# Введение

Тема курсовой работы – проектирование реляционной базы данных информационной системы бюро ритуальных услуг.

В соответствии с предметной областью работы, будущая база данных должна хранить сведения о заказах, клиентах, сотрудниках, товарах, услугах, акциях и обеспечивать целостность этих данных.

Для обеспечения функционала, а также для удобства пользования информационной системой необходимо разработать приложение, которое позволит добавлять, удалять, редактировать заказы, услуги, клиентов, выводить информацию о товарах и акциях. Приложение должно быть простым в использовании, которым могли бы пользоваться даже неквалифицированные сотрудники.

Актуальность данной работы определяется тем, что во все времена сфера ритуальных услуг пользовалась спросом, что привело к появлению большого количества соответствующих организаций. В таких организациях возможно использование данного программного продукта. Он во много раз облегчит управление этим видом бизнеса.

Аналогов данной программы немало и не только в данной области. Они обычно разрабатываются под конкретную организацию, занимающуюся определенным видом деятельности. Создаваемая информационная система бюро ритуальных услуг не привязывается к какому-либо существующему бюро, и создается на основе некого абстрактного офиса, предоставляющие клиентам различные услуги по захоронению и т.д.

Для создания информационной базы данных будет использоваться СУДБ MySQL. Для создания приложения – среда Intelj IDEA.

# Построение инфологической концептуальной модели

## Анализ предметной области и выявление необходимого набора сущностей

В ходе анализа знаний и разработке базы данных были выявлены следующие основные сущности:

Сущность Заказ описывает заказы, поступившие в бюро от клиентов, их типы и параметры. Характеризуется заказчиком, заказанными услугами и товарами, датой заказа, датой оплаты и сотрудником.

Сущность Услуга представляет собой список всех выполненных и выполняемых услуг для клиентов. Характеризуется типом услуги.

Сущность Клиент описывает всех клиентов, сделавшие заказы или пользующиеся какими-либо услугами. Характеризуется типом клиента (физ. лицо или юр. лицо).

Сущность Физ. лицо описывает клиентов являющихся физическим лицом и характеризуется фамилией, именем, отчеством и паспортными данными.

Сущность Юр. лицо описывает клиентов являющихся юредическим лицом и характеризуется адресом и лицевым счетом.

Сущность Товар описывает все Товары бюро ритуальных услуг. Характеризуется названием и стоимостью.

Сущность Сотрудник описывает сотрудников бюро. Характеризуется именем и должностью.

Сущность Акция описывает акции. Характеризуется услугой скидкой и датой акции.

## Обоснование требуемого набора атрибутов для каждой сущности и выделение идентифицирующих атрибутов

Для построения инфологической концептуальной модели необходимо для каждой сущности, выявленной в предыдущем пункте, определить требуемый набор атрибутов. Атрибутом является поименованная характеристика сущности. Его наименование должно быть уникальным для конкретного типа сущности, но может быть одинаковым для различного типа сущностей. Атрибуты используются для определения того, какая информация должна быть собрана о сущности.

Ниже представлены сущности и определенные для них атрибуты, а также ключи (подчеркнуты). Имена сущностей и атрибутов указываются, как они будут определены в созданной базе данных, в скобках указывается перевод либо описание:

1. orders (Заказы):

* order\_id (код заказа);
* client\_id (код клиента-заказчика);
* employer\_id (код сотрудника);
* order\_date (дата заказа);
* done\_date (дата выполнения);

1. services (Услуги):

* service\_id (код услуги);
* service\_type\_name (название типа услуги);
* service\_type\_cost (стоимость данного типа услуги);

1. services\_orders\_mtm (Связь многие ко многим для услуг и заказов):

* service\_id (код услуги);
* order\_id (код заказа);
* date\_time (дата/время);

1. goods (Товары):

* goods\_id (код товара);
* goods\_type\_name (название типа товара);
* goods\_type\_cost (стоимость данного товара);

1. goods\_orders\_mtm (Связь многие ко многим для услуг и заказов):

* goods\_id (код услуги);
* order\_id (код заказа);
* count (количество);

1. clients (Клиенты):

* client\_id (код клиента);
* type (тип);

1. phisical (Физ. Лица):

* name (имя);
* surname (фамилия);
* midlename (отчество);
* pasport (паспортные данные);

1. uridical (Юр. Лица):

* name (название);
* address (адрес);
* cash (лицевой счет);

1. employer (Сотрудники):

* employer\_id (код киоска);
* name (имя);

1. actions (Акции):

* action\_id (код филиала);
* type (тип услуги);
* percent (скидка);
* from\_date (с даты)
* to\_date (по дату);

## Определение связей между объектами

Кроме атрибутов каждой сущности модель данных должна определять связи между сущностями. На концептуальном уровне связи представляют собой простые ассоциации между сущностями.

Связь – это ассоциирование двух или более сущностей. Если бы назначением базы данных было только хранение отдельных, не связанных между собой данных, то ее структура могла бы быть очень простой. Однако, одно из основных требований к организации базы данных – это обеспечение возможности отыскания одних сущностей по значениям других, для чего необходимо установить между ними определенные связи. А так как в реальных базах данных нередко содержатся десятки или даже сотни сущностей, то между ними может быть установлено великое множество связей. Наличие такого множества связей и определяет сложность инфологических моделей.

Для реализации информационной системы бюро ритуальных услуг необходимо установить все связи между объектами. А именно, нужно рассмотреть всю информационную систему бюро в совокупности и определить взаимное влияние объектов, составляющих систему.

