СОДЕРЖАНИЕ

ВВЕДЕНИЕ………………………………………………………………………………………. 5

1 ПОСТРОЕНИЕ ИНФОЛОГИЧЕСКОЙ КОНЦЕПТУАЛЬНОЙ МОДЕЛИ………………. 6

1.1 Анализ предметной области и выявление необходимого набора сущностей………. 6

1.2 Обоснование требуемого набора атрибутов для каждой сущности и выделение идентифицирующих атрибутов……………………………………………………………... 7

1.3 Определение связей между объектами……………………………………………….. 10

1.4 Описание полученной модели на языке инфологического проектирования………... 11

2 ПОСТРОЕНИЕ СХЕМЫ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ…………………………... 12

2.1 Построение набора необходимых отношений базы данных…………………………. 12

2.2 Задание первичных и внешних ключей определенных отношений…………………. 13

2.3 Приведение отношения БД к третьей нормальной форме…………………………... 14

2.4 Определение ограничений целостности для внешних ключей отношений и для отношений в целом………………………………………………………………………….. 15

2.5 Графическое представление связей между внешними и первичными ключами…… 15

3 СОЗДАНИЕ СПРОЕКТИРОВАННОЙ БАЗЫ ДАННЫХ………………………………... 16

4 ЗАПИСЬ ВЫРАЖЕНИЙ, УКАЗАННЫХ В ВАРИАНТЕ ЗАДАНИЯ ТИПОВ ЗАПРОСОВ НА ЯЗЫКЕ SQL……………………………………………………………… 26

5 ВЫБОР И ОБОСНОВАНИЕ СРЕДСТВ РАЗРАБОТКИ ПРИЛОЖЕНИЯ……………… 36

6 РЕАЛИЗАЦИЯ ЗАКОНЧЕННОГО ПРИЛОЖЕНИЯ, РАБОТАЮЩЕГО С СОЗДАННОЙ БАЗОЙ ДАННЫХ…………………………………………………………………………... 37

6.1 Разработка и построение интерфейса главной и рабочей форм……………………... 37

6.2 Построение главного меню и кнопок панели инструментов………………………… 37

6.3 Выполнение программного кода в среде Microsoft Visual C#...................................... 38

ЗАКЛЮЧЕНИЕ………………………………………………………………………………… 39

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ………………………………………….. 40

ПРИЛОЖЕНИЕ А – КОНЦЕПТУАЛЬНАЯ СХЕМА БД…………………………………... .41

ПРИЛОЖЕНИЕ В – СХЕМА РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ………………………… 43

ПРИЛОЖЕНИЕ С – ГЛАВНАЯ И РАБОЧИЕ ФОРМЫ ПРИЛОЖЕНИЯ…………………. 44

ПРИЛОЖЕНИЕ D – ЛИСТИНГ ПРОГРАММЫ…………………………………………….. 46

ВВЕДЕНИЕ

Компьютерные игры в двадцать первом веке заняли место среди популярных средств получения досуга. Каждая новая игра позваляет получить своему пользователю какой-то новый опыт, которого ранее он не получал, при этом не создавая опасности для жизни. В 2018 году количество выпусков новых игр составило более сотни уникальных идей. С каждым годом количество выпускаемых игру увеличивается, а борьба за своего потенциального клиента увеличивается ещё быстрее. Уже достаточно давно существуют площадки для разработчиков, которые предоставляют услуги по размещению, хранению, обновлению и распространению компьютерных игр среди пользователей. Такие системы позволяют намного быстрее продвигать свой продукт, однако сегодня такой подход исчерпал себя из-за наплыва огроимного количества индивидуальных рпазработчиков и их низкокачественных проектов. К тому же отсутствие конкуренции на рынке площадок распространение компьютерных игр приводит только к негативным последствиям.

В 2018 году крупные игроки рынка компьютерных игр приняли решенние о переходи с централизованной площадки реализации игр на свои закрытые локальные площадки, в которых они смогут реализовывать только свою продукцию, забирая всю прибыль и внимание только себе. Для этих целей данным компаниям необходимо реализовать своё уникальное хранилище данных, в котором они будут хранить всю информацию о своих продуктах, а также о пользовательской активности в данной системе. В рамках курсовой работы будет выполнено проектирование такой системы и будет реализовано графическое приложение для получения доступа в эту систему и управление ею.

Курсовая работа имеет своей целью:

− систематизацию знаний и накопление первоначального опыта в области проектирования баз данных и автоматизированных рабочих мест, закрепление, углубление и обобщение теоретических знаний по дисциплине «Базы данных» и применение их для решения научно-технических задач в соответствии с выбранной темой;

− формирование навыков самостоятельной работы и овладение методикой и технологией проектирования баз данных в составе корпоративных информационных систем;

− приобретение навыков обобщения и анализа результатов, полученных другими разработчиками по выбранной предметной области.

1 ПОСТРОЕНИЕ ИНФОЛОГИЧЕСКОЙ  
КОНЦЕПТУАЛЬНОЙ МОДЕЛИ

Инфологическая модель данных – описание, выполненное с использованием естественного языка, математических формул, таблиц, графиков и других средств, понятных всем людям, работающих над проектированием базы данных.

Цель инфологического моделирования – обеспечение наиболее естественных для человека способов сбора и представления той информации, которую предполагается хранить в создаваемой базе данных. Поэтому инфологическую модель данных пытаются строить по аналогии с естественным языком. Основными конструктивными элементами инфологических моделей являются сущности, связи между ними и их свойства (атрибуты).

1.1 Анализ предметной области и   
выявление необходимого набора сущностей

Перед началом разработки базы данных, необходимо определить основные цели, задачи и правила для решаемой проблемы, после чего приступать к проектированию. Поэтому сформулируем краткое описание поставленной задачи.

**Наименование задачи** *–* автоматизация учета контингента студентов университета методистом университета.

**Цель работы методистов** *–* накопление и систематизация информации об учащихся.

**Функции методиста:**

* Учет учащихся учреждения образования.
* Систематизация накопленной информации.
* Составление документов и отчетов.
* Управление перемещением учащихся по структуре учебного заведения.
* Анализ накопленной информации.

**Основные бизнес-правила:**

* Информация об учащихся храниться до её явного удаления из системы.
* После составления управляющих документов, необходимо изменять соответствующие свойства учащихся на вновь определенные.
* Учащийся может иметь одну национальность и несколько гражданств.
* Каждый документ должен иметь уникальный номер, не повторяющийся в пределах одного типа содержания.
* У студента может быть один или несколько адресов, но только один из них может быть адресом регистрации.
* Академический отпуск может оформляться один раз по болезни и несколько раз по беременности и родам.

**Перечень печатных отчетов:**

* Приказ на отчисление за академическую неуспеваемость.
* Распоряжение о переводе в другую группу.
* Справка о том, что студент действительно обучается в ВУЗе по месту требования.

Для построения информационной системы требуется для начала выделить необходимы набор сущностей, которые описывают эту систему. Данный набор должен удовлетворять всем условиям на проектирование системы.

Определим минимальный набор сущностей, необходимый для проектирования информационной системы для учета контингента студентов университета. Для определения первичного набора сущностей будет проведён анализ технического задания и предметной области.

Для учета студентов в университете необходимо описание такой сущности как *студент*. В данной сущности должны быть заложены описательные характеристики студента, однозначно идентифицирующие его.

Все студенты обучаются в группах, поэтому следует выделить так же сущность – *группа*.

Студенты, организованные в группы, учатся на одном из факультетов, возглавляемом деканатом. Отсюда следует необходимость выделения сущности – *факультет*.

Структура факультета выглядит следующим образом: каждый факультет состоит из *кафедр*, за кафедрами числятся *специальности*. В свою очередь специальности имеют одну или несколько *специализаций*. На каждой специальности есть действующие *формы обучения* (одна или несколько). Следовательно, информация о данных структурах будет храниться в соответствующих сущностях.

Личные данные о студентах должны включать информацию о его *проживании*, *образовании*, *гражданстве и национальности,* *паспортные данные*, информацию о *работе* и *родственниках, семейном положении*. Всю данную информацию будем хранить в отдельных соответствующих сущностях.

Деканат оформляет на студентов различные документы, следовательно можно выделить сущность – *документ*. Каждый документ имеет свою *форму*, *назначение* и *содержание*, которые так же являются отдельными сущностями. К документу может прилагаться дополнительная информация, поэтому так же необходимо определить сущность «*Дополнительная информация к документу*».

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

Атрибут – поименованная характеристика сущности.

Атрибутом сущности является любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности. Его наименование должно быть уникальным для конкретного типа сущности, но может быть одинаковым для различного типа сущностей.

Для каждой сущности, выделенной в пункте 1.1. необходимо определить атрибуты.

1. Сущность – «Студент»:

**Атрибуты:** Surname (Фамилия), Name (Имя), Patronymic (Отчество), DateOfBirth (Дата рождения), Sex (Пол), Status (Статус), NumberOfRecordBook (Номер зачетной книжки), NumberOfStudetnID (Номер студенческого билета), ForeignLanuage (Иностранные языки), Группа, Национальность, Форма обучения.

Атрибут «форма обучения» был введен в сущность «*Студент*» т.к. первоначально студенты зачисляются на определенную форму обучения, после чего в дальнейшем распределяются по группам.

1. Сущность – «Факультет»:

**Атрибуты:** Title (Название), Dean (ФИО декана факультета), Chiper (Шифр), Telephone (Телефон), Email (адрес электронной почты).

1. Сущность – «Кафедра»:

**Атрибуты:** *Title* (Название), *Head* (ФИО заведующего кафедрой), *Chiper* (Шифр), *Telephone* (Телефон).

1. Сущность – «Специальность»:

**Атрибуты:** *Title* (Название), *Chiper* (Шифр), *Years* (Срок обучения).

1. Сущность – «Специализация»:

**Атрибуты:** Title (Название), *Chiper* (Шифр), *Qualification* (Квалификация).

1. Сущность – «Форма обучения»:

**Атрибуты:** *Form* (Название формы).

1. Сущность – «Группа»:

**Атрибуты:** *Title* (Название), *YearOfFoundation* (Год создания).

1. Сущность – «Национальность»:

**Атрибуты:** *NationalityType* (Название).

1. Сущность – «Образование»:

**Атрибуты:** *Title* (Название учреждения образования), *Type* (Тип УО), *YearEnd* (Год окончания), *DocNumber* (Номер документа об образовании), *Ball* (Средний балл аттестата), *Specialty* (Специальность, если диплом).

1. Сущность – «Работа»:

**Атрибуты:** *PlaceName* (Название места работы), *Experience* (Стаж), *Post* (Должность).

1. Сущность – «Паспорт»:

**Атрибуты:** *Serie* (Серия), *Number* (Номер), *Issue* (Кем выдан), *Date* (Дата выдачи).

1. Сущность – «Адрес»:

**Атрибуты:** *Index* (Индекс), *Country* (Страна), *District* (Район/Область), *Locality* (Населенный пункт), *Street* (Улица), *House* (Дом), *Housing* (Корпус), *Apartment* (Квартира), *Telephone* (Телефон), *isCurrent* (Текущий адрес – да/нет), *isForeign* (Иностранный адрес - да/нет), *isRegistration* (Адрес регистрации - да/нет).

1. Сущность – «Гражданство»:

**Атрибуты:** *Titile* (Название).

1. Сущность – «Семейное положение»:

**Атрибуты:**, *Married* (Замужем/Женат – да/нет), *Disabled* (Инвалид – да/нет), *Guardianship* (Опека –- да/нет), *Allowance* (Пособие – да/нет).

1. Сущность – «Родственник»:

**Атрибуты:** *Name* (ФИО родственника), *DateOfBirth* (Дата рождения), *Telephone* (Телефон), *WorkPlace* (Место работы).

