиента, добавление заказа, просмотр всех заказов. Старший смены также может просматривать всех клиентов. Администратор точки проката может проконтролировать всех пользователей по истории входа, просматривать заказы, работать с услугами. Созданная диаграмма вариантов использования представлена на рисунке 7.

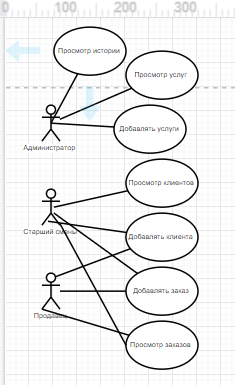


Рисунок 7 — Диаграмма вариантов использования.

1. **Разработка и настройка базы данных**.

В качестве СУБД была выбрана MySQL.

MySQL — это реляционная система управления базами данных (СУБД), которая распространяется как свободное программное обеспечение. Является одной из наиболее популярных, так как отличается гибкостью, легкостью, удобством в использовании.

Для создания базы данных в выбранной СУБД необходимо провести следующие операции:

Провести настройку следующим образом:

1. Скачать СУБД MySQL по ссылке: <https://dev.mysql.com/downloads/installer/>
2. Откроется окно выбора варианта установки MySQL:

* Developer – это вариант по умолчанию, он предполагает установку всего того, что требуется разработчику, это и MySQL Server, и MySQL Workbench и другие инструменты для работы с MySQL;
* Server Only – этот вариант предполагает установку только сервера MySQL;
* Client Only – этот вариант предполагает установку только клиентской части для работы с MySQL Server, т.е. серверная часть устанавливаться не будет;
* Full – установка всех включённых в дистрибутив компонентов;
* Custom – выборочная установка, в данном случае Вы можете установить только то, что нужно Вам.

Для данной работы рекомендуется выбрать вариант установки «Developer», так как он содержит в себе все необходимые элементы для разработки и администрирования базы данных.

После выбора данного варианта начинается установка MySQL.

1. В изменившемся окне выводится список всех устанавливаемых расширений для MySQL. Необходимо нажать кнопку «Execute». Если установка выполнена успешно, рядом с пакетами будут зелёные галочки, а также надпись «Complete»., что представлено на рисунке 8.

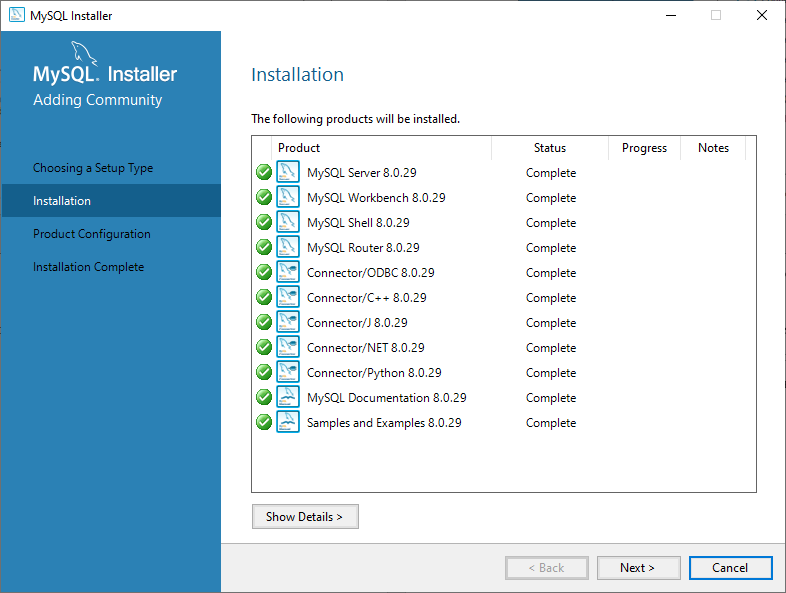


Рисунок 8 — Установка пакетов завершена.

На данном этапе процесс установки СУБД переходит в настройку работы СУБД.

В открывшемся окне производится настройка соединения, назначается тип соединения, порт.

По нажатию кнопки «Next» открывается меню выбора режима аутентификации в базе посредством СУБД. Для данной работы необходимо выбрать рекомендуемый вариант «Use Strong Password Encryption for Authentication». Далее по нажатию кнопки «Next» открывается меню создания профиля для работы с сервером и БД. Для добавления нового профиля, необходимо нажать на кнопку «Add User», где указывается имя профиля и пароль, назначается его роль. Меню создания профиля показано на рисунке 9.

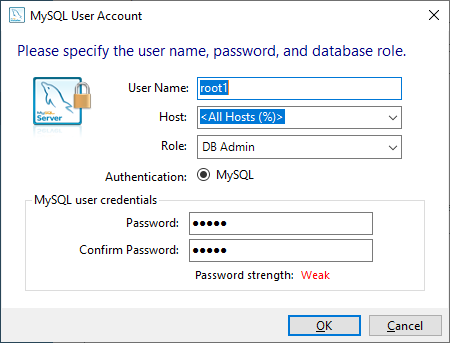


Рисунок 9 — Продолжение настройки профиля.

Следующий шаг — реализация запросов на создание таблиц и заполнение их данными. Создаются таблицы [4]. Ниже представлены SQL–запросы на создание таблиц:

1. Создание таблицы «Client»:

CREATE TABLE `client` (

`cl\_Id` int NOT NULL AUTO\_INCREMENT,

`cl\_name` varchar(255) NOT NULL,

`PassportData` varchar(225) DEFAULT NULL,

`BirthDate` varchar(225) DEFAULT NULL,

`addres` varchar(225) DEFAULT NULL,

`e–mail` varchar(225) DEFAULT NULL,

`password` varchar(225) DEFAULT NULL,

PRIMARY KEY (`cl\_Id`)

) ENGINE=InnoDB AUTO\_INCREMENT=11 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci

1. Создание таблицы «Employee»:

CREATE TABLE `employee` (

`emp\_ID` int NOT NULL,

`Position` varchar(45) NOT NULL,

`emp\_name` varchar(255) NOT NULL,

`login` varchar(225) DEFAULT NULL,

`password` varchar(225) DEFAULT NULL,

`last\_log` varchar(255) DEFAULT NULL,

`log\_type` varchar(255) DEFAULT NULL,

`photo` varchar(225) DEFAULT NULL,

PRIMARY KEY (`emp\_ID`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

Остальные запросы на создание таблиц по структуре идентичны представленным выше и приведены в приложении Б. Результат выполнения запросов на создание таблиц представлен на рисунке 10:

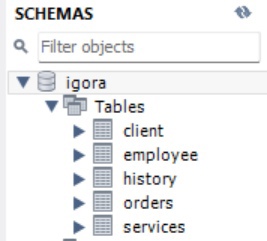


Рисунок 10 — Создана БД и таблицы.

Благодаря функционалу MySQL Workbench у разработчиков есть возможность добавлять данные несколькими способами:

* Написание SQL–запросов в поле ввода текста;
* Ручное добавление данных в ячейки таблиц базы данных;
* Импорт данных в БД из стороннего файла.

В качестве способа добавления данных в БД был выбран способ импорта данных из стороннего файла, поскольку, по сравнению с другими способами, не требует слишком много времени и интуитивно понятен. Для этого необходимо открыть меню со всеми таблицами, и выбрать ту таблицу, в которую будет производиться добавление данных. В качестве примера, выбрана таблица «Client». Необходимо нажать по названию таблицы и выбрать функцию «Table data import wizard». Далее в открывшемся окне указывается путь к csv файлу, предварительно созданному при использовании Microsoft Excel. Затем необходимо указывать, создавать ли новую таблицу, или же выбрать уже существующую. На рисунке 11 представлен результат импорта данных.

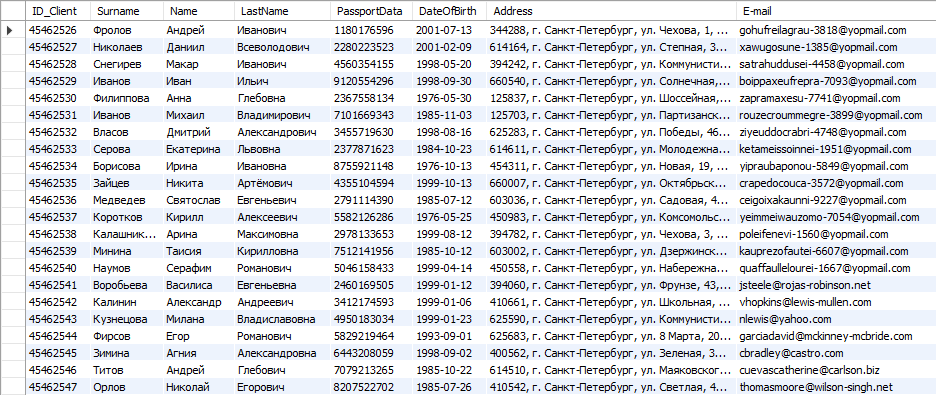


Рисунок 11 — Результат добавления данных.

# **Разработка приложения**

* 1. **Разработка авторизации**

Для входа сотрудника в приложение ему необходимо авторизоваться в системе точки проката горнолыжного курорта «Игора» – ввести верный логин и пароль (смотреть рисунок 12).

Авторизация реализована в классе DialogAutorization, с текстом данного участка кода можно ознакомиться в приложении В.