Этот процесс изображен на рис. 1.1

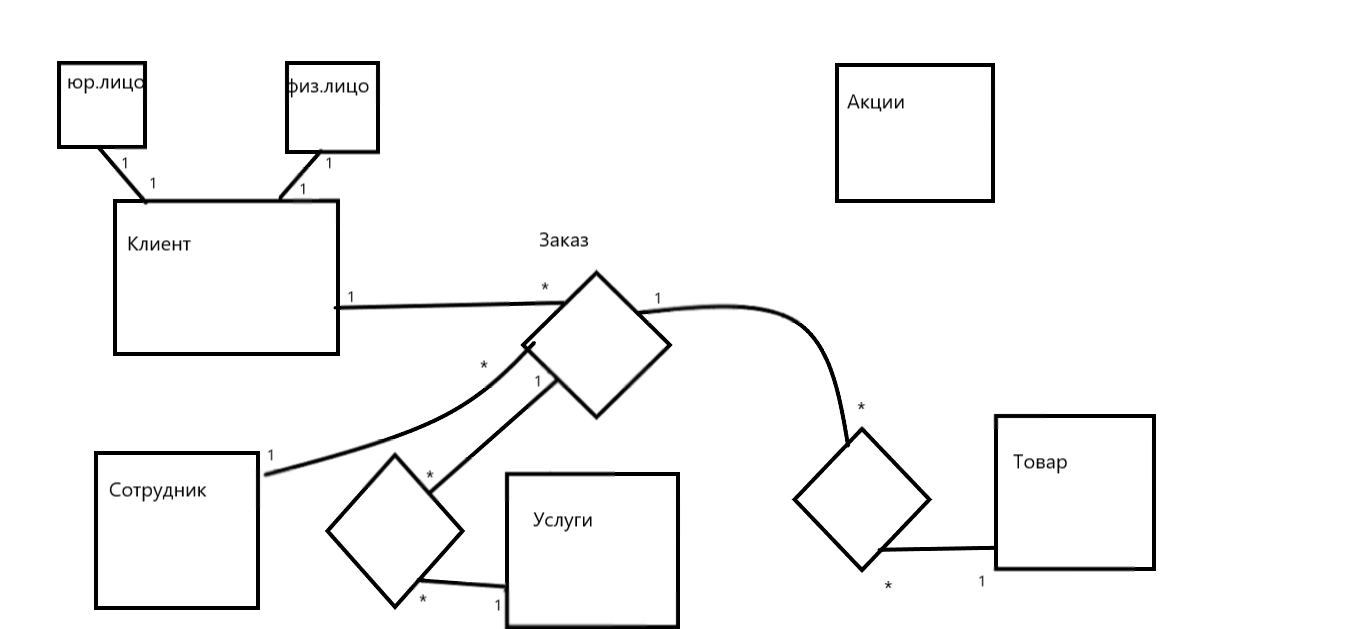


Рисунок 1.1 – Концептуальная схема базы данных

## Описание полученной модели на языке инфологического проектирования

Проектирование инфологической модели предметной области – частично формализованное описание объектов предметной области в терминах некоторой семантической модели, например, в терминах ER-модели (*англ.* entity-relationship model). По правилам построения ER-диаграмм в нотации Питера Чена, сущности изображаются прямоугольниками, их атрибуты – овалами, отношения – ромбами. Связи между объектами изображаются линиями [1].

На основе проведенного проектирования, получим ER-диаграмму базы данных фотоцентра, представленную в Приложении A.

# Построение схемы реляционной базы данных

## Построение набора необходимых отношений базы данных

Чтобы построить схему реляционной базы данных необходимо определить совокупность отношений, которые составляют базу данных. Эта совокупность отношений будет содержать всю информацию, которая должна храниться в базе данных.

В предыдущем пункте мы создали инфологическую концептуальную модель базы данных бюро ритуальных услуг, построенной с помощью языка «Таблицы-связи». На основе полученной концептуальной модели можно определить набор необходимых отношений базы данных. На рисунке 2.1 представлены отношения для базы данных бюро.