1. Сущность – «Родство»:

**Атрибуты:** *Relationship* (Тип родства).

1. Сущность – «Документ»:

**Атрибуты:** *Форма*, *Содержание*, *Назначение*,*DocNumber* (Номер документа), *DocumentDate* (Дата подписания).

1. Сущность – «Форма документа»:

**Атрибуты:** *DocumentForm* (Название формы).

1. Сущность – «Назначение документа»:

**Атрибуты:** *Appointment* (Назначение).

1. Сущность – «Содержание документа»:

**Атрибуты:** *ContentType* (Тип содержания), *DocNumber* (Номер документа).

Атрибут «*DocNumber*» был вынесен в эту сущность, т.к. номер должен быть уникальным в пределах одного типа содержания.

1. Сущность – «Дополнительная информация к документу»:

**Атрибуты:** *FromPlace* (Название места – откуда), *InPlace* (Название места – куда), *Difference* (Ликвидируемая разница), *TempDate* (Дата), *OldNumber* (Старый номер), *NewNumber* (Новый номер), *OldSurname* (Старая фамилия), *NewSurname* (Новая фамилия), *Amount* (Размер поощрения), *Qualification* (Квалификация), *Semestr* (Номер семестра).

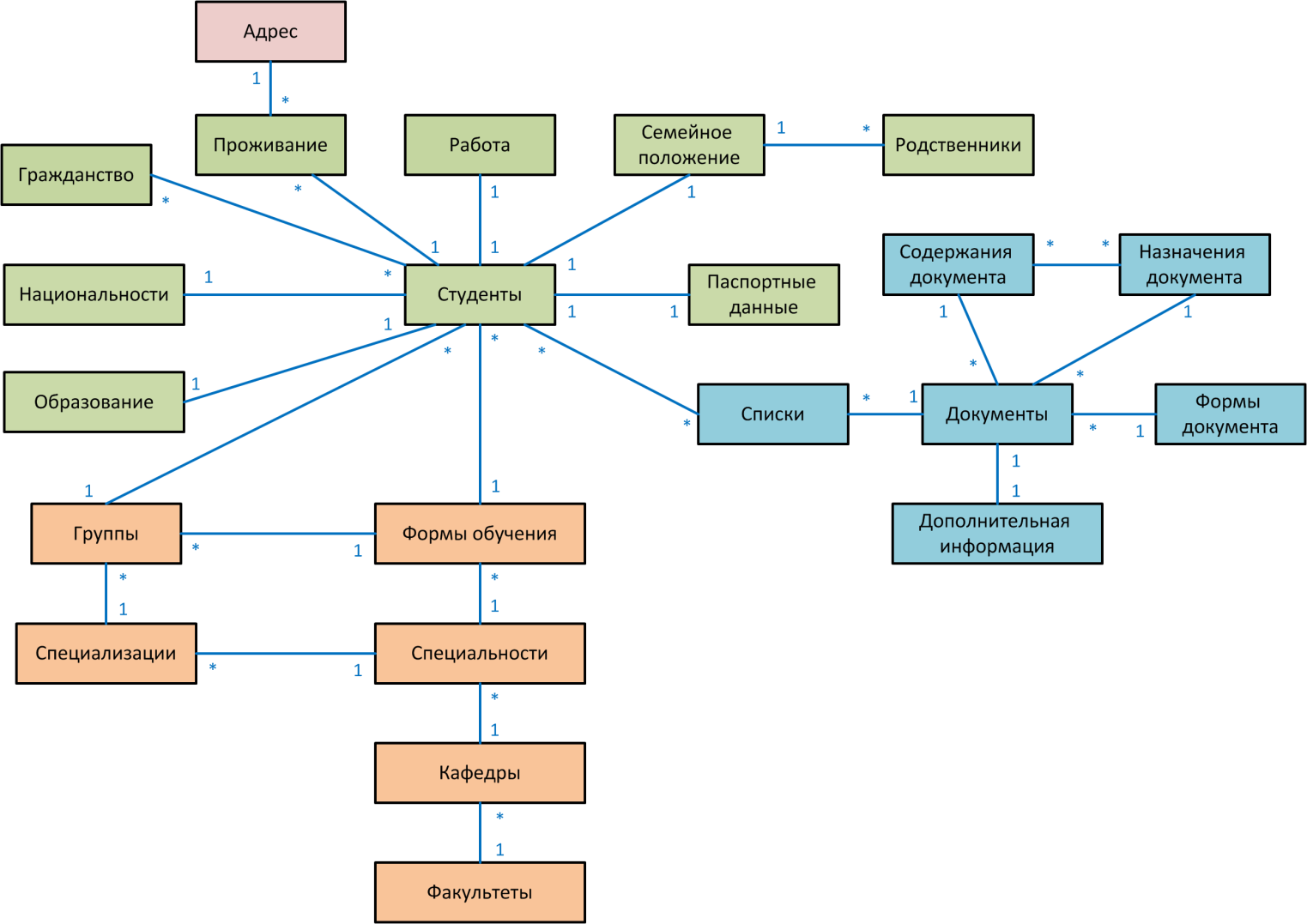
1.3 Определение связей между объектами

Следующим этапом в проектировании инфологической модели является установление связи между сущностями.

Связь – это ассоциирование двух или более сущностей. Эта ассоциация всегда является бинарной и может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь). В любой связи выделяются два конца (в соответствии с существующей парой связываемых сущностей), на каждом из которых указывается имя конца связи, степень конца связи (сколько экземпляров данной сущности связывается), обязательность связи (т.е. любой ли экземпляр данной сущности должен участвовать в данной связи).

Для реализации информационной системы учета контингента студентов университета необходимо установить все связи между объектами. А именно, нужно рассмотреть всю информационную систему в совокупности и определить взаимное влияние объектов, составляющих систему.

Этот процесс изображен на рис. 1.1



**Рисунок 1.1 – Концептуальная схема базы данных**

Проследить отношения, в которых состоят таблицы базы данных можно по схеме, изображенной на в приложении A на рис. А.1.

1.4 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 Crow's Foot(рус. «воронья лапка») сущность изображается в виде прямоугольника. Связь изображается линией, которая связывает две сущности, участвующие в отношении. Степень конца связи указывается графически, множественность связи изображается в виде «вилки» на конце связи. Модальность связи так же изображается графически — необязательность связи помечается кружком на конце связи. Атрибуты сущности записываются внутри прямоугольника, изображающего сущность.

На основе проведенного проектирования, в частности на основе инфологической схемы, приведенной на рис. 1.1, получим ER-диаграмму, проектируемой базы данных, представленную в приложении A на рис. А.2.

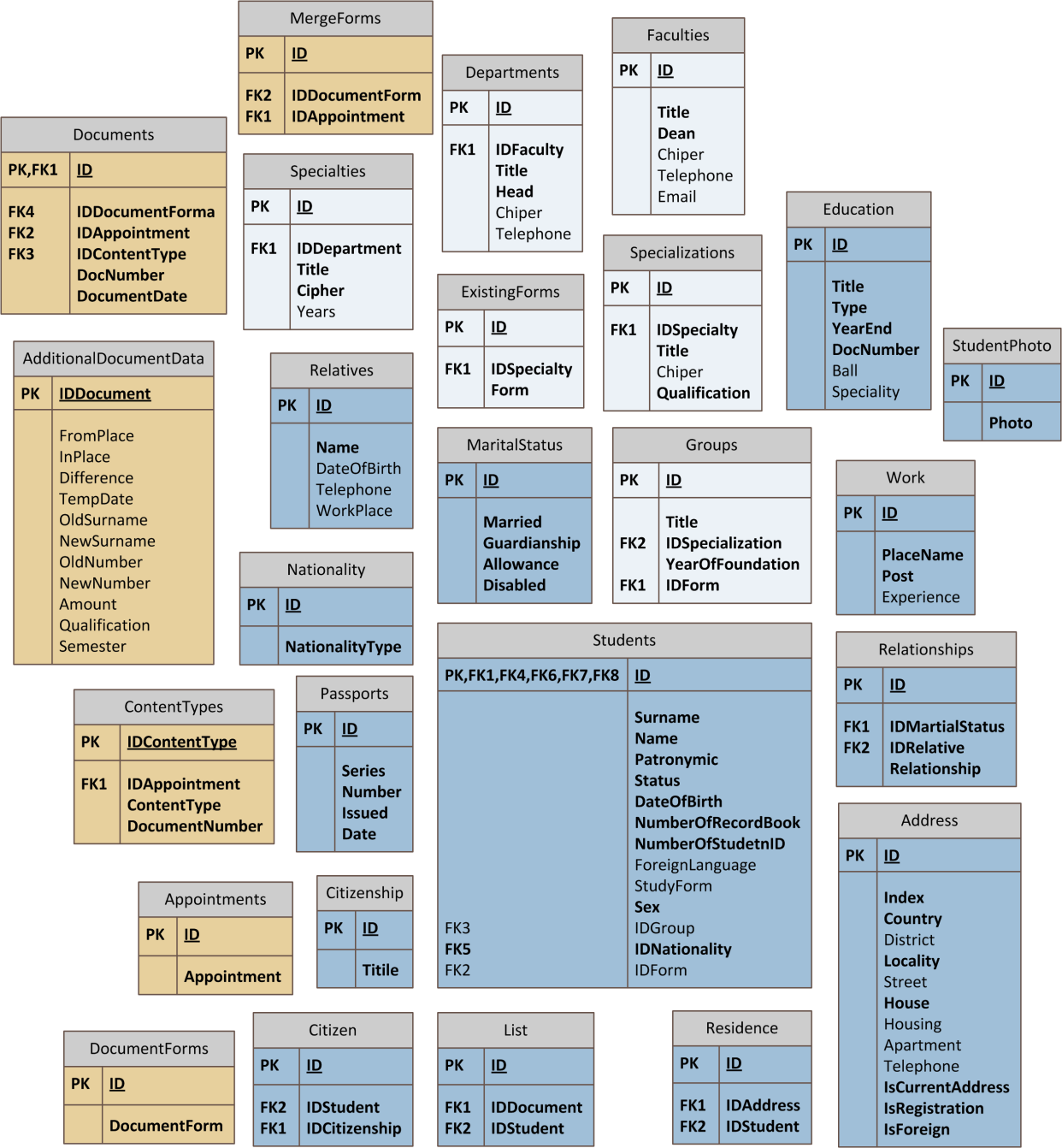
2 ПОСТРОЕНИЕ СХЕМЫ РЕЛЯЦИОННОЙ  
БАЗЫ ДАННЫХ

2.1 Построение набора необходимых отношений базы данных

Для построения схемы реляционной базы данных необходимо определить совокупность отношений, составляющих базу данных. Эта совокупность отношений будет содержать всю информацию, которая должна храниться в базе данных.

На основе полученной в первом пункте концептуальной модели можно определить набор необходимых отношений базы данных.

Для реализации связи были созданы дополнительные сущности отношения сущностей – *List*, *Residense*, *MergeForms*, *Citizen*. На рисунке 2.1 представлены отношения для базы данных информационной системы университета.



**Рисунок 2.1 – Набор необходимых отношений базы данных**

Схема реляционной базы данных представлена в приложении В.

2.2 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 атрибутам. Каждое отношение должно обладать хотя бы одним ключом. В таблице 2.1 определены первичные и внешние ключи для отношений.[4]

**Таблица 2.1 – Первичные и внешние ключи отношений**

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Название таблицы | Первичный ключ | Внешние ключи |
|  | Students | ID | IDForm  IDGroup  IDNationality |
|  | Nationality | ID |  |
|  | Citizen | ID | IDStudent  IDCitizenship |
|  | Citizenship | ID |  |
|  | Groups | ID | IDSpecialization  IDForm |
|  | Faculties | ID |  |
|  | Departments | ID | IDFaculty |
|  | Specialties | ID | IDDepartment |
|  | Specealizations | ID | IDSpecialty |
|  | ExistingForms | ID | IDSpecialty |
|  | Residence | ID | IDStudent  IDAddress |
|  | Address | ID |  |
|  | Work | ID | ID |
|  | Education | ID | ID |
|  | MaritalStatus | ID | ID |
|  | Relationships | ID | IDMaritalStatus  IDRelative |
|  | Relatives | ID |  |
|  | Passports | ID | ID |
| **Продолжение таблицы 2.1** | | | |
|  | List | ID | IDStudent  IDDocument |
|  | StudentPhoto | ID | ID |
|  | Documents | ID | IDAppointment  IDDocumentForma  IDContentType |
|  | Apointments | ID |  |
|  | DocumentForms | ID |  |
|  | MergeForms | ID | IDAppointment  IDDocumentForm |
|  | ContentTypes | IDContentType | IDAppointment |
|  | AdditionalDocumentData | IDDocument | IDDocument |

2.3 Приведение отношения БД к третьей нормальной форме

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.[4]

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 – неключевые атрибуты не должны определять другие неключевые атрибуты.