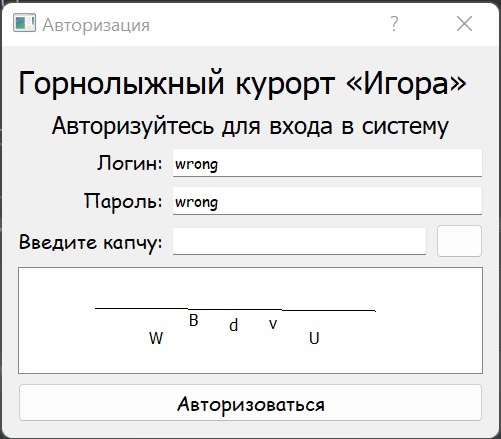


Рисунок 12 — Окно входа

Требования к авторизации:

1) Блокировка входа на 3 минуты с момента выхода из прошлой сессии (окончание времени сессии).

2) Капча.

— Появляется после второй попытки неуспешной авторизации (неправильно введен логин или пароль).

— Должна состоять минимум из трёх символов – буквы и цифры, которые выведены не в одной линии.

— Символы должны быть либо перечеркнуты, либо наложены друг на друга

— Должна присутствовать возможность повторной генерации captcha, если пользователю непонятны символы из–за шума.

3) Разграничение прав доступа к объектам интерфейса путем присвоения каждому пользователю определенного приоритета.

4) Сообщение о неуспешном входе после второй попытки.

5) Пароль.

— Скрыт маской ввода.

— Должна быть реализована возможность просмотра введенного пароля.

6) Сохранение истории входа сотрудников.

В ходе практике были реализованы пункты 2–6.

В классе DialogAutorization представлены функции, реализующие перечисленные выше требования, а также функции подключения и описания интерфейса.

1) \_\_init\_\_ – В данной функции загружается форма авторизации, объявляются переменные, создание сцены для «graphicsView» (данная сцена необхадима для графического отображения символов и форм, с помощью нее реализована функции отображения капчи), скрывается недоступный пользователю интерфейс, описывается маска ввода пароля.

2) autoriz – основная функция авторизации, перечисляющая условия проверки введенных данных, через эту функцию система обращается к БД для получения данных.

3) hide\_pas – функция переключения между режимами отображения пароля, с маской в виде точек или без нее.

4) gen\_captcha – функция генерации капчи, вызывается при двух неуспешных попыток входа в систему, а также при нажатии на кнопку «reboot\_btn». Выводит капчу в «graphicsView» и возвращает значение капчи.

5) error – функция вывода оповещающего об ошибке входа окна. Вызывается при неверном вводе пользователем логина, пароля, капчи.

6) shif\_head\_open – функция открытия окна старшего смены, вызывается при нажатии на кнопку autorization\_btn с условием ввода данных, принадлежащих сотруднику данной должности.

7) admin\_open – функция открытия окна администратора, вызывается при нажатии на кнопку autorization\_btn с условием ввода данных, принадлежащих сотруднику данной должности.

8) seller\_open – функция открытия окна продавца, вызывается при нажатии на кнопку autorization\_btn с условием ввода данных, принадлежащих сотруднику данной должности.

**Разработка функционала приложения**

После авторизации пользователю будет предоставлен доступ к соответствующему окну, в зависимости от роли пользователя

Окно администратора и его функционал описаны в классе AdminMenu.

Данное окно содержит поле вывода информации, а также кнопки для переключения между таблицами, открытия других окон и выхода из системы.

В данном классе есть следующие функции:

1) \_\_init\_\_ – функция предназнаяена для загрузки формы, подключения интерфейса к другим функциям.

2) orders – функция обращения к базе данных с запросом на получение информации таблицы заказов, вызывается при нажатии кнопки «orders\_btn». В данной функции описана запись данных в таблицу, формируемую на экране.

3) services – функция обращения к базе данных с запросом на получение информации таблицы услуг, вызывается при нажатии кнопки «services\_btn». В данной функции описана запись данных в таблицу, формируемую на экране.

4) history – функция обращения к базе данных с запросом на получение информации таблицы истории входа, вызывается при нажатии кнопки «history \_btn». В данной функции описана запись данных в таблицу, формируемую на экране.

5 add\_service – функция вызова диалогового окна, предназначенного для добавления данных в таблицу услуг.

6) exit – функция выхода из системы и внесения в таблицу истории входа записи о выходе.

Окно старшего смены и его функционал описаны в классе ShiftHeadMenu.

Данное окно содержит поле вывода информации, а также кнопки для переключения между таблицами, открытия других окон и выхода из системы.

В данном классе есть следующие функции:

1) \_\_init\_\_ – функция предназнаяена для загрузки формы, подключения интерфейса к другим функциям.

2) orders – функция обращения к базе данных с запросом на получение информации таблицы заказов, вызывается при нажатии кнопки «OrdersTable». В данной функции описана запись данных в таблицу, формируемую на экране.

3) clients – функция обращения к базе данных с запросом на получение информации таблицы клиенов, вызывается при нажатии кнопки «clientTable». В данной функции описана запись данных в таблицу, формируемую на экране.

4) add\_client – функция вызова диалогового окна, предназначенного для добавления данных в таблицу клиентов.

5) add\_order – функция вызова диалогового окна, предназначенного для добавления данных в таблицу заказов.

6) exit – функция выхода из системы и внесения в таблицу истории входа записи о выходе.

Окно продавца и его функционал описаны в классе SellerMenu.

Данное окно содержит поле вывода информации, а также кнопки для переключения между таблицами, открытия других окон и выхода из системы.

В данном классе есть следующие функции:

1) \_\_init\_\_ – функция предназнаяена для загрузки формы, подключения интерфейса к другим функциям.

2) orders – функция обращения к базе данных с запросом на получение информации таблицы заказов, вызывается при открытии окна. В данной функции описана запись данных в таблицу, формируемую на экране.

3) add\_client – функция вызова диалогового окна, предназначенного для добавления данных в таблицу клиентов.

4) add\_order – функция вызова диалогового окна, предназначенного для добавления данных в таблицу заказов.

5) exit – функция выхода из системы и внесения в таблицу истории входа записи о выходе.

Для пользователя «Администратор»

Для входа от имени администратора пользователю нужно ввести следующие данные:

1. в поле «Логин»: 1
2. в поле «Пароль»: 1

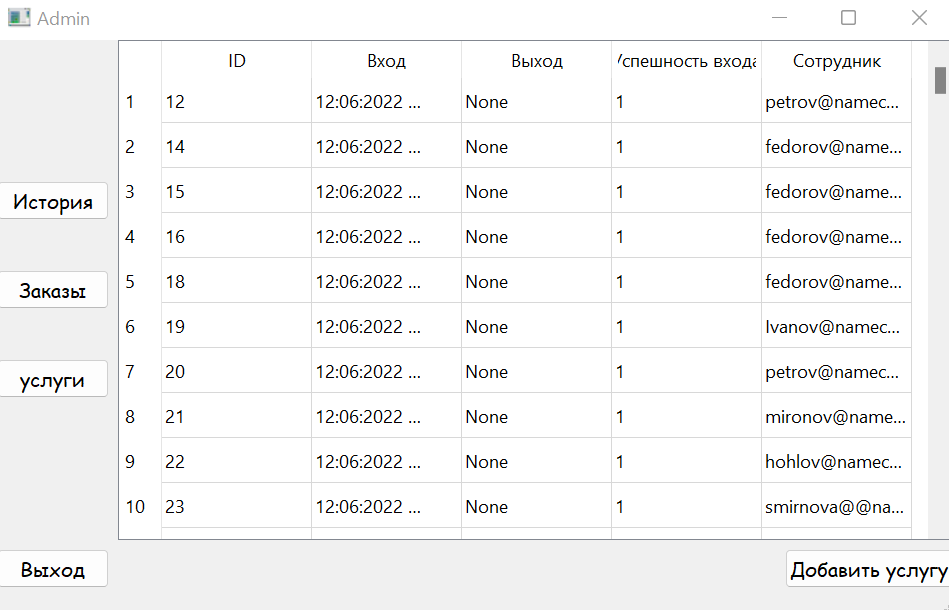
Администратору доступна возможность просмотра истории входа всех сотрудников, окно представлено на рисунке 13.

Рисунок 13 – История входа.