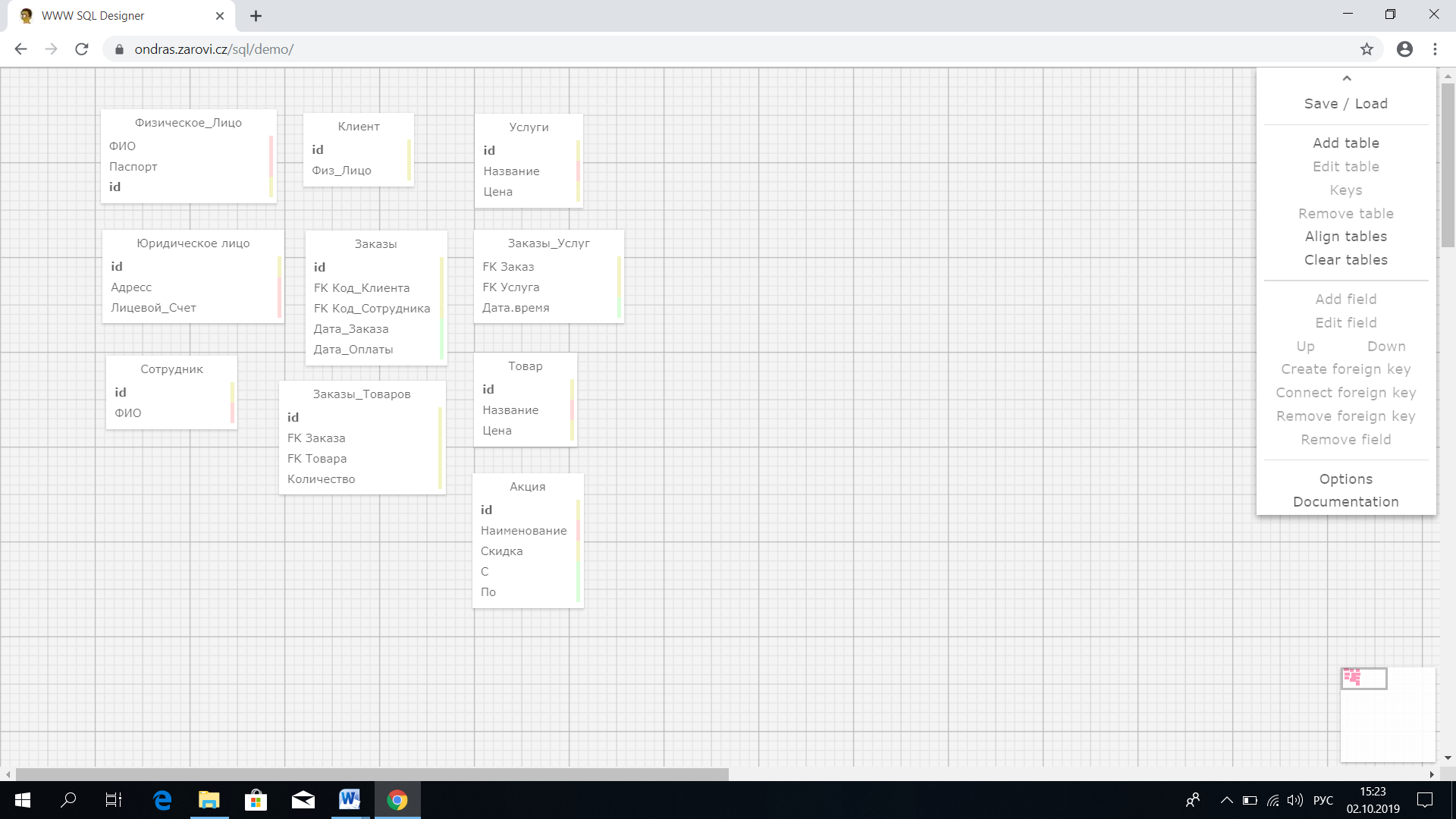


Рисунок 2.1 – Набор необходимых отношений базы данных

## Задание первичных и внешних ключей определенных отношений

В реляционной базе данных каждому объекту и сущности реального мира соответствуют кортежи отношений. И любое отношение должно обладать первичным ключом. Ключ – это минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности. Минимальность означает, что исключение из набора любого атрибута не позволяет идентифицировать сущность по оставшимся. Каждое отношение должно обладать хотя бы одним ключом. В таблице 2.1 определены первичные и внешние ключи для отношений.

Таблица 2.1 – Первичные и внешние ключи отношений.

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Название таблицы | Первичный ключ | Внешние ключи |
|  | сотрудники | id |  |
|  | услуги | id |  |
|  | клиенты | id |  |
|  | товары | id |  |
|  | акции | id |  |
|  | заказы | id | код\_клиента  код\_сотрудника |
|  | заказы\_услуг |  | код\_заказа  код\_услуги |
|  | заказы\_товаров | id | код\_заказа  код\_товара |
|  | Физическое\_лицо |  | id |
|  | Юридическое\_лицо |  | id |

В дальнейшем построении схемы реляционной базы данных ключи будут служить для организации связей между отношениями.

Таким образом, ненормализованная схема базы данных представлена в приложении B на рисунке B.1.

## Третья нормальная форма

Процесс преобразования базы данных к виду, отвечающему нормальным формам, называется нормализацией. Нормализация предназначена для приведения структуры базы данных к виду, обеспечивающему минимальную избыточность, то есть нормализация не имеет целью уменьшение или увеличение производительности работы или же уменьшение или увеличение объёма БД. Конечной целью нормализации является уменьшение потенциальной противоречивости хранимой в БД информации.

Для реляционных баз данных необходимо, чтобы все отношения базы данных обязательно находились в 1НФ. Нормальные формы более высокого порядка могут использоваться разработчиками по своему усмотрению. Однако грамотный специалист стремится к тому, чтобы довести уровень нормализации базы данных хотя бы до 3НФ, тем самым, исключив из базы данных избыточность и аномалии обновления.

Определение 3НФ –  каждый неключевой атрибут «должен предоставлять информацию о ключе, полном ключе и ни о чём, кроме ключа». Определение ограничений целостности для внешних ключей отношений и для отношений в целом

Ограничение целостности отношений заключается в том, что в любом отношении должны отсутствовать записи с одним и тем же значением первичного ключа. Конкретно требование состоит в том, что любая запись любого отношения должна быть отличной от любой другой записи этого отношения. Это требование автоматически удовлетворяется, если в системе не нарушаются базовые свойства отношений.

У проектируемой базы следующие таблицы будут иметь первичные ключи: сотрудники, услуги, клиенты, товары, акции, заказы, заказы\_товаров. Все связи имеют мощность 1-\* кроме связей Физическое\_лицо и Юридическое\_лицо. Они имеют мощность 1-1.

## Графическое представление связей между внешними и первичными ключами

По результатам нормализации, определении первичных и внешних ключей, связей между сущностями, была получена схема реляционной базы данных, представленная в приложении B на рисунке B.2. На ней изображаются все отношения базы данных, а также связей между внешними и первичными ключами. Ключи обозначаются жирным выделением, внешние ключи обозначаются FK (*от англ.* Foreign Key – внешний ключ).

# 

Рисунок 2.2 – Схема реляционной базы данных

# Создание спроектированной базы данных

Для реализации спроектированной базы данных была выбрана система управления базами данных MySQL 5. Это обусловлено тем, что, во-первых, MySQL довольно широко распространена, во-вторых, является свободно распространяемой, и, в-третьих, MySQL поддерживается большинством применяемых средств доступа к данным.