В спроектированной базе данных почти все отношения находятся в третьей нормальной форме, за исключением таблицы *AddtitonalDocumentData*, ее целесообразно оставить во второй нормальной форме.

2.4 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.

Ограничение целостности для внешних ключей состоит в том, что значение внешнего ключа должно быть равным значению первичного ключа цели; либо быть полностью неопределенным, т.е. каждое значение атрибута, участвующего во внешнем ключе должно быть неопределенным.

Условиями целостности называется набор правил, используемых для поддержания допустимых межтабличных связей и запрета на случайное изменение или удаление связанных данных. Следует устанавливать целостность данных только при выполнении следующих условий: связываемое поле из главной таблицы является полем первичного ключа и имеет уникальный индекс, связанные поля имеют один и тот же тип данных.

Для автоматического обновления связанных полей (удаления записей) при обновлении (удалении) в главной таблице, следует устанавливать обеспечение целостности данных и каскадное обновление связанных полей (каскадное удаление связанных записей).

Ограничение целостности, накладываемые на разрабатываемую систему:

* ключевое поле отношения должно быть уникальным;
* внешний ключ должен быть повторяющимся, то есть соответствовать уникальному ключу в своем отношении.

Для удовлетворения требования ограничения целостности для внешних ключей отношений и для отношений в целом необходимо, чтобы выполнялось соответствие между типами вводимых данных и типами столбцов в таблицах, а так же чтобы были заполнены все обязательные поля в таблицах, т.е. те поля которые не могут содержать значения NULL.

Для хранения информации студенте необходимо обязательно заполнить такие данные как, информация об образовании, паспортные данные и семейное положение, для этого между таблицами Students и Education, Students и MaritalStatus, Students и Passports были установлены связи один-к-одному.

Т.к. студент не обязательно может иметь данные о работе, либо фотографию, то контроль ограничения целостности для отношений между таблицами Students и Work, Students и StudentPhoto отключен.

Так как для оформляемого документа может отсутствовать дополнительная информация, то для отношения между таблицами Documents и AdditionalDocumetnData ограничение контроля целостности отключено, а кардинальность связи установлена один-к одному.

2.5 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. Полученная ER-диаграмма построена по методу Crow's Foot (рус. «воронья лапка») [4]. Средства моделирования Crow's Foot специально разработаны для построения реляционных информационных систем. На ней изображаются все отношения базы данных, а также связей между внешними и первичными ключами. Первичные ключи обозначаются знаком ключа, внешние ключи обозначаются знаком ссылки.

3 СОЗДАНИЕ СПРОЕКТИРОВАННОЙ БАЗЫ ДАННЫХ

Для реализации спроектированной базы данных была выбрана система управления базами данных MS SQL Server 2008. Это обусловлено тем, что, данная СУБД имеет большую функциональность, множество средств для поддержки и работы с ней, развитую инфраструктуру для интеграции баз данных в пользовательские приложения.

В создаваемой базе данных будут использоваться следующие типы данных:

* INT – Целочисленный тип. Размер – 4 байта
* NVARCHAR – Строковый тип переменной длины
* VARBINARY – Двоичные данные
* TINYINT – Целочисленный тип размером в 1 байт.
* BIT – Битовый тип. Используется как логический тип
* DATE – Тип, определяющий дату.
* SMALLINT – Целочисленный тип. Размер – 2 байта
* DECIMAL – Число с плавающей точкой. Размер – 8 байт

Опишем все таблицы, которые будут созданы в базе данных.

Таблица *Students* содержит список всех студентов когда либо обучавшихся в университете. Ее структура приведена в таблице 3.1.

**Таблица 3.1 – Характеристика атрибутов таблицы Students**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор студента. Ключевой атрибут |
| Surname | NVARCHAR(40) | Фамилия студента |
| Name | NVARCHAR(40) | Имя студента |
| Patronymic | NVARCHAR(40) | Отчество студента |
| Status | NVARCHAR(30) | Текущий статус студента |
| DateOfBirth | DATE | Дата рождения студента |
| Sex | BIT | Пол студента (0 –мужской, 1 – женский) |
| NumberOfRecordBook | INT | Номер зачетной книжки студента |
| NumberOfStudetnID | INT | Номер студенческого билета |
| ForeignLanguage | NVARCHAR(80) | Иностранные языки, которыми владеет студент. Необязательный атрибут |
| StudyForm | NVARCHAR(20) | Основание для обучения (договор, бюджет). Необязательный атрибут |
| **Продолжение таблицы 3.1** | | |
| IDGroup | INT | Идентификатор группы студента. Необязательный атрибут |
| IDForm | INT | Идентификатор формы обучения. Необязательный атрибут |
| IDNationality | INT | Идентификатор национальности студента |

Таблица *Work* содержит информацию о прежних рабочих местах студентов. Ее структура приведена в таблице 3.2.

**Таблица 3.2 – Характеристика атрибутов таблицы Work**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор работы. Ключевой атрибут |
| PlaceName | NVARCHAR(80) | Название места работы |
| Post | NVARCHAR(60) | Занимаемая должность |
| Experience | NVARCHAR(10) | Стаж работы. Необязательный атрибут |

Таблица *Education* содержит информацию об образовании студентов. Ее структура приведена в таблице 3.3.

**Таблица 3.3 – Характеристика атрибутов таблицы Education**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор образования. Ключевой атрибут |
| Title | NVARCHAR(80) | Название учреждения образования |
| Type | NVARCHAR(40) | Тип УО |
| YearEnd | SMALLINT | Год окончания |
| DocNumber | NVARCHAR(15) | Номер документа об образовании |
| Ball | DECIMAL (3,1) | Средний балл аттестата. Необязательный атрибут |
| Speciality | NVARCHAR(70) | Полученная специальность. Необязательный атрибут |

Таблица *Passports* содержит паспортные данные студентов. Ее структура приведена в таблице 3.4.

**Таблица 3.4 – Характеристика атрибутов таблицы Passports**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор паспортных данных. Ключевой атрибут |
| Serie | NVARCHAR(10) | Серия |
| Number | INT | Номер |
| Issued | NVARCHAR(80) | Кем выдан |
| Date | DATE | Дата выдачи |

Таблица *MaritalStatus* содержит информацию о семейном положении студентов. Ее структура приведена в таблице 3.5.

**Таблица 3.5 – Характеристика атрибутов таблицы MaritalStatus**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор семейного положения. Ключевой атрибут |
| Married | BIT | Замужем/Женат – да/нет |
| Guardianship | BIT | Опека – да/нет |
| Allowance | BIT | Пособие – да/нет |
| Disabled | BIT | Инвалид – да/нет |

Таблица *Address* содержит информацию о местах проживания студентов. Ее структура приведена в таблице 3.6.

**Таблица 3.6 – Характеристика атрибутов таблицы Address**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор адреса. Ключевой атрибут |
| Index | INT | Почтовый индекс |
| Country | NVARCHAR(60) | Страна |
| District | NVARCHAR(60) | Район/Область. Необязательный атрибут |
| Locality | NVARCHAR(60) | Населенный пункт |
| Street | NVARCHAR(60) | Улица. Необязательный атрибут |
| House | NVARCHAR(10) | Дом |
| **Продолжение таблицы 3.6** | | |
| Housing | NVARCHAR(10) | Корпус. Необязательный атрибут |
| Apartment | NVARCHAR(10) | Квартира. Необязательный атрибут |
| Telephone | NVARCHAR(20) | Телефон. Необязательный атрибут |
| isCurrent | BIT | Является ли адрес текущим местом проживания студента – да/нет |
| isRegistration | BIT | Является ли адрес местом регистрации студента – да/нет |
| isForeign | BIT | Является ли адрес иностранным – да/нет |

Таблица *Residence* служит для соединения адресов проживания ос студентами. Ее структура приведена в таблице 3.7.

**Таблица 3.7 – Характеристика атрибутов таблицы Residence**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор проживания. Ключевой атрибут |
| IDAddress | BIT | Идентификатор адреса |
| IDStudent | BIT | Идентификатор студента |

Таблица *Relatives* служит хранения информации о родственниках студентов. Ее структура приведена в таблице 3.8.

**Таблица 3.8 – Характеристика атрибутов таблицы Relatives**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор родственника. Ключевой атрибут |
| Name | NVARCHAR(50) | ФИО родственника |
| DateOfBirth | DATE | Дата рождения. Необязательный атрибут |
| Telephone | NVARCHAR(20) | Контактный телефон. Необязательный атрибут |
| WorkPlace | NVARCHAR(80) | Место работы. Необязательный атрибут |

Таблица *Relationships* служит для связи родственников со студентами и определения степени родства. Ее структура приведена в таблице 3.9.

**Таблица 3.9 – Характеристика атрибутов таблицы Relationships**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор родства |
| IDMaritalStatus | INT | Идентификатор семейного положения |
| IDRelative | INT | Идентификатор родственника |
| Relationship | NVARCHAR(30) | Степень родства |

Таблица *Citizenship* служит для хранения типов гражданств. Ее структура приведена в таблице 3.10.

**Таблица 3.10 – Характеристика атрибутов таблицы Citizenship**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор гражданства |
| Title | NVARCHAR(30) | Название гражданства |

Таблица *Citizen* служит для связи студентов и их гражданств. Ее структура приведена в таблице 3.11.

**Таблица 3.11 – Характеристика атрибутов таблицы Citizen**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор гражданина |
| IDCitizenship | INT | Идентификатор типа гражданства |
| IDStudent | INT | Идентификатор студента |

Таблица *Nationality* служит для связи студентов и их гражданств. Ее структура приведена в таблице 3.12.

**Таблица 3.12 – Характеристика атрибутов таблицы Nationality**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор национальности |
| NationalityType | NVARCHAR(30) | Название национальности |

Таблица *StudentPhoto* служит для хранения фотографии студента. Эта таблица была создана отдельно для того, чтобы при выполнении запросов не нужно было загружать большой объем данных, который представляет собой изображение, а выполнять загрузку только по требованию. Ее структура приведена в таблице 3.13.

**Таблица 3.13 – Характеристика атрибутов таблицы StudentPhoto**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор фотографии |
| NationalityType | VARBINARY(MAX) | Двоичные данные фотографии |

Таблица *Faculties* служит для хранения информации о факультетах. Ее структура приведена в таблице 3.14.

**Таблица 3.14 – Характеристика атрибутов таблицы Faculties**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор факультета |
| Title | NVARCHAR(80) | Название факультета |
| Dean | NVARCHAR(50) | ФИО декана факультета |
| Chiper | NVARCHAR(15) | Шифр факультета (сокращение). Необязательный параметр |
| Telephone | NVARCHAR(25) | Телефон деканата. Необязательный параметр |
| Email | NVARCHAR(50) | Адрес электронной почты. Необязательный параметр |

Таблица *Departments* служит для хранения информации о кафедрах. Ее структура приведена в таблице 3.15.

**Таблица 3.15 – Характеристика атрибутов таблицы Departments**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор факультета |
| Title | NVARCHAR(80) | Название кафедры |
| Head | NVARCHAR(50) | ФИО заведующего кафедрой |
| Chiper | NVARCHAR(15) | Шифр кафедры (сокращение). Необязательный параметр |
| Telephone | NVARCHAR(25) | Телефон кафедры. Необязательный параметр |
| IDFaculty | INT | Идентификатор факультета, к которому принадлежит кафедра |

Таблица *Specialties* служит для хранения информации о специальностях. Ее структура приведена в таблице 3.16.