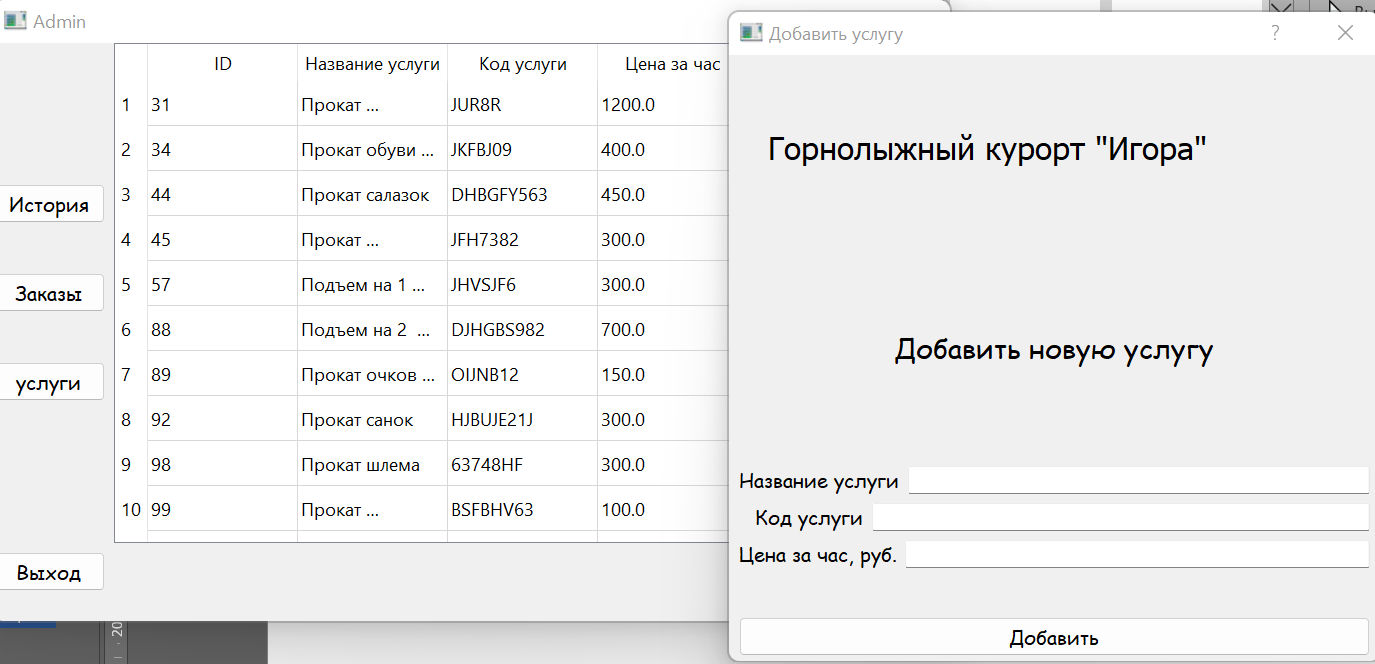
Пользователь также может просматривать и добавлять доступные услуги, что видно на рисунке 14.

Рисунок 14 – Окно услуг.

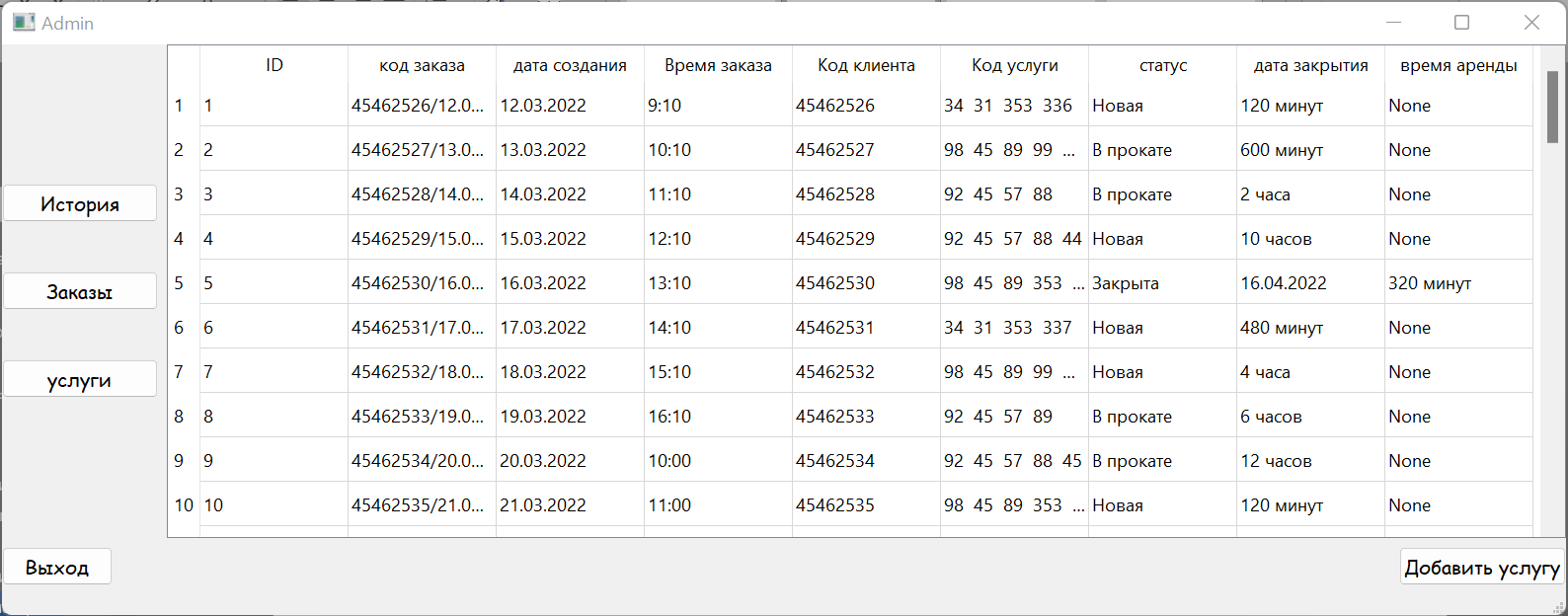
У администратора имеется возможность просмотреть перечень доступных заказов, рисунок 15.

Рисунок 15 – Доступные заказы.

Для пользователя «Старший смены»

Для входа от имени начальника смены пользователю нужно ввести следующие данные:

1. в поле «Логин»: 3
2. в поле «Пароль»: 3

Начальнику смены доступна возможность просмотра всех имеющихся клиентов, а также добавления новых, рисунок 16.

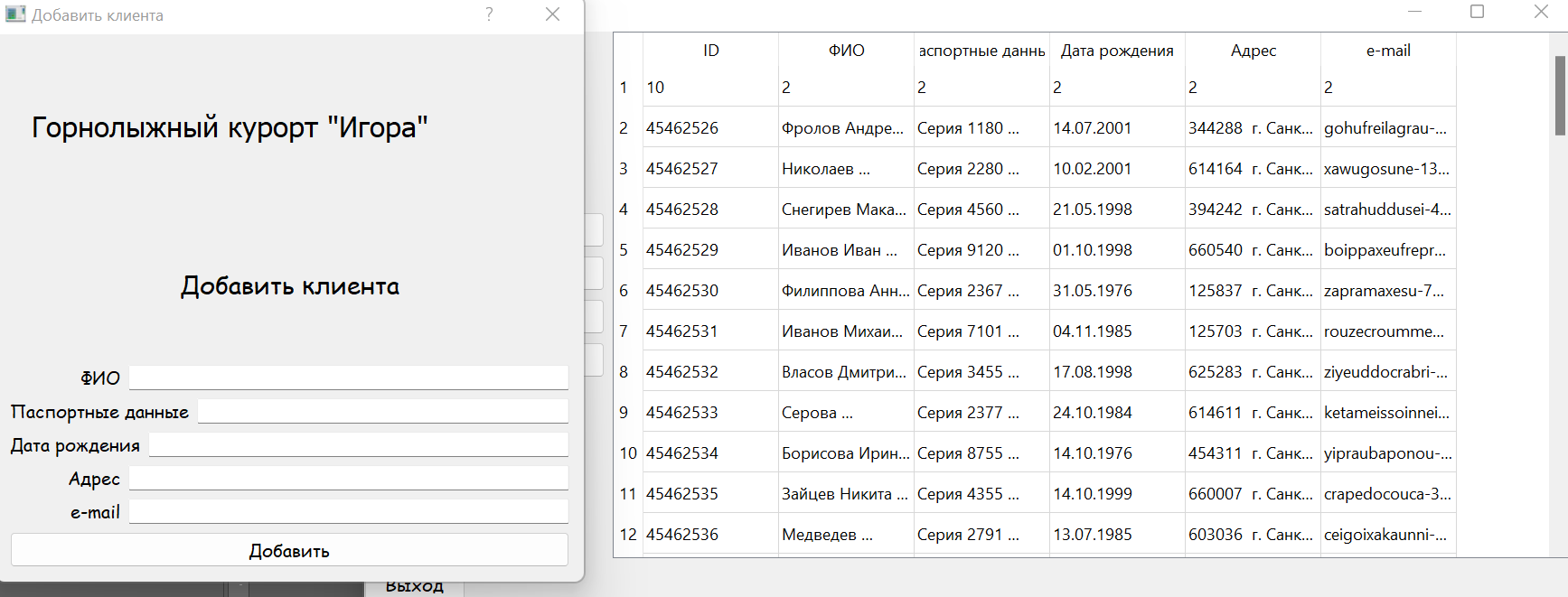


Рисунок 16 – Окна управления клиентами.

Пользователь также может просматривать доступные заказы и добавлять новые, рисунок 17.