В создаваемой базе данных будут использоваться следующие типы данных [3]:

1. INTEGER – Целочисленный тип. Размер – 4 байта
2. VARCHAR – Строковый тип
3. DECIMAL – Вещественный тип
4. TINYINT – Целочисленный тип размером в 1 байт. Определяется как логический тип.
5. DATE – Тип, определяющий дату.
6. DATETIME – Тип, определяющий дату и время.

Опишем все таблицы, которые будут созданы в базе данных.

Таблица клиенты содержит перечень всех клиентов бюро ритуальных услуг. Ее структура приведена в таблице 3.1.

Таблица 3.1 – Характеристика атрибутов таблицы клиенты.

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id | INT(11) | Идентификатор клиента. Ключевой атрибут |
| Физ\_Лицо | TINYINT(1) | Определяет, является ли клиент физическим лицом |

Таблица физическое\_лицо содержит клиентов являющихся физ. лицами. Ее структура приведена в таблице 3.2.

Таблица 3.2 – Характеристика атрибутов таблицы физическое\_лицо.

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id | INT(11) | Идентификатор клиента. Ключевой атрибут |
| ФИО | VARCHAR(255) | Фамилия Имя Отчество |
| Паспорт | VARCHAR(255) | Серийный номер паспорта |

Таблица юридическое\_лицо содержит клиентов являющихся юр. лицами. Ее структура приведена в таблице 3.3.

Таблица 3.3 – Характеристика атрибутов таблицы юридическое\_лицо.

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id | INT(11) | Идентификатор клиента. Ключевой атрибут |
| Адрес | VARCHAR(255) | Адрес компании |
| Лицевой счет | VARCHAR(255) | Лицевой счет клиенты |

Таблица заказы содержит заказы клиентов. Ее структура приведена в таблице 3.4.

Таблица 3.4 – Характеристика атрибутов таблицы заказы.

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id | INT(11) | Идентификатор заказов. |
| Код\_Клиента | INT(11) | Идентификатор таблицы клиентов. |
| Код\_Сотрудника | INT(11) | Идентификатор таблицы сотрудников. |
| Дата\_Заказа | DATE | Дата заказа |
| Дата\_Оплаты | DATE | Дата оплаты |

Таблица сотрудники содержит всех сотрудников. Ее структура приведена в таблице 3.5.

Таблица 3.5 – Характеристика атрибутов таблицы сотрудники.

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id | INT(11) | Идентификатор сотрудника. Ключевой атрибут |
| ФИО | VARCHAR(255) | ФИО сотрудника |

Таблица услуги содержит все типы услуг. Ее структура приведена в таблице 3.6.

Таблица 3.6 – Характеристика атрибутов таблицы услуги.

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id | INT(11) | Идентификатор услуги. Ключевой атрибут |
| Название | VARCHAR(50) | Название услуги |
| Цена | INT(11) | Цена услуги |

Таблица товары содержит перечень всех товаров. Ее структура приведена в таблице 3.7.

Таблица 3.7 – Характеристика атрибутов таблицы товары.

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id | INT(11) | Идентификатор товара. Ключевой атрибут |
| Название | VARCHAR(50) | Название товара |
| Цена | INT(11) | Цена товара |

Таблица заказы\_услуг содержит все заказы на услуги. Ее структура приведена в таблице 3.8.

Таблица 3.8 – Характеристика атрибутов таблицы заказы\_услуг.

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| Код\_Заказа | INT(11) | Идентификатор заказа |
| Код\_Услуги | INT(11) | Идентификатор услуги |
| Дата\_Время | DATETIME | Дата и время выполнения услуги |

Таблица заказы\_товаров содержит все заказы на товары. Ее структура приведена в таблице 3.9.

Таблица 3.9 – Характеристика атрибутов таблицы заказы\_товаров.

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id | INT(11) | Идентификатор заказа товара. Ключевой атрибут |
| Код\_Заказа | INT(11) | Идентификатор заказа |
| Код\_Товара | INT(11) | Идентификатор товара |
| Количество | INT(11) | Количество товара |

Таблица акции содержит перечень всех акций. Ее структура приведена в таблице 3.10.

Таблица 3.10 – Характеристика атрибутов таблицы акции.

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| id | INT(11) | Идентификатор акции. Ключевой атрибут |
| Наименование | VARCHAR(50) | Название товара или услуги |
| Скидка | DECIMAL | Скидка |
| С | DATE | Дата начала акции |
| По | DATE | Дата конца |

# Запись выражений указанных в варианте задания типов запросов на языке SQL

Метод, выполняющий выборку клиентов по различным параметрам:

public void onFilterMethod(){

Dialog<PhysicalPerson> dialog = new Dialog<>();

dialog.setTitle("Добавить");

dialog.setHeaderText(null);

// Set the icon (must be included in the project).

// Set the button types.

ButtonType addButtonType = new ButtonType("Add", ButtonBar.ButtonData.OK\_DONE);

dialog.getDialogPane().getButtonTypes().addAll(addButtonType, ButtonType.CANCEL);

// Create the name and password labels and fields.