**Таблица 3.16 – Характеристика атрибутов таблицы Specialties**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор специальности |
| Title | NVARCHAR(80) | Название специальности |
| Chiper | NVARCHAR(15) | Шифр специальности (сокращение) |
| Years | TINYINT | Срок обучения (лет). Необязательный параметр |
| IDDepartment | INT | Идентификатор кафедры, к которой принадлежит специальность |

Таблица *Specializations* служит для хранения информации о специализациях. Ее структура приведена в таблице 3.17.

**Таблица 3.17 – Характеристика атрибутов таблицы Specializations**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор специализации |
| Title | NVARCHAR(80) | Название специализации |
| Chiper | NVARCHAR(15) | Шифр специализации (сокращение). Необязательный параметр |
| Qualification | NVARCHAR(50) | Квалификация, после окончания |
| IDSpecialty | INT | Идентификатор специальности, к которой принадлежит специализация |

Таблица *ExistingForms* служит для хранения информации о действующих формах обучения по специальностям. Ее структура приведена в таблице 3.18.

**Таблица 3.18 – Характеристика атрибутов таблицы ExistingForms**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор действующей формы обучения |
| IDSpecialty | INT | Идентификатор специальности, на которой действует данная форма обучения |
| Form | NVARCHAR(30) | Название формы обучения |

Таблица *Groups* служит для хранения информации о группах студентов. Ее структура приведена в таблице 3.19.

**Таблица 3.19 – Характеристика атрибутов таблицы Groups**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | INT | Идентификатор группы |
| IDExistingForm | INT | Идентификатор формы обучения, по которой обучается группа |
| IDSpecialization | INT | Идентификатор специализации группы |
| YearOfFoundation | INT | Год создания |
| Title | NVARCHAR(15) | Название группы |

Таблица *ContentTypes* служит для хранения информации о типах содержания документов. Ее структура приведена в таблице 3.20.

**Таблица 3.20 – Характеристика атрибутов таблицы ContentTypes**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| IDСontentType | SMALLINT | Идентификатор типа содержания документа |
| IDAppointment | SMALLINT | Идентификатор типа назначения документа |
| ContentType | NVARCHAR(100) | Тип содержания |
| DocNumber | INT | Год создания |

Таблица *Appointments* служит для хранения информации о типах назначения документов. Ее структура приведена в таблице 3.21.

**Таблица 3.21 – Характеристика атрибутов таблицы Appointments**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | SMALLINT | Идентификатор типа назначения документа |
| Appointment | NVARCHAR(50) | Тип назначения |

Таблица *DocumentForms* служит для хранения информации о формах документов. Ее структура приведена в таблице 3.22.

**Таблица 3.22 – Характеристика атрибутов таблицы DocumentForms**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | SMALLINT | Идентификатор формы документа |
| DocumentForm | NVARCHAR(20) | Форма документа |

Таблица *MergeForms* служит для соединения форм и назначений документов, т.к. каждая форма документа может иметь свои определенные назначения. Ее структура приведена в таблице 3.23.

**Таблица 3.23 – Характеристика атрибутов таблицы MergeForms**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | SMALLINT | Идентификатор соединения |
| IDDocumentForm | SMALLINT | Идентификатор формы документа |
| IDAppointment | SMALLINT | Идентификатор типа назначения |

Таблица *Documents* служит для хранения информации о составляемых документах. Ее структура приведена в таблице 3.24.

**Таблица 3.24 – Характеристика атрибутов таблицы Documents**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | SMALLINT | Идентификатор документа |
| IDDocumentForm | SMALLINT | Идентификатор формы документа |
| IDAppointment | SMALLINT | Идентификатор типа назначения |
| IDContentType | SMALLINT | Идентификатор типа содержания |
| DocNumber | INT | Номер документа |
| DocumentDate | DATE | Дата подписания документа |

Таблица *List* служит для связи документов и студентов, подпадающих под действие этих документов. Ее структура приведена в таблице 3.25.

**Таблица 3.25 – Характеристика атрибутов таблицы List**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| ID | SMALLINT | Идентификатор соединения |
| IDDocument | SMALLINT | Идентификатор документа |
| IDStudent | SMALLINT | Идентификатор студента |

Таблица *AdditionalDocumentData* служит для связи документов и студентов, подпадающих под действие этих документов. Ее структура приведена в таблице 3.26.

**Таблица 3.26 – Характеристика атрибутов таблицы AdditionalDocumentData**

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Тип | Описание |
| IDDocument | INT | Идентификатор данных |
| FromPlace | NVARCHAR(80) | Место откуда был перевод. Необязательный параметр |
| InPlace | NVARCHAR(80) | Место назначения перевода. Необязательный параметр |
| OldNumber | NVARCHAR(15) | Старый номер. Необязательный параметр |
| NewNumber | NVARCHAR(15) | Новый номер. Необязательный параметр |
| OldSurname | NVARCHAR(40) | Старая фамилия. Необязательный параметр |
| NewSurname | NVARCHAR(40) | Новая фамилия. Необязательный параметр |
| Amount | NVARCHAR(50) | Размер поощрения. Необязательный параметр |
| Qualification | NVARCHAR(50) | Присвоенная квалификация. Необязательный параметр |
| Difference | NVARCHAR(150) | Ликвидированная разница. Необязательный параметр |
| Semestr | TINYINT | Номер семестра. Необязательный параметр |
| TempDate | DATE | Дата окончания отпуска. Необязательный параметр |

В спроектированной базе данных реализован контроль целостности и корректности модификаций, путем выполнения триггеров. Так для большинства таблиц в базе данных определены триггеры, которые после удаления главной сущности производят «зачистку» зависимых сущностей, по требуемым правилам. А для таблицы *Documents* определен триггер, который обновляет номер документа в зависимости от типа содержания.

Например, после удаления группы, студенты, входившие в состав этой группы, не удаляются, а лишь их статус устанавливается в значение «не определено» и идентификатор группы становится равным NULL. Далее приводится реализация данного триггера:

CREATE TRIGGER [dbo].[UpdateGroup]

ON [dbo].[Groups]

AFTER DELETE

AS

BEGIN

SET NOCOUNT ON;

UPDATE Students

SET Students.IDGroup=NULL

WHERE Students.IDGroup IN

(

SELECT ID

FROM DELETED

)

END

4 ЗАПИСЬ ВЫРАЖЕНИЙ, УКАЗАННЫХ В ВАРИАНТЕ  
ЗАДАНИЯ ТИПОВ ЗАПРОСОВ НА ЯЗЫКЕ SQL

1. **Запрос, возвращающий список студентов зачисленных на первый курс.**

SELECT

[Extent1].[ID] AS [ID],

[Extent1].[Surname] AS [Surname],

[Extent1].[Name] AS [Name],

[Extent1].[Patronymic] AS [Patronymic],

[Extent1].[Status] AS [Status],

[Extent1].[DateOfBirth] AS [DateOfBirth],

[Extent1].[NumberOfRecordBook] AS [NumberOfRecordBook],

[Extent1].[NumberOfStudetnID] AS [NumberOfStudetnID],

[Extent1].[ForeignLanguage] AS [ForeignLanguage],

[Extent1].[StudyForm] AS [StudyForm],

[Extent1].[Sex] AS [Sex],

[Extent1].[IDGroup] AS [IDGroup],

[Extent1].[IDNationality] AS [IDNationality],

[Extent1].[IDForm] AS [IDForm]

FROM [dbo].[Students] AS [Extent1]

INNER JOIN (SELECT [Extent2].[IDStudent] AS [IDStudent], [Extent4].[Appointment] AS [Appointment], [Extent5].[DocumentDate] AS [DocumentDate1]

FROM [dbo].[List] AS [Extent2]

INNER JOIN [dbo].[Documents] AS [Extent3] ON [Extent2].[IDDocument] = [Extent3].[ID]

INNER JOIN [dbo].[Appointments] AS [Extent4] ON [Extent3].[IDAppointment] = [Extent4].[ID]

LEFT OUTER JOIN [dbo].[Documents] AS [Extent5] ON [Extent2].[IDDocument] = [Extent5].[ID] ) AS [Join3] ON [Extent1].[ID] = [Join3].[IDStudent]

WHERE ([Join3].[Appointment] LIKE N'%зачисление%') AND ((DATEPART (year, [Join3].[DocumentDate1])) = (DATEPART (year, SysDateTime())))

1. **Запрос, возвращающий список студентов отчисленных по результатам зимней экзаменационной сессии.**

SELECT

[Extent1].[ID] AS [ID],

[Extent1].[Surname] AS [Surname],

[Extent1].[Name] AS [Name],

[Extent1].[Patronymic] AS [Patronymic],

[Extent1].[Status] AS [Status],

[Extent1].[DateOfBirth] AS [DateOfBirth],

[Extent1].[NumberOfRecordBook] AS [NumberOfRecordBook],

[Extent1].[NumberOfStudetnID] AS [NumberOfStudetnID],

[Extent1].[ForeignLanguage] AS [ForeignLanguage],

[Extent1].[StudyForm] AS [StudyForm],

[Extent1].[Sex] AS [Sex],

[Extent1].[IDGroup] AS [IDGroup],

[Extent1].[IDNationality] AS [IDNationality],

[Extent1].[IDForm] AS [IDForm]

FROM [dbo].[Students] AS [Extent1]

INNER JOIN (SELECT [Extent2].[IDStudent] AS [IDStudent]

FROM [dbo].[List] AS [Extent2]

INNER JOIN [dbo].[Documents] AS [Extent3] ON [Extent2].[IDDocument] = [Extent3].[ID]

INNER JOIN [dbo].[Appointments] AS [Extent4] ON [Extent3].[IDAppointment] = [Extent4].[ID]

LEFT OUTER JOIN [dbo].[Documents] AS [Extent5] ON [Extent2].[IDDocument] = [Extent5].[ID]

WHERE ([Extent4].[Appointment] LIKE N'%отчисление%') AND ((DATEPART (month, [Extent5].[DocumentDate])) >= 1) AND ((DATEPART (month, [Extent5].[DocumentDate])) <= 2) ) AS [Filter1] ON [Extent1].[ID] = [Filter1].[IDStudent]

1. **Запрос, возвращающий список студентов, переведены из другого ВУЗа, с другого факультета с указанием ликвидации разницы.**

SELECT

[Extent1].[ID] AS [ID],

[Extent1].[Surname] AS [Surname],

[Extent1].[Name] AS [Name],

[Extent1].[Patronymic] AS [Patronymic],

[Extent1].[Status] AS [Status],

[Extent1].[DateOfBirth] AS [DateOfBirth],

[Extent1].[NumberOfRecordBook] AS [NumberOfRecordBook],

[Extent1].[NumberOfStudetnID] AS [NumberOfStudetnID],

[Extent1].[ForeignLanguage] AS [ForeignLanguage],

[Extent1].[StudyForm] AS [StudyForm],

[Extent1].[Sex] AS [Sex],

[Extent1].[IDGroup] AS [IDGroup],

[Extent1].[IDNationality] AS [IDNationality],

[Extent1].[IDForm] AS [IDForm],

[Extent7].[FromPlace] AS [FromPlace],

[Extent7].[InPlace] AS [InPlace],

[Extent7].[Difference] AS [Difference]

FROM [dbo].[Students] AS [Extent1]

INNER JOIN (SELECT [Extent2].[IDDocument] AS [IDDocument], [Extent2].[IDStudent] AS [IDStudent], [Extent4].[ContentType] AS [ContentType1], [Extent6].[ContentType] AS [ContentType2]

FROM [dbo].[List] AS [Extent2]

INNER JOIN [dbo].[Documents] AS [Extent3] ON [Extent2].[IDDocument] = [Extent3].[ID]

LEFT OUTER JOIN [dbo].[ContentTypes] AS [Extent4] ON [Extent3].[IDContentType] = [Extent4].[IDContentType]

LEFT OUTER JOIN [dbo].[Documents] AS [Extent5] ON [Extent2].[IDDocument] = [Extent5].[ID]

LEFT OUTER JOIN [dbo].[ContentTypes] AS [Extent6] ON [Extent5].[IDContentType] = [Extent6].[IDContentType] ) AS [Join4] ON [Extent1].[ID] = [Join4].[IDStudent]