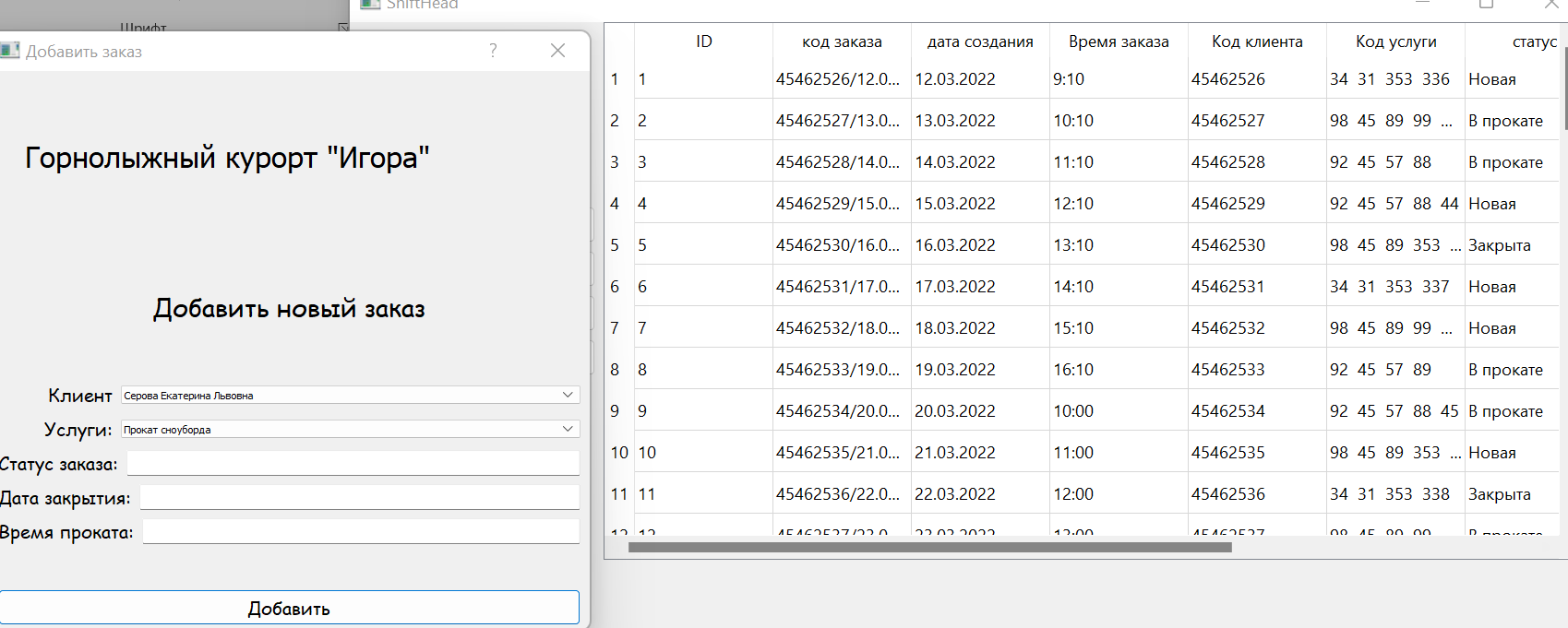


Рисунок 17 – окна управления заказами.Для пользователя «Продавец»

«Продавец» имеет возможность просмотра таблицы заказов и добавления нового заказа, а также добавления нового клиента.

Вход от имени продавца осуществляется следующим образом:

1. в поле «Логин»: 2
2. в поле «Пароль»: 2

Продавцу доступна возможность добавления нового клиента.

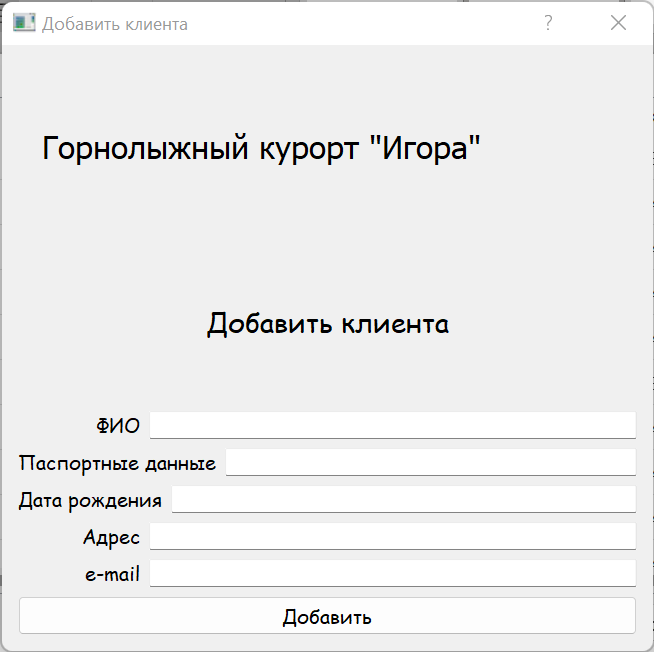


Рисунок 18 – Окно добавления клиента.

Пользователю доступен просмотр всех заказов, окно просмотра представлено на рисунке 19.

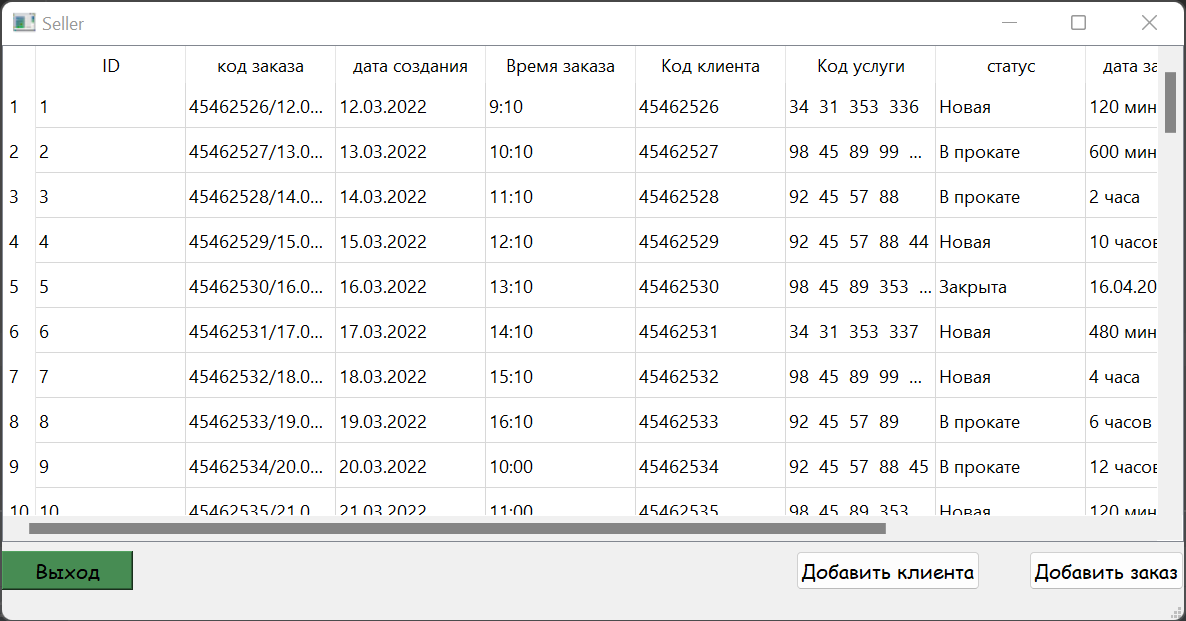


Рисунок 19 – Окно просмотра всех заказов.

Продавец также имеет возможность добавить новый заказ, диалоговое окно добавления заказа представлено на рисунке 20.

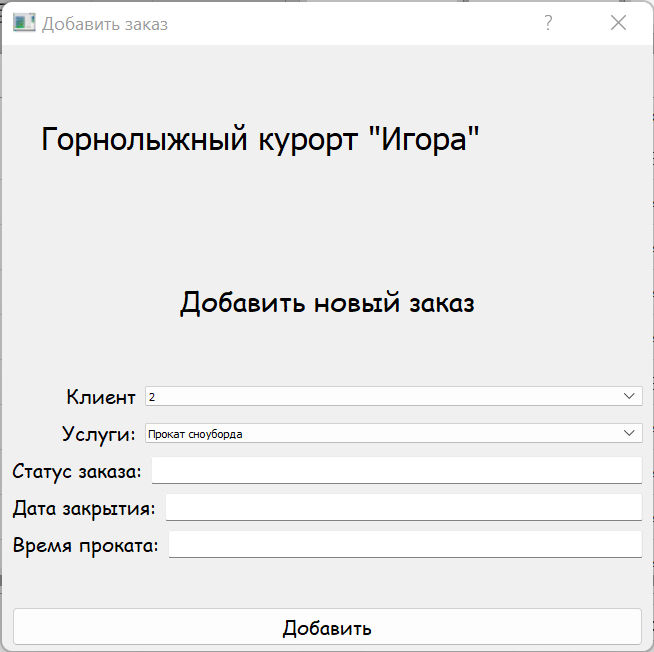


Рисунок 20 – Окно добавления заказа.

# **ЗАКЛЮЧЕНИЕ**

Во время прохождения учебной практики были закреплены теоретические знания, полученные за прохождение программы 4 курса СПО по специальности 09.02.07 «Информационные системы и программирование», сформированы основные навыки и умения для освоения рабочей профессии. Были получены практические навыки разработки приложения для организованной работы с базой данных.

Были закреплены необходимые навыки работы в интегрированной среде разработки PyCharm, используя хранилище репозиториев Gogs.