GridPane grid = new GridPane();

grid.setHgap(10);

grid.setVgap(10);

grid.setPadding(new Insets(20, 50, 10, 10));

/\* ListView<Goods> goods = new ListView<>(ThirdPage.getUsersData());

goods.getSelectionModel().selectFirst();\*/

//grid.add(new Label("Наименование:"), 0, 0);

//grid.add(goods, 1, 0);

LimitedTextField surname = new LimitedTextField(32);

surname.setText("A");

LimitedTextField surname2 = new LimitedTextField(32);

surname2.setText("я");

grid.add(new Label("Фамилия:"), 0, 0);

grid.add(surname, 1, 0);

grid.add(surname2, 2, 0);

LimitedTextField name = new LimitedTextField(32);

name.setText("A");

LimitedTextField name2 = new LimitedTextField(32);

name2.setText("я");

grid.add(new Label("Имя:"), 0, 1);

grid.add(name, 1, 1);

grid.add(name2, 2, 1);

LimitedTextField lastname = new LimitedTextField(32);

lastname.setText("A");

LimitedTextField lastname2 = new LimitedTextField(32);

lastname2.setText("я");

grid.add(new Label("Отчество:"), 0, 2);

grid.add(lastname, 1, 2);

grid.add(lastname2, 2, 2);

LimitedTextField passport = new LimitedTextField(10);

passport.setText("A");

LimitedTextField passport2 = new LimitedTextField(10);

passport2.setText("я");

grid.add(new Label("Паспорт:"), 0, 3);

grid.add(passport, 1, 3);

grid.add(passport2, 2, 3);

/\*DatePicker order = new DatePicker();

order.setValue(LocalDate.now());

order.setShowWeekNumbers(true);

grid.add(new Label("Дата заказа:"), 0, 2);

grid.add(order, 1, 2);\*/

// Enable/Disable login button depending on whether a name was entered.

Node addButton = dialog.getDialogPane().lookupButton(addButtonType);

addButton.setDisable(false);

// Do some validation (using the Java 8 lambda syntax).

/\*surname.textProperty().addListener((observable, oldValue, newValue) -> {

addButton.setDisable(newValue.trim().isEmpty() || name.getText().isEmpty() || passport.getText().isEmpty());

});

name.textProperty().addListener((observable, oldValue, newValue) -> {

addButton.setDisable(newValue.trim().isEmpty() || surname.getText().isEmpty()

|| passport.getText().isEmpty());

});

passport.textProperty().addListener((observable, oldValue, newValue) -> {

addButton.setDisable(newValue.trim().isEmpty() || surname.getText().isEmpty()

|| name.getText().isEmpty());

});\*/

dialog.getDialogPane().setContent(grid);

// Request focus on the name field by default.

Platform.runLater(surname::requestFocus);

// Convert the result to a name-password-pair when the login button is clicked.

dialog.setResultConverter(dialogButton -> {

if (dialogButton == addButtonType) {

int max = 0;

for(int i = 0;i < sample.Client.getUsersData().size();i++)

max = Math.max(Client.getUsersData().get(i).getId(), max);

max++;

return new PhysicalPerson(max, surname.getText() , name.getText() ,

lastname.getText(), passport.getText());

}

return null;

});

Optional<PhysicalPerson> result = dialog.showAndWait();

result.ifPresent(data -> {

sample.Client.getUsersData().add(new PhysicalPerson

(data.getId(), data.getSurname(), data.getName(), data.getLastName() , data.getPassport()));

//sql insert into here

Statement stmt = null;

// Try to connect

try {

usersData.clear();

Connection connection = DriverManager.getConnection

("jdbc:mysql://localhost/FuneralHomeDB", "user", "fakdepolice");

stmt = connection.createStatement();

String sql = "SELECT id, surname, name, lastname, Passport FROM PHYSICALPERSON WHERE ( name between \""

+ name.getText() + "\" AND \"" + name2.getText() + "\") and ( surname between \""

+ surname.getText() + "\" AND \"" + surname2.getText() + "\") and ( lastname between \""

+ lastname.getText() + "\" AND \"" + lastname2.getText() + "\") and ( passport between \""

+ passport.getText() + "\" AND \"" + passport2.getText() + "\")";

ResultSet rs = stmt.executeQuery(sql);

//STEP 5: Extract data from result set

while (rs.next()) {

//Retrieve by column name

int id = rs.getInt("ID");

// String FIO = rs.getString("FIO");

String surname1 = rs.getString("SURNAME");

String name1 = rs.getString("NAME");

String lastName = rs.getString("LASTNAME");

String passport1 = rs.getString("passport");

//Display values

/\*System.out.print("ID: " + id);

System.out.print(", FIO: " + FIO);

System.out.print(", passport:\n " + passport);\*/

usersData.add(new PhysicalPerson(id, surname1, name1, lastName, passport1));

}

names.setItems(usersData);

}catch (SQLException e) {

e.printStackTrace();

}

// sql insert end

});

}

Метод, выполняющий выборку товаров по различным параметрам:

public void onFilterMethod(){

Dialog<Data.Goods> dialog = new Dialog<>();

dialog.setTitle("Фильтр");

dialog.setHeaderText(null);

// Set the icon (must be included in the project).

// Set the button types.

ButtonType addButtonType = new ButtonType("Search", ButtonBar.ButtonData.OK\_DONE);

dialog.getDialogPane().getButtonTypes().addAll(addButtonType, ButtonType.CANCEL);

// Create the name and password labels and fields.

GridPane grid = new GridPane();

grid.setHgap(10);

grid.setVgap(10);

grid.setPadding(new Insets(20, 50, 10, 10));

/\* ListView<Goods> goods = new ListView<>(ThirdPage.getUsersData());

goods.getSelectionModel().selectFirst();\*/

//grid.add(new Label("Наименование:"), 0, 0);

//grid.add(goods, 1, 0);

LimitedTextField name = new LimitedTextField(32);

name.setText("A");

LimitedTextField name2 = new LimitedTextField(32);

name2.setText("я");

grid.add(new Label("Наименование:"), 0, 0);

grid.add(name, 1, 0);

grid.add(name2, 2, 0);

LimitedTextField cost = new LimitedTextField(32);

cost.setText("0");

LimitedTextField cost2 = new LimitedTextField(32);

cost2.setText("999999");

grid.add(new Label("цена:"), 0, 1);

grid.add(cost, 1, 1);

grid.add(cost2, 2, 1);

/\*DatePicker order = new DatePicker();

order.setValue(LocalDate.now());

order.setShowWeekNumbers(true);

grid.add(new Label("Дата заказа:"), 0, 2);

grid.add(order, 1, 2);\*/

// Enable/Disable login button depending on whether a name was entered.