LEFT OUTER JOIN [dbo].[AdditionalDocumentData] AS [Extent7] ON [Join4].[IDDocument] = [Extent7].[IDDocument]

WHERE ([Join4].[ContentType1] LIKE N'%другого факультета%') OR ([Join4].[ContentType2] LIKE N'%другого ВУЗа%')

1. **Запрос, выводящий список студентов, восстановившихся на факультете в текущем учебном году.**

SELECT

[Filter1].[ID1] AS [ID],

[Filter1].[Surname] AS [Surname],

[Filter1].[Name] AS [Name],

[Filter1].[Patronymic] AS [Patronymic],

[Filter1].[Status] AS [Status],

[Filter1].[DateOfBirth] AS [DateOfBirth],

[Filter1].[NumberOfRecordBook] AS [NumberOfRecordBook],

[Filter1].[NumberOfStudetnID] AS [NumberOfStudetnID],

[Filter1].[ForeignLanguage] AS [ForeignLanguage],

[Filter1].[StudyForm] AS [StudyForm],

[Filter1].[Sex] AS [Sex],

[Filter1].[IDGroup] AS [IDGroup],

[Filter1].[IDNationality] AS [IDNationality],

[Filter1].[IDForm] AS [IDForm]

FROM (SELECT [Extent1].[ID] AS [ID1], [Extent1].[Surname] AS [Surname], [Extent1].[Name] AS [Name], [Extent1].[Patronymic] AS [Patronymic], [Extent1].[Status] AS [Status], [Extent1].[DateOfBirth] AS [DateOfBirth], [Extent1].[NumberOfRecordBook] AS [NumberOfRecordBook], [Extent1].[NumberOfStudetnID] AS [NumberOfStudetnID], [Extent1].[ForeignLanguage] AS [ForeignLanguage], [Extent1].[StudyForm] AS [StudyForm], [Extent1].[Sex] AS [Sex], [Extent1].[IDGroup] AS [IDGroup], [Extent1].[IDNationality] AS [IDNationality], [Extent1].[IDForm] AS [IDForm]

FROM [dbo].[Students] AS [Extent1]

LEFT OUTER JOIN [dbo].[ExistingForms] AS [Extent2] ON [Extent1].[IDForm] = [Extent2].[ID]

LEFT OUTER JOIN [dbo].[Specialties] AS [Extent3] ON [Extent2].[IDSpecialty] = [Extent3].[ID]

LEFT OUTER JOIN [dbo].[Departments] AS [Extent4] ON [Extent3].[IDDepartment] = [Extent4].[ID]

INNER JOIN [dbo].[Faculties] AS [Extent5] ON [Extent4].[IDFaculty] = [Extent5].[ID]

WHERE [Extent5].[Title] LIKE N'%факультет информационных технологий%' ) AS [Filter1]

INNER JOIN (SELECT [Extent6].[IDStudent] AS [IDStudent], [Extent7].[DocumentDate] AS [DocumentDate1], [Extent9].[DocumentDate] AS [DocumentDate2]

FROM [dbo].[List] AS [Extent6]

INNER JOIN [dbo].[Documents] AS [Extent7] ON [Extent6].[IDDocument] = [Extent7].[ID]

INNER JOIN [dbo].[Appointments] AS [Extent8] ON [Extent7].[IDAppointment] = [Extent8].[ID]

LEFT OUTER JOIN [dbo].[Documents] AS [Extent9] ON [Extent6].[IDDocument] = [Extent9].[ID]

WHERE [Extent8].[Appointment] LIKE N'%восстановление%' ) AS [Filter2] ON [Filter1].[ID1] = [Filter2].[IDStudent]

WHERE ([Filter2].[DocumentDate2] > '01/09/2011') AND ([Filter2].[DocumentDate1] < GETDATE())

1. **а) Запрос, выводящий список студентов, которые должны выйти из академического отпуска.**

SELECT

[Extent1].[ID] AS [ID],

[Extent1].[Surname] AS [Surname],

[Extent1].[Name] AS [Name],

[Extent1].[Patronymic] AS [Patronymic],

[Extent1].[Status] AS [Status],

[Extent1].[DateOfBirth] AS [DateOfBirth],

[Extent1].[NumberOfRecordBook] AS [NumberOfRecordBook],

[Extent1].[NumberOfStudetnID] AS [NumberOfStudetnID],

[Extent1].[ForeignLanguage] AS [ForeignLanguage],

[Extent1].[StudyForm] AS [StudyForm],

[Extent1].[Sex] AS [Sex],

[Extent1].[IDGroup] AS [IDGroup],

[Extent1].[IDNationality] AS [IDNationality],

[Extent1].[IDForm] AS [IDForm]

FROM [dbo].[Students] AS [Extent1]

INNER JOIN (SELECT [Extent2].[IDStudent] AS [IDStudent], [Extent4].[Appointment] AS [Appointment], [Extent6].[ContentType] AS [ContentType], [Extent7].[TempDate] AS [TempDate]

FROM [dbo].[List] AS [Extent2]

INNER JOIN [dbo].[Documents] AS [Extent3] ON [Extent2].[IDDocument] = [Extent3].[ID]

INNER JOIN [dbo].[Appointments] AS [Extent4] ON [Extent3].[IDAppointment] = [Extent4].[ID]

LEFT OUTER JOIN [dbo].[Documents] AS [Extent5] ON [Extent2].[IDDocument] = [Extent5].[ID]

INNER JOIN [dbo].[ContentTypes] AS [Extent6] ON [Extent5].[IDContentType] = [Extent6].[IDContentType]

INNER JOIN [dbo].[AdditionalDocumentData] AS [Extent7] ON [Extent2].[IDDocument] = [Extent7].[IDDocument] ) AS [Join5] ON [Extent1].[ID] = [Join5].[IDStudent]

WHERE ([Join5].[Appointment] LIKE N'%академический%') AND ([Join5].[ContentType] LIKE N'%выход%') AND ([Join5].[TempDate] > CAST( SysDateTime() AS datetime2))

**б) запрос, выводящий список студентов, которые находятся в академическом отпуске.**

SELECT \*

FROM [dbo].[Students]

WHERE [Status] LIKE N'%в академ. отпуске%'

1. **Запрос, возвращающий список студентов, сменивших фамилию к моменту окончания обучения в ВУЗе.**

SELECT

[Extent1].[ID] AS [ID],

[Extent1].[Surname] AS [Surname],

[Extent1].[Name] AS [Name],

[Extent1].[Patronymic] AS [Patronymic],

[Extent1].[Status] AS [Status],

[Extent1].[DateOfBirth] AS [DateOfBirth],

[Extent1].[NumberOfRecordBook] AS [NumberOfRecordBook],

[Extent1].[NumberOfStudetnID] AS [NumberOfStudetnID],

[Extent1].[ForeignLanguage] AS [ForeignLanguage],

[Extent1].[StudyForm] AS [StudyForm],

[Extent1].[Sex] AS [Sex],

[Extent1].[IDGroup] AS [IDGroup],

[Extent1].[IDNationality] AS [IDNationality],

[Extent1].[IDForm] AS [IDForm]

FROM [dbo].[Students] AS [Extent1]

INNER JOIN (SELECT [Extent2].[IDStudent] AS [IDStudent], [Extent4].[Appointment] AS [Appointment]

FROM [dbo].[List] AS [Extent2]

INNER JOIN [dbo].[Documents] AS [Extent3] ON [Extent2].[IDDocument] = [Extent3].[ID]

INNER JOIN [dbo].[Appointments] AS [Extent4] ON [Extent3].[IDAppointment] = [Extent4].[ID] ) AS [Join2] ON [Extent1].[ID] = [Join2].[IDStudent]

WHERE [Join2].[Appointment] LIKE N'%смена фамилии%'

1. **Запрос, выводящий список группы с отображением студентов, находящихся в академ. отпусках.**

SELECT Students.Surname, Students.Name, Students.Patronymic, Students.Status, Groups.Title

FROM Students INNER JOIN

Groups ON Students.IDGroup = Groups.ID

WHERE Groups.Title LIKE '08-ИТ-2'

1. **Запрос, подсчитывающий кол-во студентов, обучающихся на договорной и бюджетной основе.**

CREATE PROCEDURE GetStudentsCountByForm @form nvarchar(15)

AS

BEGIN

declare @retVal int

set @retVal = (SELECT COUNT(\*) FROM Students WHERE StudyForm=@form)

select @retVal

END

1. **Запрос-процедура, выполняющий объединение двух групп.**

CREATE PROCEDURE MergeGroups @main\_id int, @second\_id int, @spec\_id int, @new\_title nvarchar(15)

AS

BEGIN

SET NOCOUNT ON;

UPDATE Groups

SET Title = @new\_title, IDSpecialization=@spec\_id

WHERE ID=@main\_id

UPDATE Students

SET IDGroup = @main\_id

WHERE IDGroup = @second\_id

DELETE FROM Groups

WHERE Groups.ID=@second\_id

END

1. **а) запрос, возвращающий список всех выданных справок в течение текущего учебного года.**

SELECT DocumentForms.DocumentForm,Students.Surname, Students.Name, Students.Patronymic, Documents.DocumentDate

FROM Documents INNER JOIN

DocumentForms ON Documents.IDDocumentForma = DocumentForms.ID INNER JOIN

List ON Documents.ID = List.IDDocument INNER JOIN

Students ON List.IDStudent = Students.ID

WHERE DocumentForm='справка' AND (Documents.DocumentDate BETWEEN '01/09/2011' AND GETDATE())

**б) запрос, выводящий общее кол-во справок, выданных в текущем учебном году.**

SELECT COUNT(\*)

FROM Documents INNER JOIN

DocumentForms ON Documents.IDDocumentForma = DocumentForms.ID INNER JOIN

List ON Documents.ID = List.IDDocument INNER JOIN

Students ON List.IDStudent = Students.ID

WHERE DocumentForm='справка' AND (Documents.DocumentDate BETWEEN '01/09/2011' AND GETDATE())

1. **Запрос, выводящий список студентов, которым был сделан выговор по университету за текущий семестр.**

SELECT

[EXTENT1].[ID] AS [ID],

[EXTENT1].[SURNAME] AS [SURNAME],

[EXTENT1].[NAME] AS [NAME],

[EXTENT1].[PATRONYMIC] AS [PATRONYMIC],

[EXTENT1].[STATUS] AS [STATUS],

[EXTENT1].[DATEOFBIRTH] AS [DATEOFBIRTH],

[EXTENT1].[NUMBEROFRECORDBOOK] AS [NUMBEROFRECORDBOOK],

[EXTENT1].[NUMBEROFSTUDETNID] AS [NUMBEROFSTUDETNID],

[EXTENT1].[FOREIGNLANGUAGE] AS [FOREIGNLANGUAGE],

[EXTENT1].[STUDYFORM] AS [STUDYFORM],

[EXTENT1].[SEX] AS [SEX],

[EXTENT1].[IDGROUP] AS [IDGROUP],

[EXTENT1].[IDNATIONALITY] AS [IDNATIONALITY],

[EXTENT1].[IDFORM] AS [IDFORM]

FROM [DBO].[STUDENTS] AS [EXTENT1]

INNER JOIN (SELECT [EXTENT2].[IDSTUDENT] AS [IDSTUDENT], [EXTENT5].[DOCUMENTDATE] AS [DOCUMENTDATE1]

FROM [DBO].[LIST] AS [EXTENT2]

INNER JOIN [DBO].[DOCUMENTS] AS [EXTENT3] ON [EXTENT2].[IDDOCUMENT] = [EXTENT3].[ID]

LEFT OUTER JOIN [DBO].[CONTENTTYPES] AS [EXTENT4] ON [EXTENT3].[IDCONTENTTYPE] = [EXTENT4].[IDCONTENTTYPE]

LEFT OUTER JOIN [DBO].[DOCUMENTS] AS [EXTENT5] ON [EXTENT2].[IDDOCUMENT] = [EXTENT5].[ID]