В ходе прохождения учебной практики была достигнута её цель и решены следующие задачи:

* Разработаны формы в формате .ui в приложении «Qt Designer» для дальнейшего их использования в визуализации разрабатываемого приложения.
* Разработаны классы окон приложения и описаны их функционал.
* Обеспечено взаимодействие приложения с базой данных на удаленном сервере.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. ГОСТ 7.32–2017 отчет о научно–исследовательской работе. М., Стандартинформ, 2017.
2. Прохоренок Н. А. Python 3 и PyQt. Разработка приложений. — СПб.: БХВ–Петербург, 2012. — 704 с.: ил.
3. Буйначев С. К. Основы программирования на языке python: учебное пособие / С. К. Буйначев, Н. Ю. Боклаг. – Екатеринбург: Издательствово Урал. ун–та, 2014. – 91, [1] c.
4. 3) 2 Лутц М. Изучаем Python, 3 е издание – Пер. с англ. – СПб.: Символ Плюс, 2009. – 848 с., ил.
5. Inno Setup Downloads. — URL: https://jrsoftware.org/isdl.php (дата обращения 2022–06–13).

**ПРИЛОЖЕННИЕ А. Иллюстрации форм для приложения**

Иллюстрация формы авторизации с капчей (смотреть рисунок А.1).

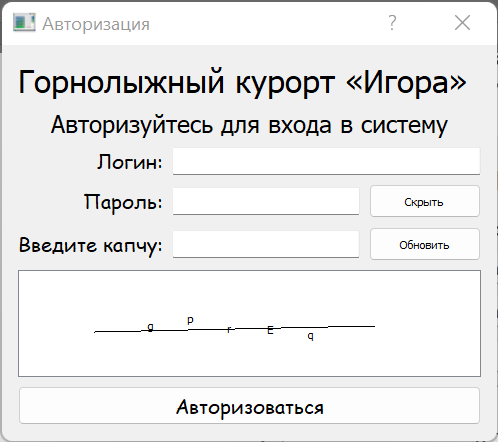


Рисунок А.1

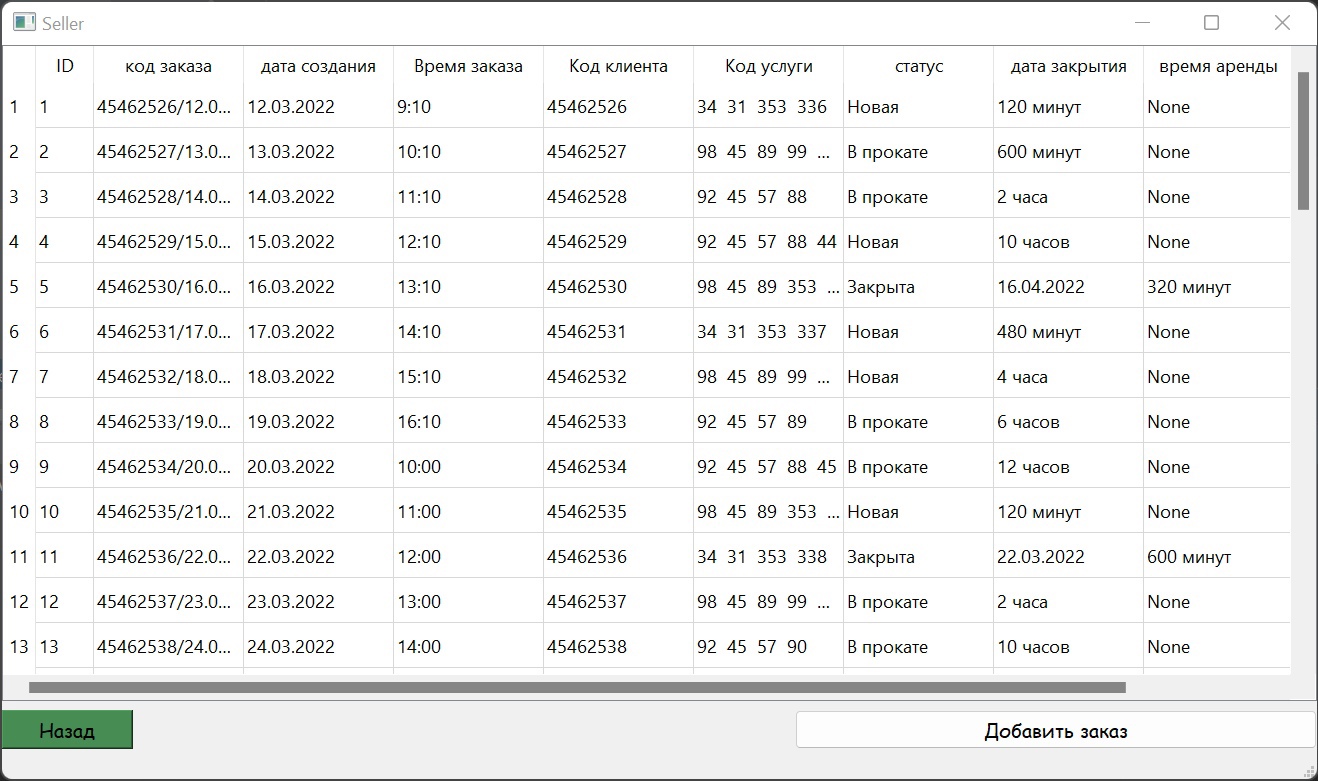
Иллюстрация формы просмотра заказов (смотреть рисунок А.2).

Рисунок А.2

Иллюстрация формы добавления заказа (смотреть рисунок А.3).

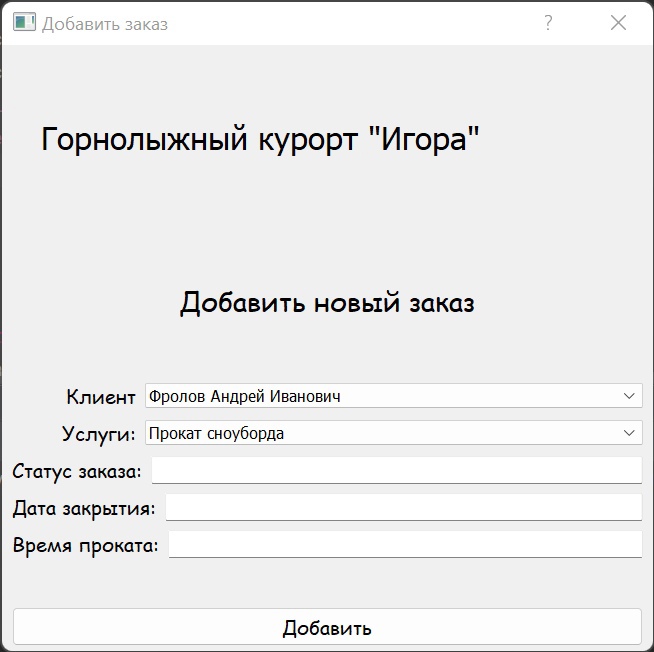


Рисунок А.3

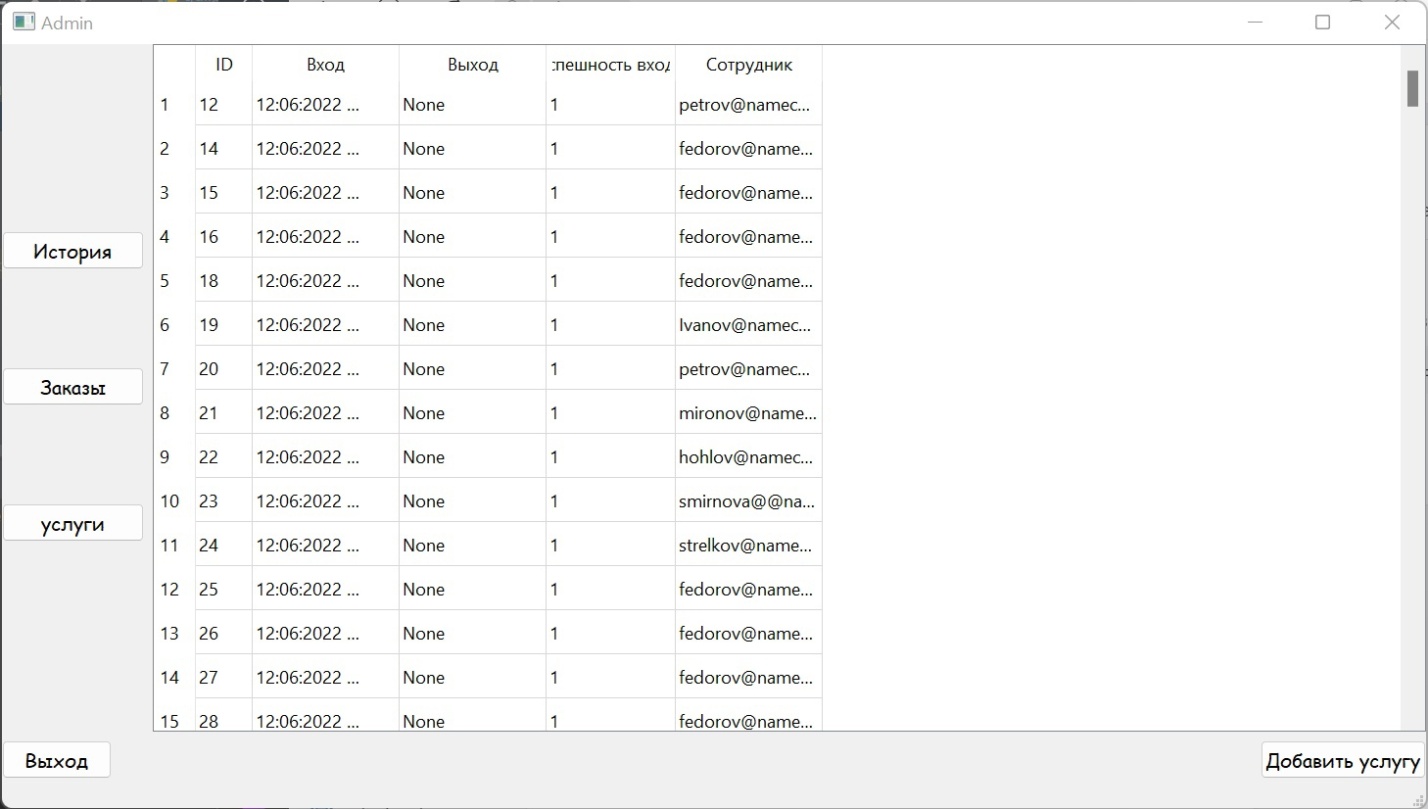
Иллюстрация формы просмотра истории входа всех сотрудников (смотреть рисунок А.4).