Node addButton = dialog.getDialogPane().lookupButton(addButtonType);

addButton.setDisable(false);

// Do some validation (using the Java 8 lambda syntax).

name.textProperty().addListener((observable, oldValue, newValue) -> {

addButton.setDisable(newValue.trim().isEmpty() || cost.getText().isEmpty());

});

cost.textProperty().addListener((observable, oldValue, newValue) -> {

addButton.setDisable(newValue.trim().isEmpty() || name.getText().isEmpty() );

try {

Integer.parseInt(newValue.trim());

}catch (Exception e){

addButton.setDisable(true);

Integer.parseInt(cost2.getText());

}

});

name2.textProperty().addListener((observable, oldValue, newValue) -> {

addButton.setDisable(newValue.trim().isEmpty() || cost.getText().isEmpty());

});

cost2.textProperty().addListener((observable, oldValue, newValue) -> {

addButton.setDisable(newValue.trim().isEmpty() || name.getText().isEmpty() );

try {

Integer.parseInt(newValue.trim());

Integer.parseInt(cost.getText());

}catch (Exception e){

addButton.setDisable(true);

}

});

dialog.getDialogPane().setContent(grid);

// Request focus on the name field by default.

Platform.runLater(name::requestFocus);

// Convert the result to a name-password-pair when the login button is clicked.

dialog.setResultConverter(dialogButton -> {

if (dialogButton == addButtonType) {

int max = 0;

for(int i = 0;i < sample.Client.getUsersData().size();i++)

max = Math.max(Client.getUsersData().get(i).getId(), max);

max++;

return new Data.Goods(max, name.getText() , Integer.parseInt(cost.getText()) );

}

return null;

});

Optional<Data.Goods> result = dialog.showAndWait();

result.ifPresent(data -> {

usersData.add(new Data.Goods(data.getId(), data.getName(), data.getCost()) );

//sql insert into here

Statement stmt = null;

// Try to connect

try {

usersData.clear();

Connection connection = DriverManager.getConnection

("jdbc:mysql://localhost/FuneralHomeDB", "user", "fakdepolice");

stmt = connection.createStatement();

String sql = "SELECT id, name, cost FROM GOODS WHERE ( name between \""

+ name.getText() + "\" AND \"" + name2.getText() + "\") and ( cost between "

+ cost.getText() + " AND " + cost2.getText() + ") ";

ResultSet rs = stmt.executeQuery(sql);

//STEP 5: Extract data from result set

while (rs.next()) {

//Retrieve by column name

int id = rs.getInt("ID");

// String FIO = rs.getString("FIO");

String name1 = rs.getString("NAME");

int cost1 = rs.getInt("COST");

//Display values

usersData.add(new Data.Goods(id, name1, cost1));

}

names.setItems(usersData);

}catch (SQLException e) {

e.printStackTrace();

}

// sql insert end

});

}

Метод, выполняющий выборку услуг по различным параметрам:

public void onFilterMethod(){

Dialog<Service> dialog = new Dialog<>();

dialog.setTitle("Фильтр");

dialog.setHeaderText(null);

// Set the icon (must be included in the project).

// Set the button types.

ButtonType addButtonType = new ButtonType("Search", ButtonBar.ButtonData.OK\_DONE);

dialog.getDialogPane().getButtonTypes().addAll(addButtonType, ButtonType.CANCEL);

// Create the name and password labels and fields.

GridPane grid = new GridPane();

grid.setHgap(10);

grid.setVgap(10);

grid.setPadding(new Insets(20, 50, 10, 10));

/\* ListView<Goods> goods = new ListView<>(ThirdPage.getUsersData());

goods.getSelectionModel().selectFirst();\*/

//grid.add(new Label("Наименование:"), 0, 0);

//grid.add(goods, 1, 0);

LimitedTextField name = new LimitedTextField(32);

name.setText("A");

LimitedTextField name2 = new LimitedTextField(32);

name2.setText("я");

grid.add(new Label("Наименование:"), 0, 0);

grid.add(name, 1, 0);

grid.add(name2, 2, 0);

LimitedTextField cost = new LimitedTextField(32);

cost.setText("0");

LimitedTextField cost2 = new LimitedTextField(32);

cost2.setText("999999");

grid.add(new Label("цена:"), 0, 1);

grid.add(cost, 1, 1);

grid.add(cost2, 2, 1);

/\*DatePicker order = new DatePicker();

order.setValue(LocalDate.now());

order.setShowWeekNumbers(true);

grid.add(new Label("Дата заказа:"), 0, 2);

grid.add(order, 1, 2);\*/

// Enable/Disable login button depending on whether a name was entered.

Node addButton = dialog.getDialogPane().lookupButton(addButtonType);

addButton.setDisable(false);

// Do some validation (using the Java 8 lambda syntax).

name.textProperty().addListener((observable, oldValue, newValue) -> {

addButton.setDisable(newValue.trim().isEmpty() || cost.getText().isEmpty());

});

cost.textProperty().addListener((observable, oldValue, newValue) -> {

addButton.setDisable(newValue.trim().isEmpty() || name.getText().isEmpty() );

try {

Integer.parseInt(newValue.trim());

}catch (Exception e){

addButton.setDisable(true);

Integer.parseInt(cost2.getText());

}

});

name2.textProperty().addListener((observable, oldValue, newValue) -> {

addButton.setDisable(newValue.trim().isEmpty() || cost.getText().isEmpty());

});

cost2.textProperty().addListener((observable, oldValue, newValue) -> {

addButton.setDisable(newValue.trim().isEmpty() || name.getText().isEmpty() );

try {

Integer.parseInt(newValue.trim());

Integer.parseInt(cost.getText());

}catch (Exception e){

addButton.setDisable(true);

}

});

dialog.getDialogPane().setContent(grid);

// Request focus on the name field by default.