LEFT OUTER JOIN [DBO].[APPOINTMENTS] AS [EXTENT6] ON [EXTENT5].[IDAPPOINTMENT] = [EXTENT6].[ID]

WHERE ([EXTENT3].[DOCUMENTDATE] < (SYSDATETIME())) AND ([EXTENT4].[CONTENTTYPE] LIKE N'%ВЫГОВОР%') ) AS [FILTER1] ON [EXTENT1].[ID] = [FILTER1].[IDSTUDENT]

WHERE [FILTER1].[DOCUMENTDATE1] > '01.09.2011'

1. **а) Запрос, выводящий список студентов, получивших более 3-х не аттестаций в указанном семестре.**

CREATE PROCEDURE GetBadStudents @semestr int

AS

BEGIN

SELECT

[Extent1].[ID] AS [ID],

[Extent1].[Surname] AS [Surname],

[Extent1].[Name] AS [Name],

[Extent1].[Patronymic] AS [Patronymic],

[Extent1].[Status] AS [Status],

[Extent1].[DateOfBirth] AS [DateOfBirth],

[Extent1].[NumberOfRecordBook] AS [NumberOfRecordBook],

[Extent1].[NumberOfStudetnID] AS [NumberOfStudetnID],

[Extent1].[ForeignLanguage] AS [ForeignLanguage],

[Extent1].[StudyForm] AS [StudyForm],

[Extent1].[Sex] AS [Sex],

[Extent1].[IDGroup] AS [IDGroup],

[Extent1].[IDNationality] AS [IDNationality],

[Extent1].[IDForm] AS [IDForm]

FROM [dbo].[Students] AS [Extent1]

INNER JOIN (SELECT [Filter1].[IDDocument], [Filter1].[IDStudent], [Extent5].[DocumentDate] AS [DocumentDate1]

FROM (SELECT [Extent2].[IDDocument] AS [IDDocument], [Extent2].[IDStudent] AS [IDStudent]

FROM [dbo].[List] AS [Extent2]

INNER JOIN [dbo].[Documents] AS [Extent3] ON [Extent2].[IDDocument] = [Extent3].[ID]

INNER JOIN [dbo].[ContentTypes] AS [Extent4] ON [Extent3].[IDContentType] = [Extent4].[IDContentType]

WHERE ([Extent3].[DocumentDate] < (SysDateTime())) AND ([Extent4].[ContentType] LIKE N'%не выполнение аттестационных норм%') ) AS [Filter1]

INNER JOIN [dbo].[Documents] AS [Extent5] ON [Filter1].[IDDocument] = [Extent5].[ID] ) AS [Join3] ON [Extent1].[ID] = [Join3].[IDStudent]

LEFT OUTER JOIN [dbo].[Groups] AS [Extent6] ON [Extent1].[IDGroup] = [Extent6].[ID]

WHERE [Join3].[DocumentDate1] >CAST('01/09/'+CAST (YEAR([Extent6].[YearOfFoundation])+(@semestr/2) as nvarchar) as DATE) AND

[Join3].[DocumentDate1] < CAST('31/12/'+CAST (YEAR([Extent6].[YearOfFoundation])+(@semestr/2) as nvarchar) as DATE)

END

**б) запрос, возвращающий численное соотношение успевающих и неуспевающих студентов по курсу.**

DECLARE @good int;

SET @good = (SELECT

COUNT(\*)

FROM [dbo].[Students] AS [Extent1]

LEFT OUTER JOIN [dbo].[Groups] AS [Extent2] ON [Extent1].[IDGroup] = [Extent2].[ID]

WHERE (([Extent1].[Status] LIKE N'%обучается%') OR ([Extent1].[Status] LIKE N'%отпуске%')) AND (4 = (2012 - [Extent2].[YearOfFoundation])) )

DECLARE @bad int;

SET @bad =(SELECT COUNT(\*)

FROM (SELECT [Extent1].[ID] AS [ID1], [Extent1].[Surname] AS [Surname], [Extent1].[Name] AS [Name], [Extent1].[Patronymic] AS [Patronymic], [Extent1].[Status] AS [Status], [Extent1].[DateOfBirth] AS [DateOfBirth], [Extent1].[NumberOfRecordBook] AS [NumberOfRecordBook], [Extent1].[NumberOfStudetnID] AS [NumberOfStudetnID], [Extent1].[ForeignLanguage] AS [ForeignLanguage], [Extent1].[StudyForm] AS [StudyForm], [Extent1].[Sex] AS [Sex], [Extent1].[IDGroup] AS [IDGroup], [Extent1].[IDNationality] AS [IDNationality], [Extent1].[IDForm] AS [IDForm], [Join2].[IDStudent], [Join2].[ContentType]

FROM [dbo].[Students] AS [Extent1]

INNER JOIN (SELECT [Extent2].[IDStudent] AS [IDStudent], [Extent4].[ContentType] AS [ContentType]

FROM [dbo].[List] AS [Extent2]

INNER JOIN [dbo].[Documents] AS [Extent3] ON [Extent2].[IDDocument] = [Extent3].[ID]

INNER JOIN [dbo].[ContentTypes] AS [Extent4] ON [Extent3].[IDContentType] = [Extent4].[IDContentType] ) AS [Join2] ON [Extent1].[ID] = [Join2].[IDStudent]

WHERE ([Join2].[ContentType] LIKE N'%аттестационных%') AND (([Extent1].[Status] LIKE N'%обучается%') OR ([Extent1].[Status] LIKE N'%отпуске%')) ) AS [Filter1]

LEFT OUTER JOIN [dbo].[Groups] AS [Extent5] ON [Filter1].[IDGroup] = [Extent5].[ID]

WHERE 4 = (2012 - [Extent5].[YearOfFoundation]))

SELECT CAST(@good as nvarchar(4)) +'/'+CAST (@bad as nvarchar(4)) AS [Соотношение успевающих к неуспевающим]

5 ВЫБОР И ОБОСНОВАНИЕ СРЕДСТВ  
РАЗРАБОТКИ ПРИЛОЖЕНИЯ

Для реализации задачи по разработке информационной системы учета контингента студентов университета была выбрана СУБД MS SQL Server 2008. Данная СУБД является одной из наиболее популярных систем управления и обслуживанию баз данных. В ней применяются новейшие разработки и последние достижения в области проектирования, построения и обслуживания баз данных. Так же MS SQL Server стал де факто стандартом источника данных для всех решений Microsoft, что облегчает его интеграцию с различными программными пакетами.

Для создания приложения используется IDE Visual Studio 2010, а я зыком написания приложения был выбран – C#. Данное сочетание позволяет реализовать самые смелые идеи, т.к. предоставляет мощные средства для написания приложений.

Для соединения с базой данных было принято использовать ADO.NET Entity Framework – объектно-ориентированную технологию доступа к данным, являющуюся object-relational mapping (ORM) решением для .NET Framework от Microsoft.

В Entity Framework реляционные таблицы, столбцы и ограничения внешнего ключа логических моделей преобразуются в сущности и связи концептуальных моделей. Это позволяет достичь большей гибкости при определении объектов и оптимизации логической модели. С помощью инструментов модель EDM (сущностная модель данных) формируются расширяемые классы данных, основанные на концептуальной модели.

6 РЕАЛИЗАЦИЯ ЗАКОНЧЕННОГО ПРИЛОЖЕНИЯ,  
РАБОТАЮЩЕГО С СОЗДАННОЙ БАЗОЙ ДАННЫХ

6.1 Разработка и построение интерфейса главной и рабочей форм

Главная форма приложения является объектом класса *MainWindow*, наследуемый от класса Window, определенного в .NET Framework. Создание всех компонентов формы, в частности главного меню, управляющих элементов, окон сообщений, диалогов настроек и выбора файлов, панели с закладками и др., происходит в методе по мере их выхова, соответствующими им конструкторами.

Все основные таблицы для представления данных были выполнены в виде DataGrid, что упрощает понимание и просмотр информации, т.к. она представляется в табличном виде.

Все основные формы и виды приложения выполнены в одном окне на основе страничного интерфейса. Дочерние (вспомогательные) формы выполнены в виде диалоговых окон. Основной упор при проектировании интерфейса приложения был сделан на привлекательность и понятность для конечного пользователя. Поэтому были заменены многие стандартные элементы оформления на доработанные.

При проектировании приложения были учтены все возможные случаи некорректной работы программы, поэтому большинство нештатных ситуаций сопровождается оповещениями с описанием проблемы.

Скриншоты главной и некоторых диалоговых окон представлены в приложении C.

6.2 Построение главного меню и кнопок панели инструментов

Главное меню программы представлено пятью пунктами: СПИСКИ, ДОКУМЕНТЫ, УЧЕТ, НАСТРОЙКИ, О ПРОГРАММЕ. Данные пункты выполнены в виде одной панели, которая реагирует на нажатия пользователя.

Основные операции по работе с тем или иным окном приложения вынесены в нижнюю панель, где располагаются пиктограммы, представляющие собой подобие кнопок. Все они являются объектами класса *CircleIcon*.

Элементы выбора представлены объектами *ComboBox* и *TreeView*.

6.3 Выполнение программного кода в среде Microsoft Visual C#

Опишем работу приложения с базой данных. Все необходимые интерфейсы для работы с базами данных находятся в классе *DatabaseAgent*. Подключение к базе данных начинается с формирования строки подключения и последующим созданием контекста на основе данной строки:

string conn = GetConnection(); [// получение](file:///\\получение) строки подключения путем построения объекта EntityConnectionStringBuilder и дальнейшем преобразовании его в строку

\_database = new ISDatabaseEntities(conn); // создание контекста модели, для последующей работы с базой данных и её сущностями

После подключения для осуществления запросов необходимо проверить достоверность базы данных, если проверка проходит успешно, то будут разблокированы возможности по работе с базой. Для этого используются методы: *DatabaseExist*() и событие *ConnectAction*.

Все запросы по работе с базой данных обращаются к контексту и строятся с помощью LINQ to Entity [2]. В следствии чего упрощается построение запросов, вызов процедур и фильтрация результатов. Пример текста запроса, возвращающего список студентов, со статусом «отчислен» выглядит следующим образом:

DatabaseAgent.Instance.Database.Students

.Where(s=>s.Status.Equals(“отчислен”);

Результат выполненного запроса представляет собой последовательность IQueryable<T>, которую можно назначить в качестве источника данных для таблиц, и прочих элементов управления.

Функционал по работе с отчетами реализован в классе *DocumentsManager*, в котором представлены методы для создания трех видов отчета и печати документов.

Управление информационными сообщениями и уведомлениями происходит в классе MessagesManager, который содержит все основные тексты уведомления и методы по выводу информационных сообщений и сообщений об ошибках.

Все перечисления и неизменяемые наборы данных расположены в классе *Enums*.

Хранение и обработку настроек приложения обеспечивает класс *AppSettings*, который позволяет загржать, сохранять и изменять настройки программы.

ЗАКЛЮЧЕНИЕ

В результате выполненной работы, была создана база данных для учета контингента студентов университета, а также приложение, а так же эффективно работающее с этой базой данных приложение. Полученная комбинация представляет собой информационную систему ВУЗа.

Разработанная база данных удовлетворяет всем требованиям, предъявленным в задании, и позволяет без проблем хранить и извлекать требуемую информацию.

Созданное приложение позволяет упростить работу с информацией для работников университета, позволяя систематизировать всю необходимую информацию. Также приложение упрощает операции по удалению, изменению, добавлению данных. В приложении реализованы запросы, позволяющие пользователю выбрать всю необходимую информацию по заданным критериям, осуществлять поиск данных и формировать отчеты.

Разработанная система реагирует на ошибочный ввод данных, а также способна определять возникающие ошибки и уведомлять об этом пользователя, чтобы в любой момент он знал из-за чего или почему произошла ошибка, и оперативно устранил её.