Рисунок А.4

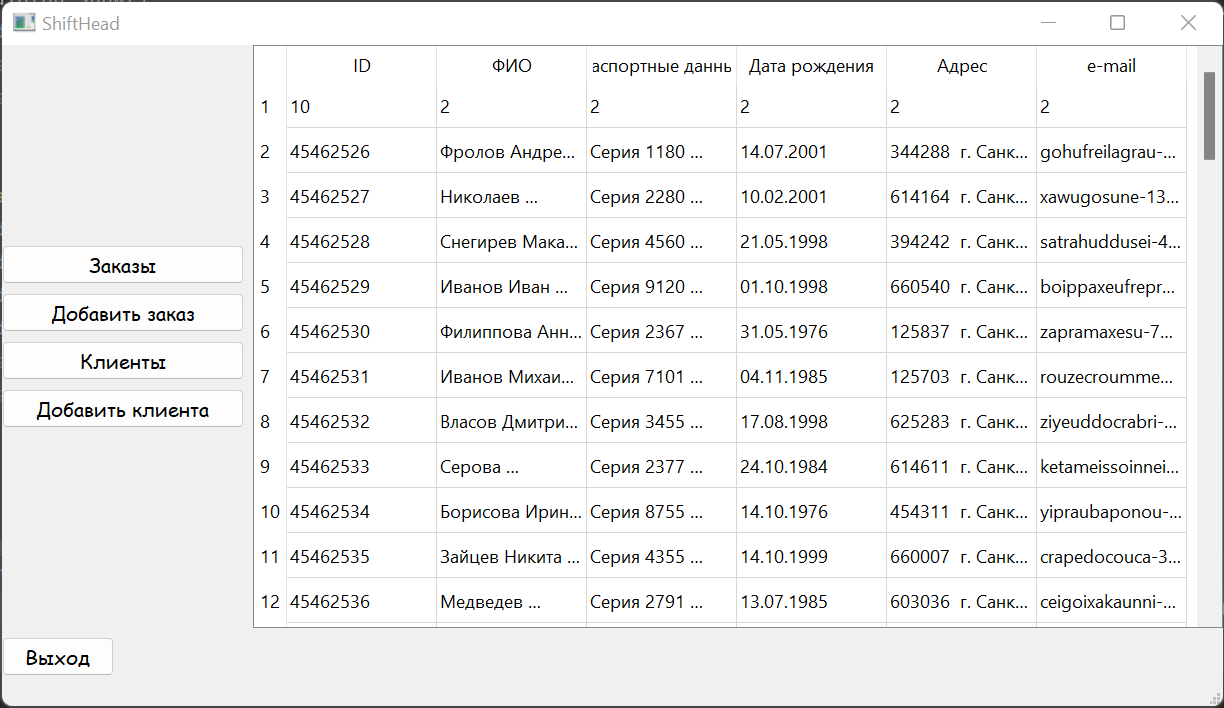
Иллюстрация формы просмотра всех клиентов (смотреть рисунок А.5).

Рисунок А.5

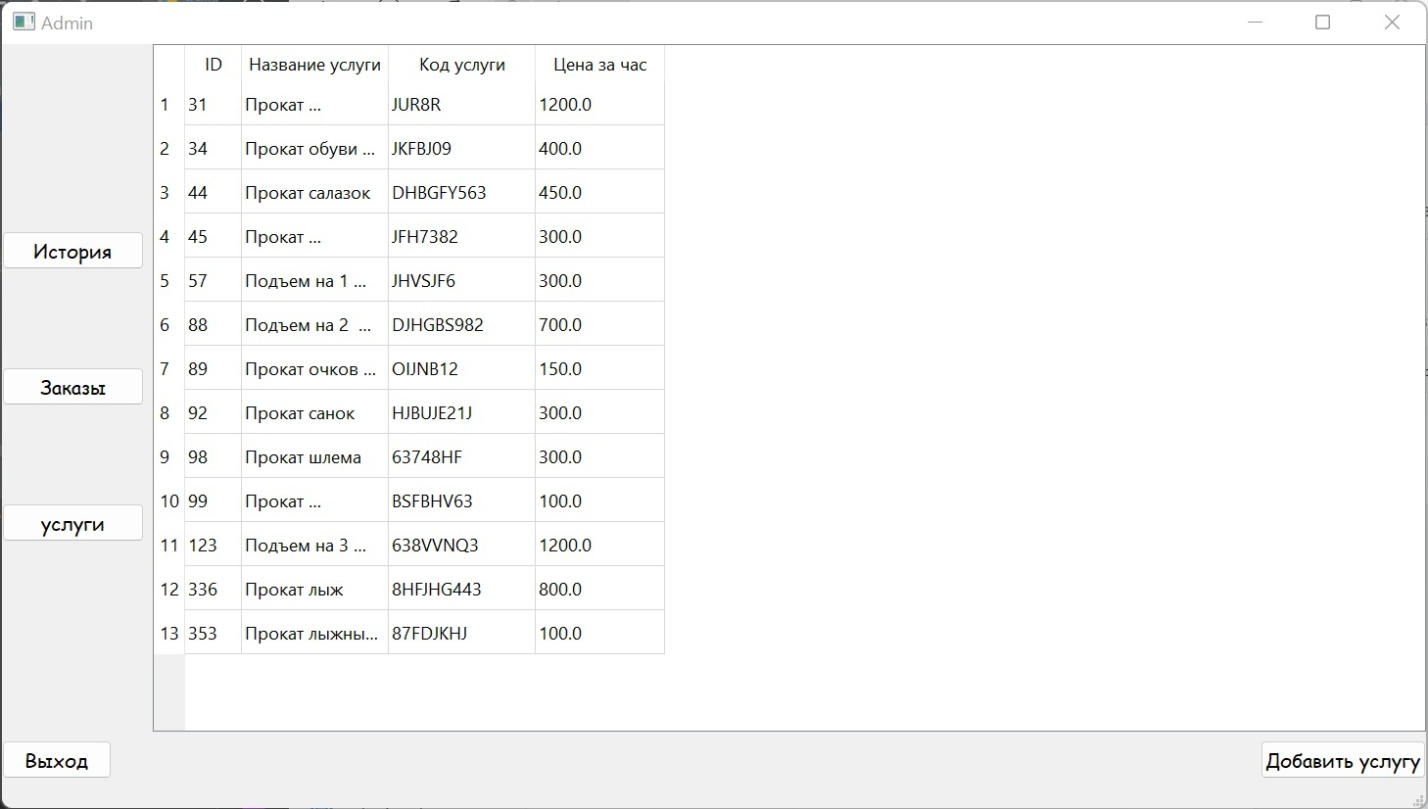
Иллюстрация формы просмотра всех доступных услуг (смотреть рисунок А.6).

Рисунок А.6

# **ПРИЛОЖЕННИЕ Б. Запросы на создание таблиц**

**Б.1 Таблица «History»**

CREATE TABLE `history` (

`his\_id` int NOT NULL AUTO\_INCREMENT,

`logDate` varchar(225) DEFAULT NULL,

`logoutdate` varchar(225) DEFAULT NULL,

`lock` varchar(45) DEFAULT NULL,

`emp\_login` varchar(225) DEFAULT NULL,

PRIMARY KEY (`his\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=295 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

**Б.2 Таблица «Orders»**

CREATE TABLE `orders` (

`ord\_id` int NOT NULL AUTO\_INCREMENT,

`order\_code` varchar(225) DEFAULT NULL,

`createDate` varchar(45) DEFAULT NULL,

`createTime` varchar(45) DEFAULT NULL,

`cl\_id` int DEFAULT NULL,

`services` varchar(225) DEFAULT NULL,

`status` varchar(225) DEFAULT NULL,

`closeTime` varchar(45) DEFAULT NULL,

`rentTime` varchar(45) DEFAULT NULL,

`emp\_id` int DEFAULT NULL,

PRIMARY KEY (`ord\_id`),

KEY `emp\_id\_idx` (`emp\_id`),

CONSTRAINT `emp\_id` FOREIGN KEY (`emp\_id`) REFERENCES `employee` (`emp\_ID`)

) ENGINE=InnoDB AUTO\_INCREMENT=74 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

**Б.3 Таблица «Services»**

CREATE TABLE `services` (

`serv\_id` int NOT NULL AUTO\_INCREMENT,

`serv\_Title` varchar(45) DEFAULT NULL,

`serv\_code` varchar(45) DEFAULT NULL,

`price` float DEFAULT NULL,

PRIMARY KEY (`serv\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=356 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