Platform.runLater(name::requestFocus);

// Convert the result to a name-password-pair when the login button is clicked.

dialog.setResultConverter(dialogButton -> {

if (dialogButton == addButtonType) {

int max = 0;

for(int i = 0;i < sample.Client.getUsersData().size();i++)

max = Math.max(Client.getUsersData().get(i).getId(), max);

max++;

return new Service(max, name.getText() , Integer.parseInt(cost.getText()) );

}

return null;

});

Optional<Service> result = dialog.showAndWait();

result.ifPresent(data -> {

usersData.add(new Service(data.getId(), data.getName(), data.getCost()) );

//sql insert into here

Statement stmt = null;

// Try to connect

try {

usersData.clear();

Connection connection = DriverManager.getConnection

("jdbc:mysql://localhost/FuneralHomeDB", "user", "fakdepolice");

stmt = connection.createStatement();

String sql = "SELECT id, name, cost FROM SERVICES WHERE ( name between \""

+ name.getText() + "\" AND \"" + name2.getText() + "\") and ( cost between "

+ cost.getText() + " AND " + cost2.getText() + ") ";

ResultSet rs = stmt.executeQuery(sql);

//STEP 5: Extract data from result set

while (rs.next()) {

//Retrieve by column name

int id = rs.getInt("ID");

// String FIO = rs.getString("FIO");

String name1 = rs.getString("NAME");

int cost1 = rs.getInt("COST");

//Display values

/\*System.out.print("ID: " + id);

System.out.print(", FIO: " + FIO);

System.out.print(", passport:\n " + passport);\*/

usersData.add(new Service(id, name1, cost1));

}

names.setItems(usersData);

}catch (SQLException e) {

e.printStackTrace();

}

// sql insert end

});

}

# Выбор и основание средств разработки приложения

Для создания приложения используется платформа Java Standard Edition 6. Главное достоинство Java – кроссплатфоменность, т.е. возможность работы независимо от компьютерной архитектуры.

Подключение приложения к базе данных будет осуществляться через драйвер JDBC MySQL Connector/J от компании Sun Microsystems [X].

# Реализация законченного приложения, работающего с созданной базой данных

Главная форма приложения является объектом класса Main, наследуемый от класса Aplication, определенного в пакете javafx. application.Application

Рабочие формы приложения представлены в виде таблиц и диалоговых окон. Данные диалоги предназначены для выбора параметров запросов, добавления, редактирования записей.

Все классы форм и фреймов содержатся в пакете com.example.cursv1.sample. Диаграмма классов на языке UML [6] данного пакета представлена на рисунке 6.1.

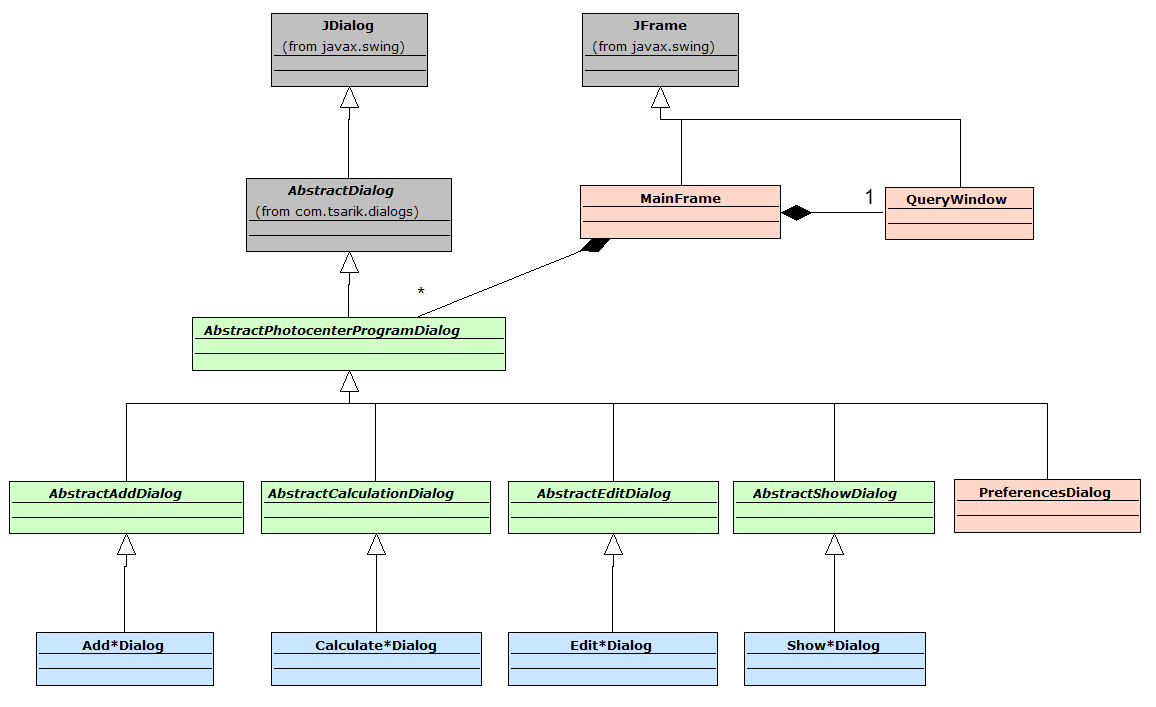


Рисунок 6.1 – Диаграмма классов пакета com.exapmle.cursv1.sample

Скриншоты главной и некоторых диалоговых окон представлены в приложении C.