В процессе выполнения данной курсовой работы были закреплены навыки в программировании на языке C#, проектировании баз данных и реализации их в СУБД MS SQL Server 2008.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

* 1. Шилдт Г. Полный справочник по SQL.: Пер. с англ. – М.: ООО “И.Д. Вильямс”, 2004. – 752 c.: ил.
  2. Раттц Д. LINQ язык интегрированных запросов в C# 2008 для профессионалов. Пер. с англ. – М.: Вильямс, 2008. – 645с.: ил.
  3. Хернандес М., Вьескас Д. SQL-запросы. Практическое руководство.: Пер. с англ. – М.: Лори, 2003. – 473 c.: ил.
  4. Коннолли Т., Бегг К., Базы данных. Проектирование, реализация и сопровождение. Теория и практика.: Пер. с англ. – М.: Вильямс, 2003. – 1500 с.: ил.
  5. Jennings R., Professional ADO.NET 3.5 with LINQ and the Entity Framework.: – New York.: Wrox, 2009. – 560 c.: ил.

ПРИЛОЖЕНИЕ A

(обязательное)

КОНЦЕПТУАЛЬНАЯ СХЕМА БД

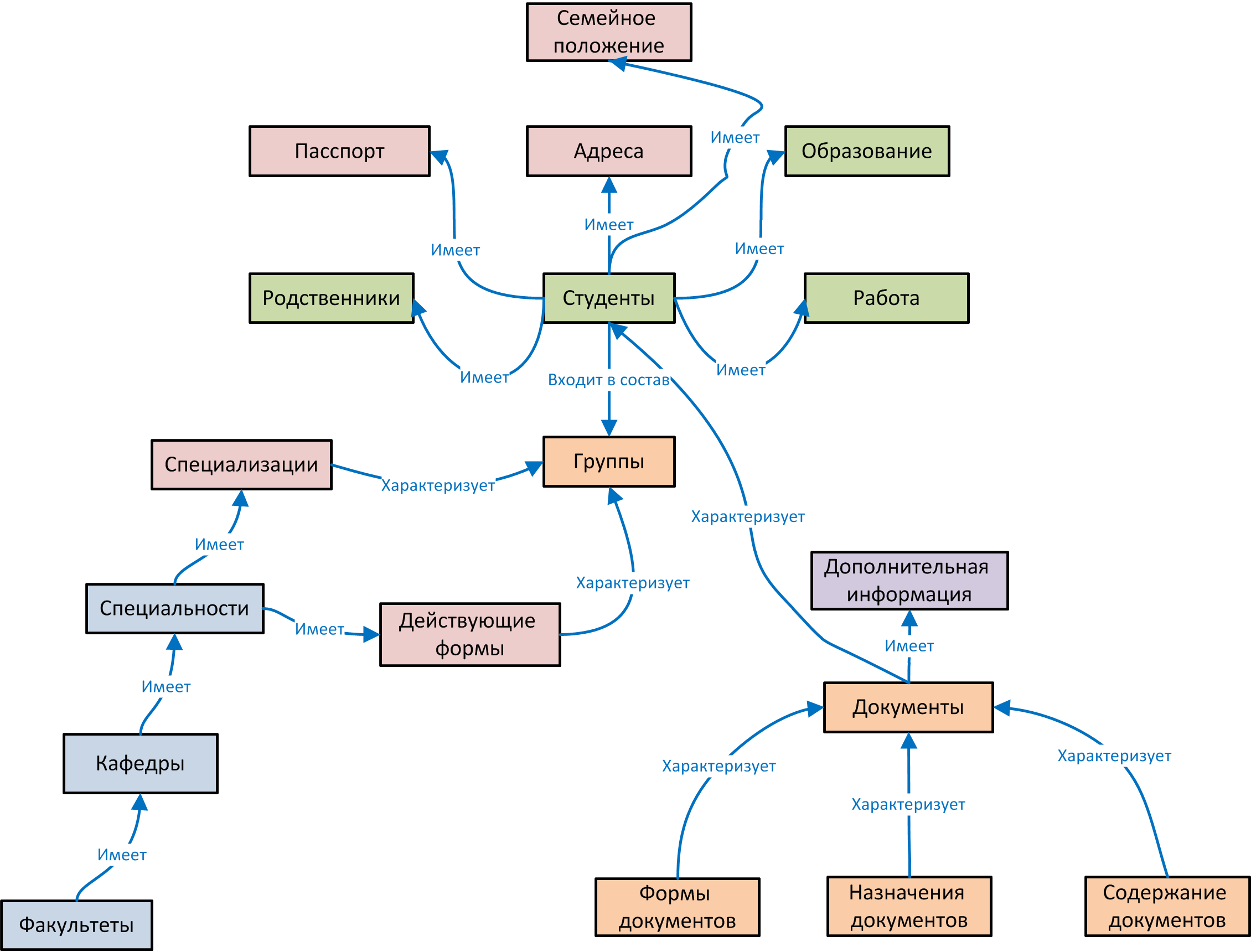


Рисунок A.1 – Инфологическая схема проектируемой базы данных

# 

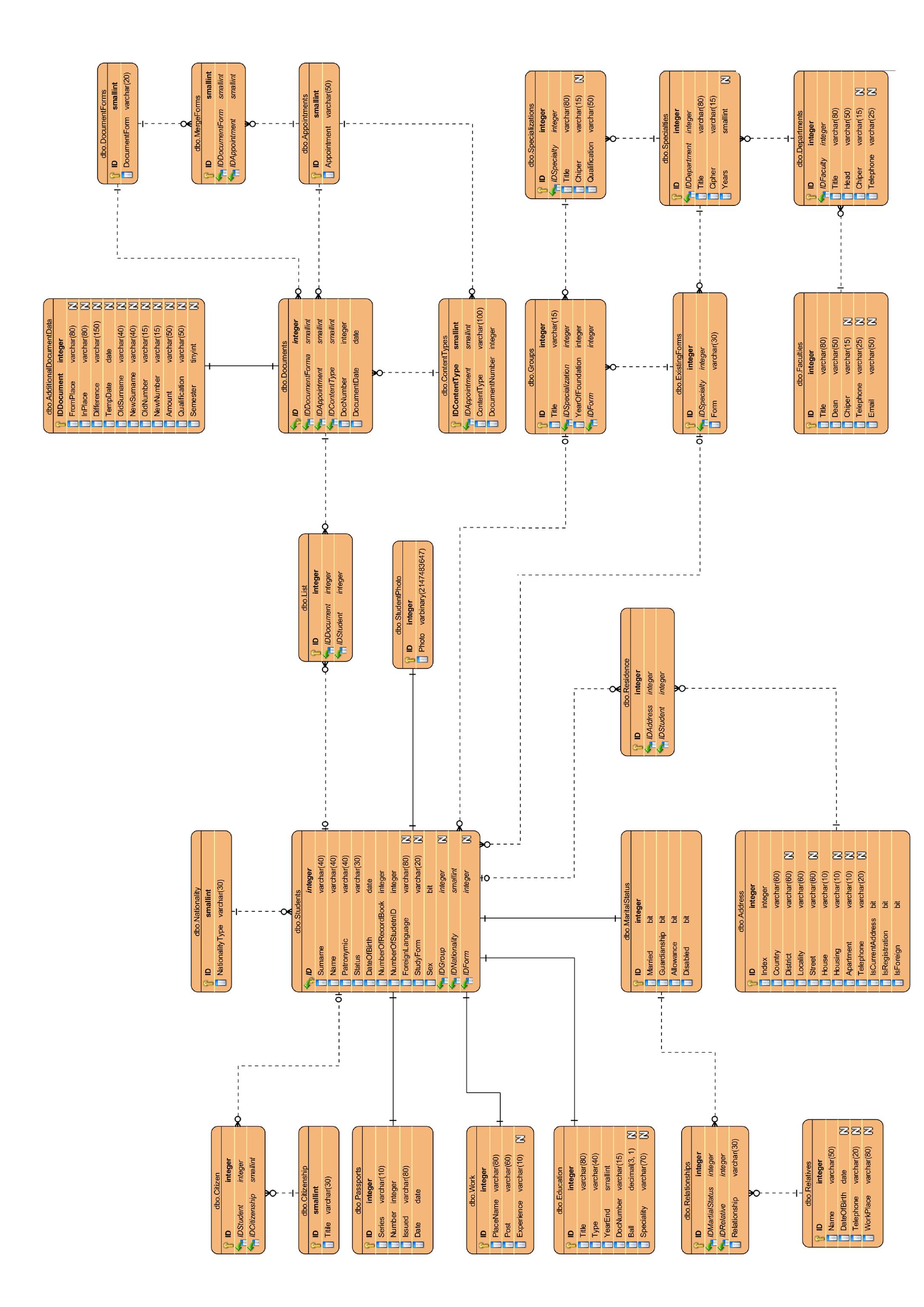


Рисунок A.2 – ER-диаграмма проектируемой базы данных по методу Crow’s Foot

ПРИЛОЖЕНИЕ B

(обязательное)

СХЕМА РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ

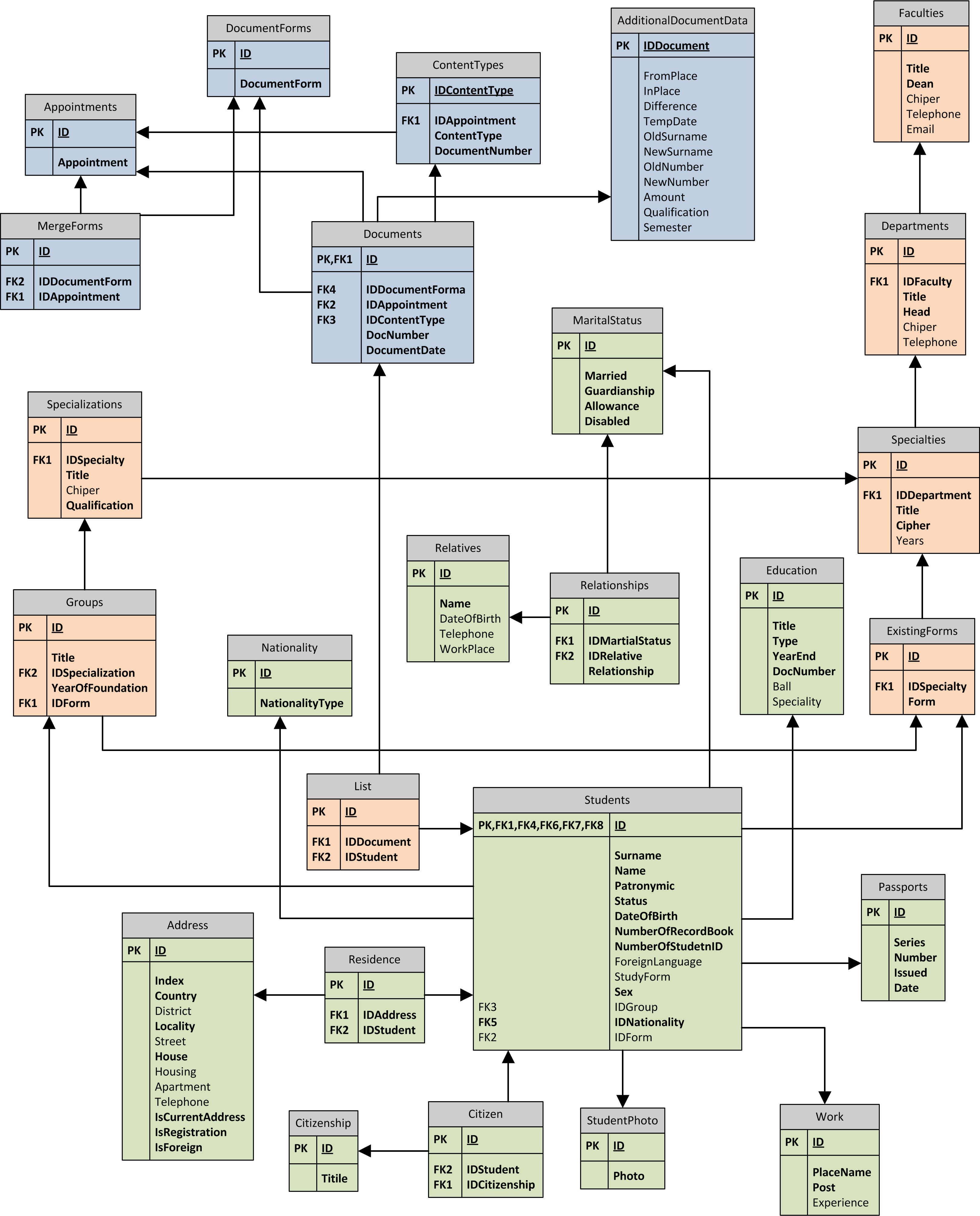


Рисунок B.1 – Схема реляционной базы данных

ПРИЛОЖЕНИЕ C

(обязательное)