**ПРИЛОЖЕННИЕ В. Исходный код приложения**

**Gui.py**

import sys

import datetime

from main import Database

import pymysql

import random

from PyQt5 import uic, QtWidgets

from PyQt5.QtCore import Qt

from PyQt5.QtWidgets import QMainWindow, QMessageBox, QApplication, QDialog, QGraphicsScene, QTableWidgetItem

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super(MainWindow, self).\_\_init\_\_()

self.ui = uic.loadUi("forms/admin.ui", self)

self.setWindowTitle("ничто")

def authoriz(self, wnd):

dialog = DialogAutorization(wnd)

dialog.setWindowTitle("Авторизация")

dialog.show()

dialog.exec\_()

class DialogAutorization(QDialog):

def \_\_init\_\_(self, wnd, parent = None):

self.wnd = wnd

super(DialogAutorization, self).\_\_init\_\_(parent)

self.ui = uic.loadUi("forms/auth.ui", self)

self.setWindowTitle("Авторизация")

self.scene = QGraphicsScene(0, 0, 350, 50)

self.scene.clear()

self.ui.autorization\_btn.clicked.connect(self.autoriz)

self.ui.captcha\_gen.setScene(self.scene)

self.ui.reboot\_btn.clicked.connect(self.gen\_captcha)

self.ui.reboot\_btn.setEnabled(False)

self.ui.line\_cap.setEnabled(False)

self.db = Database()

self.enter\_try = 0

self.cur\_captcha = None

self.ui.line\_pas.setEchoMode(QtWidgets.QLineEdit.Password)

self.ui.hide\_btn.clicked.connect(self.hide\_pas)

self.hiden = True

def autoriz(self):

global login

global emp\_id

errorcount = 0

login = self.ui.line\_log.text()

password = self.ui.line\_pas.text()

lock = True

if self.enter\_try>1 and self.ui.line\_cap.text() != self.cur\_captcha:

print(self.ui.line\_cap.text())

if errorcount == 0:

self.error()

errorcount+=1

self.enter\_try += 1

lock = False

else:

lock = True

print(lock)

if self.enter\_try >= 2:

self.gen\_captcha()

self.ui.reboot\_btn.setEnabled(True)

self.ui.line\_cap.setEnabled(True)

if login == '' or password == '':

if errorcount == 0:

self.error()

errorcount += 1

if login not in self.db.check\_login():

if errorcount == 0:

self.error()

errorcount += 1

self.enter\_try += 1

else:

aut = self.db.get\_log(login)

autpas = aut[0]

role = aut[1]

emp\_id = aut[2]

time = datetime.datetime.now()

ent\_time = time.strftime("%d:%m:%Y %H:%M:%S")

status ="auth succes"

self.db.log\_his(login,ent\_time,status)

if password != autpas:

self.enter\_try += 1

if errorcount == 0:

self.error()

errorcount += 1

elif lock== True:

print(lock)

if role == 'Старший смены' :

self.shif\_head\_open()

if role == 'Администратор':

self.admin\_open()

if role == 'Продавец':

self.seller\_open()

errorcount = 0

def hide\_pas(self):

if self.hiden == True:

self.hiden = False

self.ui.line\_pas.setEchoMode(QtWidgets.QLineEdit.Normal)

else:

self.hiden = True

self.ui.line\_pas.setEchoMode(QtWidgets.QLineEdit.Password)

def gen\_captcha(self):

self.scene.clear()

symb = 'qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM1234567890'

s\_count = 5

cur\_symb = [1, 2, 3, 4, 5]

x, y = 30, 20

self.scene.addLine(20, random.randint(10, 40), 300, random.randint(10, 40))

for i in range(s\_count):

cur\_symb[i] = symb[random.randint(0, 61)]

text = self.scene.addText(f"{cur\_symb[i]}")

x += 40

text.moveBy(x, y + random.randint(–10, 10))

self.cur\_captcha = ''.join(cur\_symb)

print(self.cur\_captcha)

def error(self):

self.mesbox = QMessageBox(self)

self.mesbox.setWindowTitle("Ошибка")

self.mesbox.setText("Ошибка входа")

self.mesbox.setStandardButtons(QMessageBox.Ok)

self.mesbox.show()

def shif\_head\_open(self):

self.ui.close()

self.ui = ShiftHeadMenu()

self.ui.show()

def admin\_open(self):

self.ui.close()

self.ui = AdminMenu()

self.ui.show()

def seller\_open(self):

self.ui.close()

self.ui = SellerMenu()

self.ui.show()

class ShiftHeadMenu(QMainWindow):

def \_\_init\_\_(self):

super(ShiftHeadMenu, self).\_\_init\_\_()

self.ui = uic.loadUi("forms/shift\_head.ui", self)

self.window().setWindowTitle("ShiftHead")

self.ui.back\_btn.clicked.connect(self.exit)

self.table = self.ui.tableWidget

self.db = Database()

self.ui.add\_order\_btn.clicked.connect(self.add\_order)

self.ui.OrdersTable.clicked.connect(self.orders)

self.ui.clientTable.clicked.connect(self.clients)

self.ui.add\_client\_btn.clicked.connect(self.add\_client)

def orders(self):

self.table.clear()

out = self.db.get\_ord()

self.table.setColumnCount(9) # кол–во столбцов

self.table.setRowCount(len(out)) # кол–во строк

self.table.setHorizontalHeaderLabels(

['ID', 'код заказа', 'дата создания', 'Время заказа', 'Код клиента', 'Код услуги', 'статус',

'дата закрытия', 'время аренды'])

for i, order in enumerate(out):

for x, field in enumerate(order): # i, x – координаты ячейки, в которую будем записывать текст

item = QTableWidgetItem()

item.setText(str(field)) # записываем текст в ячейку

item.setFlags(Qt.ItemIsEnabled)

self.table.setItem(i, x, item)

def add\_order(self):

dialog = DialogAdd()

dialog.setWindowTitle("Добавить заказ")

dialog.show()

dialog.exec\_()

self.table.clear()

self.orders()

def clients(self):

self.table.clear()

out = self.db.get\_clnt()

self.table.setColumnCount(6) # кол–во столбцов

self.table.setRowCount(len(out)) # кол–во строк

self.table.setHorizontalHeaderLabels(

['ID', 'ФИО', 'Паспортные данные', 'Дата рождения', 'Адрес', 'e–mail'])

for i, order in enumerate(out):

for x, field in enumerate(order): # i, x – координаты ячейки, в которую будем записывать текст

item = QTableWidgetItem()

item.setText(str(field)) # записываем текст в ячейку

item.setFlags(Qt.ItemIsEnabled)

self.table.setItem(i, x, item)

def add\_client(self):

dialog = DialogAddClient()

dialog.setWindowTitle("Добавить клиента")

dialog.show()

dialog.exec\_()

self.table.clear()

self.clients()

def exit(self):

dialog = DialogAutorization(self.window)

self.ui.close()

dialog.setWindowTitle("Авторизация")

dialog.show()

dialog.exec\_()

class AdminMenu(QMainWindow):

def \_\_init\_\_(self):

super(AdminMenu, self).\_\_init\_\_()

self.ui = uic.loadUi("forms/admin.ui", self)

self.window().setWindowTitle("Admin")

self.db = Database()

self.ui.orders\_btn.clicked.connect(self.orders)

self.ui.expenses\_btn.clicked.connect(self.services)

self.ui.history\_btn.clicked.connect(self.history)

self.ui.add\_serv.clicked.connect(self.add\_service)

self.ui.back\_btn.clicked.connect(self.exit)

self.table = self.ui.tableWidget

def orders(self):

self.table.clear()

out = self.db.get\_ord()

self.table.setColumnCount(9) # кол–во столбцов

self.table.setRowCount(len(out)) # кол–во строк

self.table.setHorizontalHeaderLabels(['ID', 'код заказа', 'дата создания','Время заказа','Код клиента','Код услуги','статус', 'дата закрытия','время аренды'])

for i, order in enumerate(out):

for x, field in enumerate(order): # i, x – координаты ячейки, в которую будем записывать текст

item = QTableWidgetItem()

item.setText(str(field)) # записываем текст в ячейку

if x == 0: # для id делаем некликабельные ячейки

item.setFlags(Qt.ItemIsEnabled)

self.table.setItem(i, x, item)

def services(self):

self.table.clear()

out = self.db.get\_serv()

self.table.setColumnCount(4) # кол–во столбцов

self.table.setRowCount(len(out)) # кол–во строк

self.table.setHorizontalHeaderLabels(

['ID', 'Название услуги', 'Код услуги', 'Цена за час'])

for i, order in enumerate(out):

for x, field in enumerate(order): # i, x – координаты ячейки, в которую будем записывать текст

item = QTableWidgetItem()

item.setText(str(field)) # записываем текст в ячейку

if x == 0: # для id делаем некликабельные ячейки

item.setFlags(Qt.ItemIsEnabled)

self.table.setItem(i, x, item)

def history(self):

self.table.clear()

out = self.db.get\_his()

self.table.setColumnCount(5) # кол–во столбцов

self.table.setRowCount(len(out)) # кол–во строк

self.table.setHorizontalHeaderLabels(

['ID', 'Вход', 'Выход', 'Успешность входа', 'Сотрудник'])

for i, order in enumerate(out):

for x, field in enumerate(order): # i, x – координаты ячейки, в которую будем записывать текст

item = QTableWidgetItem()