## Построение главного меню и кнопок панели инструментов

Главное меню программы представлено тремя подменю: Users, Privileges, .

Линейка меню является объектом класса JMenuBar. Создание линейки главного меню:

// place menu bar

<MenuBar fx:id="menuBar" ><!--onKeyPressed="#handleKeyInput"-->

<menus>

<Menu text="Users">

<items>

<MenuItem text="New" onAction="#handleNew"/>

<MenuItem text="Delete" onAction ="#handleDrop"/>

</items>

</Menu>

<Menu text="Privileges">

<items>

<MenuItem text="Add" onAction="#handleGrant"/>

<MenuItem text="Revoke" onAction="#handleRevoke"/>

</items>

</Menu>

<Menu text="Show info">

<items>

<MenuItem text="DBs" onAction="#onShowMethod"/>

<MenuItem text="Users" onAction="#onShowUserMethod"/>

</items>

</Menu>

</menus>

</MenuBar>

## Выполнение программного кода на языке Java

Опишем работу приложения с базой данных. Все необходимые интерфейсы для работы с базами данных находятся в пакете java.sql. Как говорилось ранее в п.5, в программе используется драйвер JDBC MySQL Connector/J для подключения к базе данных. Подключение происходит следующим образом [5]:

Class.*forName*("com.mysql.jdbc.Driver");

Connection con = DriverManager.*getConnection*(url, user, password);

После подключения для осуществления запросов необходимо сначала получиться объект интерфейса Statement. Для этого используется метод: createStatement. Интерфейс Statement позволяет выполнять статические SQL предложения и возвращать результаты. Интерфейс PreparedStatement расширяет возможности интерфейса Statement, позволяет осуществлять повторное выполнение запросов, параметризацию запросов и др. Интерфейс CallableStatement используется для вызова хранимых процедур.

Выполнение запроса происходит следующим образом:

Statement stmt = null;

Class.forName("org.mariadb.jdbc.Driver");

System.out.println("Driver loaded(Client)");

// Try to connect

try {

Connection connection = DriverManager.getConnection

("jdbc:mysql://localhost/FuneralHomeDB", "user", "fakdepolice");

stmt = connection.createStatement();

String sql = "SELECT id, surname, name, lastname, Passport FROM PHYSICALPERSON";

ResultSet rs = stmt.executeQuery(sql);

//STEP 5: Extract data from result set

while(rs.next()){

//Retrieve by column name

int id = rs.getInt("ID");

// String FIO = rs.getString("FIO");

String surname = rs.getString("SURNAME");

String name = rs.getString("NAME");

String lastName = rs.getString("LASTNAME");

String passport = rs.getString("passport");

//Display values

/\*System.out.print("ID: " + id);

System.out.print(", FIO: " + FIO);

System.out.print(", passport:\n " + passport);\*/

usersData.add(new PhysicalPerson(id, surname, name, lastName,passport));

}

sql = "SELECT id, name, address, bankbook FROM JURIDICALPERSON";

rs = stmt.executeQuery(sql);

//STEP 5: Extract data from result set

while(rs.next()){

//Retrieve by column name

int id = rs.getInt("id");

String name = rs.getString("name");

String address = rs.getString("address");

String bankBook = rs.getString("bankBook");

usersData.add(new JuridicalPerson(id,name,address,bankBook));

}

rs.close();

//System.out.println("It works!");

connection.close();

}catch (Exception e){

System.out.println(e.getMessage());

}

/\* usersData.add(new PhysicalPerson(usersData,"Алексеев Алексей","VM12345678"));

usersData.add(new PhysicalPerson(usersData,"Иванов Иван","VM10000000"));

usersData.add(new JuridicalPerson(usersData, "г. Минск ул. Лопатина д.12", "qwe11000011"));\*/

}

# Заключение

В результате выполненной курсовой работы была создана база данных фотоцентра, а также программный продукт, позволяющий пользователю взаимодействовать с базой данных. Базы данных была разработана в среде MySQL, приложение – Java SE.

Приложение позволяет:

* просматривать различные данные по Бюро ритуальных услуг;
* добавлять и редактировать заказы, услуги и клиентов Бюро.

Запросы, указанные в варианте задания, были «растворены» в приложении.

В процессе выполнения данной курсовой работы были закреплены навыки в программировании на Java SE 6.0, проектировании баз данных и реализации их в СУБД MySQL.

# Список использованных источников

1. Data flow diagram - Wikipedia, the free encyclopedia / Многоязычная общедоступная свободно распространяемая энциклопедия, публикуемая в Интернете Википедия. Режим доступа: http://en.wikipedia.org/wiki/Data\_flow\_diagram
2. Entity-relationship model – Wikipedia, the free encyclopedia / Многоязычная общедоступная свободно распространяемая энциклопедия, публикуемая в Интернете Википедия. Режим доступа: http://en.wikipedia.org/wiki/Entity-relationship\_model
3. MySQL 5.0 Reference Manual [Электронный ресурс] / Официальный сайт MySQL. Документация по MySQL. Режим доступа:

http://dev.mysql.com/doc/refman/5.0/en/index.html

1. Документация по языку Java от компании Sun [Электронный ресурс]. – Элек-трон. текстовые дан. (270 Мб). – Sun Microsystems, Inc., 2006
2. Документация по MySQL Connector/J [Электронный ресурс]. – Элек-трон. текстовые дан. (3 Мб). – Sun Microsystems, Inc., 2008
3. Unified Modeling Language – Wikipedia, the free encyclopedia / Многоязычная общедоступная свободно распространяемая энциклопедия, публикуемая в Интернете Википедия. Режим доступа: http://en.wikipedia.org/wiki/Unified\_Modeling\_Language

Приложение A: Концептуальная схема БД



Рисунок A.1 – Инфологическая схема проектируемой базы данных