item.setText(str(field)) # записываем текст в ячейку

item.setFlags(Qt.ItemIsEnabled)

self.table.setItem(i, x, item)

def add\_service(self):

dialog = DialogAddServ()

dialog.setWindowTitle("Добавить услугу")

dialog.show()

dialog.exec\_()

self.table.clear()

self.orders()

def exit(self):

time = datetime.datetime.now()

ent\_time = time.strftime("%d:%m:%Y %H:%M:%S")

status = "session ended"

self.db.logout\_his(login, ent\_time, status)

dialog = DialogAutorization(self.window)

self.ui.close()

dialog.setWindowTitle("Авторизация")

dialog.show()

dialog.exec\_()

class SellerMenu(QMainWindow):

def \_\_init\_\_(self):

super(SellerMenu, self).\_\_init\_\_()

self.ui = uic.loadUi("forms/seller.ui", self)

self.window().setWindowTitle("Seller")

self.table = self.ui.order\_table

self.db = Database()

self.ui.order\_add\_btn.clicked.connect(self.add\_order)

self.table = self.ui.order\_table

self.ui.back\_btn.clicked.connect(self.exit)

self.ui.add\_clnt.clicked.connect(self.add\_client)

self.orders()

def exit(self):

dialog = DialogAutorization(self.window)

self.ui.close()

dialog.setWindowTitle("Авторизация")

dialog.show()

dialog.exec\_()

def orders(self):

self.table.clear()

out = self.db.get\_ord()

self.table.setColumnCount(9) # кол–во столбцов

self.table.setRowCount(len(out)) # кол–во строк

self.table.setHorizontalHeaderLabels(['ID', 'код заказа', 'дата создания','Время заказа','Код клиента','Код услуги','статус', 'дата закрытия','время аренды'])

for i, order in enumerate(out):

for x, field in enumerate(order): # i, x – координаты ячейки, в которую будем записывать текст

item = QTableWidgetItem()

item.setText(str(field)) # записываем текст в ячейку

item.setFlags(Qt.ItemIsEnabled)

self.table.setItem(i, x, item)

def add\_order(self):

dialog = DialogAdd()

dialog.setWindowTitle("Добавить заказ")

dialog.show()

dialog.exec\_()

self.table.clear()

self.orders()

def add\_client(self):

dialog = DialogAddClient()

dialog.setWindowTitle("Добавить клиента")

dialog.show()

dialog.exec\_()

self.table.clear()

self.orders()

class DialogAdd(QDialog):

def \_\_init\_\_(self):

super(DialogAdd, self).\_\_init\_\_()

self.ui = uic.loadUi("forms/add\_order.ui", self)

self.setWindowTitle("Добавить")

self.db = Database()

self.ui.add\_btn\_2.clicked.connect(self.add)

self.build\_combobox\_clients()

self.build\_combobox\_serv()

def add(self):

self.datetime = datetime.datetime.now()

self.client = self.comboClient.currentText()

self.client\_code = str(self.db.get\_clnt\_code(self.client))

self.date = str(datetime.date.today())

self.code = str(self.client\_code) + "/" + str(self.date)

self.service = self.comboServ.currentText()

self.service\_code = self.db.get\_serv\_code(self.service)

self.time= str(self.datetime.strftime("%H:%M"))

self.status = self.ui.order\_status.text()

self.close\_date= self.ui.end\_date.text()

self.rent\_time=self.ui.use\_time.text()

self.db.add\_ord(self.code, self.date, self.time, self.client\_code, self.service\_code, self.status, self.close\_date,self.rent\_time, emp\_id)

self.ui.close()

def build\_combobox\_clients(self):

"""

Добавление списка клиентов в ComboBox

:return:

"""

clients = self.db.get\_clnt\_name()

self.comboClient.clear()

if self.comboClient is not None:

self.comboClient.addItems(clients)

def build\_combobox\_serv(self):

"""

Добавление списка услуг в ComboBox

:return:

"""

services = self.db.get\_serv\_title()

self.comboServ.clear()

if self.comboServ is not None:

self.comboServ.addItems(services)

class DialogAddClient(QDialog):

def \_\_init\_\_(self):

super(DialogAddClient, self).\_\_init\_\_()

self.ui = uic.loadUi("forms/add\_client.ui", self)

self.setWindowTitle("Добавить")

self.db = Database()

self.ui.add\_btn\_2.clicked.connect(self.add\_cln)

def add\_cln(self):

self.name = self.ui.name\_line.text()

self.p\_data = self.ui.passport\_line.text()

self.birth = self.birth\_line.text()

self.addres = self.addres\_line.text()

self.mail = self.ui.email\_line.text()

self.db.add\_clnt(self.name, self.p\_data, self.birth, self.addres, self.mail)

class DialogAddServ(QDialog):

def \_\_init\_\_(self):

super(DialogAddServ, self).\_\_init\_\_()

self.ui = uic.loadUi("forms/add\_serv.ui", self)

self.setWindowTitle("Добавить")

self.db = Database()

self.ui.add\_btn\_2.clicked.connect(self.add)

def add(self):

self.title = self.ui.serv\_title.text()

self.code = self.ui.serv\_code.text()

self.price = self.ui.serv\_price.text()

self.db.add\_serv(self.title, self.code, self.price)

self.ui.close()

class Builder:

def \_\_init\_\_(self):

self.app = QApplication(sys.argv)

self.wnd = MainWindow()

self.auth()

def auth(self):

self.wnd.authoriz(self.wnd)

self.app.exec()

if \_\_name\_\_ == '\_\_main\_\_':

B = Builder()

**Main.py**

import pymysql

import datetime

import mysql

from mysql.connector import connect, Error

class Database:

def \_\_init\_\_(self):

self.connection = pymysql.connect(

host='localhost',

port=3306,

user='root',

password='root',

database='igora',

)

def get\_his(self):

cursor = self.connection.cursor()

cursor.execute('select \* from History')

his = cursor.fetchall()

cursor.close()

return (his)

def log\_his(self, login, time, status):

cursor = self.connection.cursor()

cursor.execute(f"INSERT INTO history VALUES (NULL, %s, NULL, %s, %s)", (time, status, login))

self.connection.commit()

def logout\_his(self, login, time, status):

cursor = self.connection.cursor()

cursor.execute(f"INSERT INTO history VALUES (NULL, NULL, %s, %s, %s)", (time, status, login))

self.connection.commit()

def get\_emp(self):

cursor = self.connection.cursor()

cursor.execute('select \* from employee')

emps = cursor.fetchall()

cursor.close()

return (emps)

def get\_clnt(self):

cursor = self.connection.cursor()

cursor.execute('select \* from client')

clnt = cursor.fetchall()

cursor.close()

return clnt

def get\_clnt\_name(self):

clients = []

cursor = self.connection.cursor()

cursor.execute(f"select cl\_name from client")

clnt = cursor.fetchall()

cursor.close()

for i in clnt:

clients.append(str(i)[2:–3])

return clients

def get\_clnt\_code(self, name):

cursor = self.connection.cursor()

cursor.execute(f"SELECT `cl\_Id` FROM client WHERE cl\_name='{name}'")

clnt = cursor.fetchone()

cursor.close()

return clnt[0]

def add\_clnt(self, name, data, birth, addres, mail):

cursor = self.connection.cursor()

cursor.execute(f"INSERT INTO client VALUES (NULL, %s, %s, %s, %s, %s, NULL)", (name, data, birth, addres, mail))

cursor.close()

self.connection.commit()

def get\_ord(self):

cursor = self.connection.cursor()

cursor.execute('select \* from orders')

ord = cursor.fetchall()

cursor.close()

return (ord)

def add\_ord(self, code, date, time, client, serv, status, close\_time, rent\_time, emp\_id):

cursor = self.connection.cursor()

cursor.execute(f"INSERT INTO orders VALUES (NULL, %s, %s, %s, %s, %s, %s, %s, %s, %s)",

(code, date, time, client, serv, status, close\_time, rent\_time, emp\_id))

cursor.close()

self.connection.commit()

def get\_serv(self):

cursor = self.connection.cursor()

cursor.execute('select \* from services')

serv = cursor.fetchall()

cursor.close()

return serv

def get\_serv\_title(self):

services = []

cursor = self.connection.cursor()

cursor.execute(f"select serv\_Title from services")

serv = cursor.fetchall()

cursor.close()

for i in serv:

services.append(str(i)[2:–3])

return services

def get\_serv\_code(self, title):

cursor = self.connection.cursor()

cursor.execute(f"select serv\_id from services WHERE serv\_Title='{title}'")

serv = cursor.fetchone()

cursor.close()

return serv[0]

def add\_serv(self, title, code, price):

cursor = self.connection.cursor()

cursor.execute(f"INSERT INTO services VALUES (NULL, %s, %s, %s)", (title, code, price))

cursor.close()

self.connection.commit()

def check\_login(self):

log = []

cursor = self.connection.cursor()

cursor.execute(f"""SELECT login FROM Employee""")

rows = cursor.fetchall()

for i in rows:

for j in i:

log.append(j)

return log

def get\_log(self, login):

log = []

cursor = self.connection.cursor()

cursor.execute(f"""SELECT password, Position, emp\_ID FROM employee WHERE login = '{login}'""")

rows = cursor.fetchall()

for i in rows:

for j in i:

log.append(j)

return log

if \_\_name\_\_ == '\_\_main\_\_':

db = Database()

print(db.get\_clnt\_name())