ГЛАВНАЯ И РАБОЧИЕ ФОРМЫ ПРИЛОЖЕНИЯ

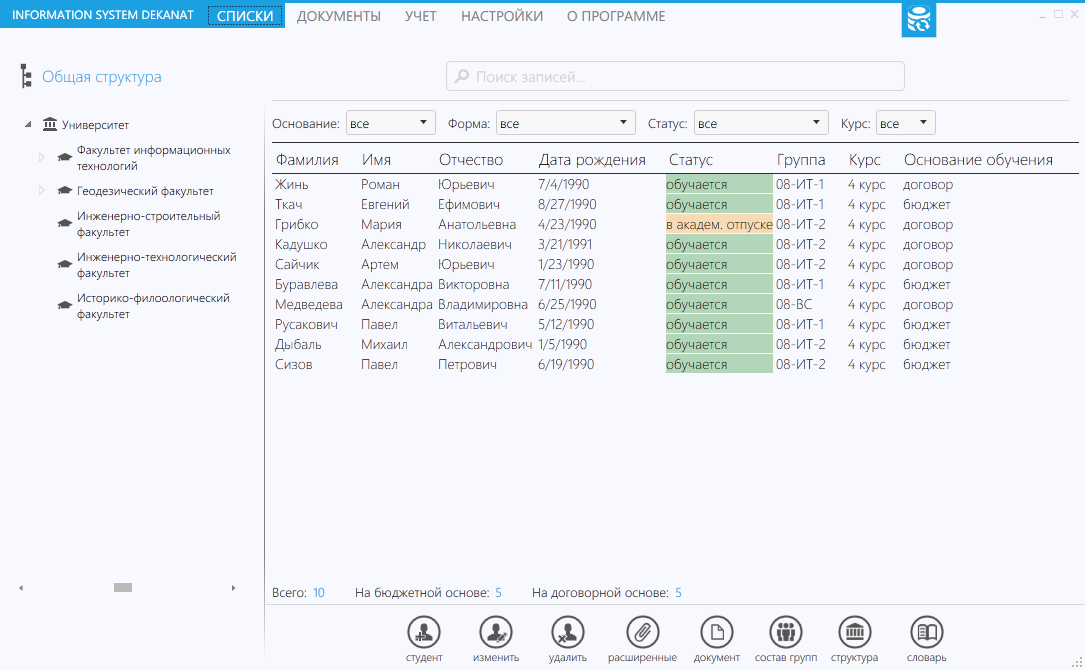


Рисунок C.1 – Главная форма приложения

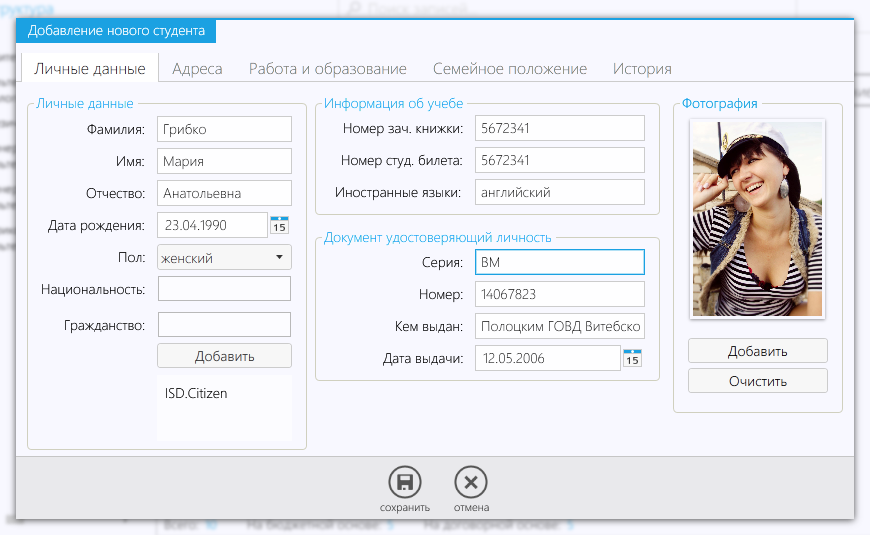


Рисунок C.2 – Диалоговое окно «Добавление нового студента»

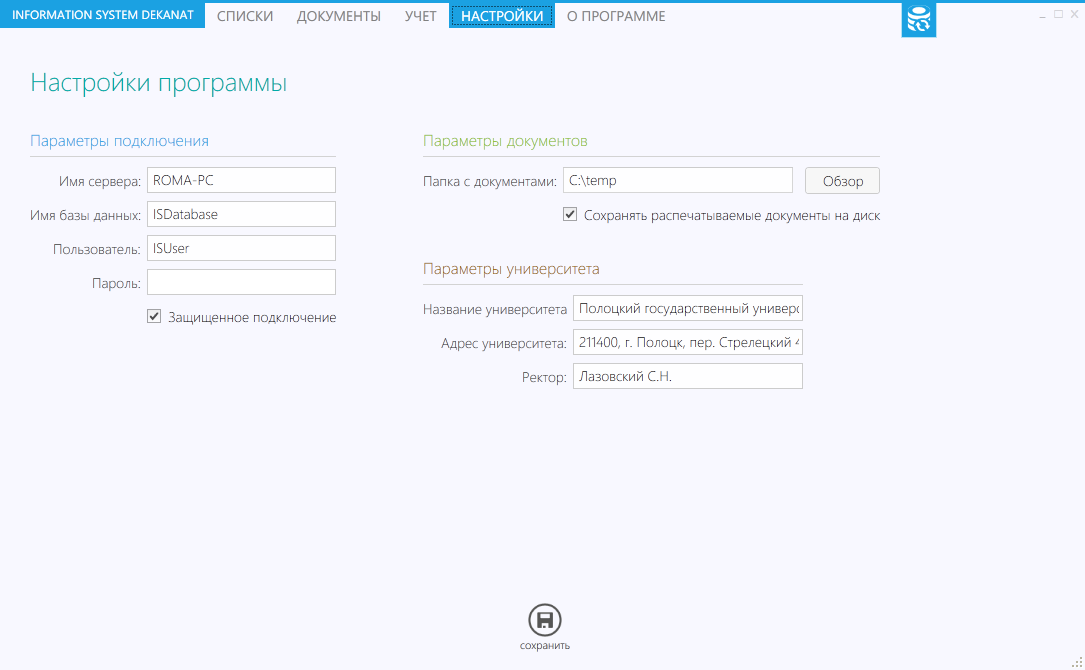


Рисунок C.3 – Страница настроек приложения

ПРИЛОЖЕНИЕ D

(обязательное)

ЛИСТИНГ ПРОГРАММЫ

**Исходный код класса DatabaseAgent**

public class DatabaseAgent

{

public event DatabaseEvents ConnectAction;

ISDatabaseEntities \_database;

public ISDatabaseEntities Database

{

get

{

try

{

return \_database;

}

catch

{

MessageManager.ShowError(Errors.ConnectError, Errors.ConnectErrorTitle);

return null;

}

}

}

public bool isConnected { get; set; }

public DatabaseAgent() { }

private static DatabaseAgent instance;

public static DatabaseAgent Instance

{

get

{

if (instance == null)

instance = new DatabaseAgent();

return instance;

}

}

public void Connect()

{

string conn = GetConnection();

try

{

\_database = new ISDatabaseEntities(conn);

if (\_database.DatabaseExists())

{

isConnected = true;

if (ConnectAction != null) ConnectAction(true);

}

else

{

isConnected = false;

if (ConnectAction != null) ConnectAction(false);

MessageManager.ShowError(Errors.ConnectError, Errors.ConnectErrorTitle);

}

}

catch

{

if (ConnectAction != null) ConnectAction(false);

isConnected = false;

MessageManager.ShowError(Errors.ConnectError, Errors.ConnectErrorTitle);

}

return;

}

private string GetConnection()

{

if (AppSettings.Instance.isConnectionSetFill)

{

string providerName = "System.Data.SqlClient";

string serverName = AppSettings.Instance.ServerName;

string databaseName = AppSettings.Instance.DatabaseName;

SqlConnectionStringBuilder sqlBuilder = new SqlConnectionStringBuilder();

sqlBuilder.DataSource = serverName;

sqlBuilder.InitialCatalog = databaseName;

sqlBuilder.IntegratedSecurity = AppSettings.Instance.SecureConnection;

sqlBuilder.UserID = AppSettings.Instance.UserName;

sqlBuilder.Password = AppSettings.Instance.UserPassword;

sqlBuilder.ConnectTimeout = 3;

string providerString = sqlBuilder.ToString();

EntityConnectionStringBuilder entityBuilder = new EntityConnectionStringBuilder();

entityBuilder.Provider = providerName;

entityBuilder.ProviderConnectionString = providerString;

entityBuilder.Metadata = @"res://\*/UniversityModel.csdl|res://\*/UniversityModel.ssdl|res://\*/UniversityModel.msl";

return entityBuilder.ToString();

}

else return null;

}

public IQueryable<Departments> GetDepartmentsByFaculty(Faculties fac)

{

try

{

return Database.Departments.Where(s => s.IDFaculty == fac.ID);

}

catch

{

return null;

}

}

public IQueryable<Specialties> GetSpecialtiesByDepartment(Departments dep)

{

try

{

return Database.Specialties.Where(s => s.IDDepartment == dep.ID);

}

catch

{

return null;

}

}

public IQueryable<Specializations> GetSpecializationsBySpecialty(Specialties spec)

{

try

{

return Database.Specializations.Where(s => s.IDSpecialty == spec.ID);

}

catch

{

return null;

}

}

public IQueryable<Groups> GetGroupsBySpecizliztioin(Specializations spec)

{

try

{

return Database.Groups.Where(s => s.IDSpecialization == spec.ID);

}

catch

{

return null;

}

}

public ObjectSet<Faculties> FacultiesList

{

get

{

try

{

return Database.Faculties;

}

catch

{

MessageManager.ShowError(Errors.ConnectError, Errors.ConnectErrorTitle);

return null;

}

}

}

public void ChangeStudentsStatus(string newstatus, IEnumerable list)

{

foreach (Students st in list)

{

Instance.Database.Students.First(s => s.ID == st.ID).Status = newstatus;

}

return;

}

public void ChangeStudentsForm(int? newID, IEnumerable list)

{

foreach (Students st in list)

{

Instance.Database.Students.First(s => s.ID == st.ID).IDForm = newID;

}

return;

}

public void ChangeStudentsGroup(int? newID, IEnumerable list)

{

foreach (Students st in list)

{

Instance.Database.Students.First(s => s.ID == st.ID).IDGroup = newID;

}

return;

}

public void ChangeStudentsStudyForm(string newForm, IEnumerable list)

{

foreach (Students st in list)

{

Instance.Database.Students.First(s => s.ID == st.ID).StudyForm = newForm;

}

return;

}

}

**Исходный код класса DocumentsManager**

public class DocumentsManager

{

public DocumentsManager(DocumentType type, Documents document)

{

try

{

Stream stream = null;

if (document.ContentTypes.ContentType.Contains("по месту"))

{

stream = GetResourceTemplate(DocumentType.Certificate);

}

else if (document.ContentTypes.Appointments.Appointment.Contains("группу"))

{

stream = GetResourceTemplate(DocumentType.CommandTransfer);

}

else if (document.ContentTypes.ContentType.Contains("неуспеваемость"))

{

stream = GetResourceTemplate(DocumentType.CommandDisabled);

}

if (stream != null)

{

MemoryStream mem = new MemoryStream();

stream.CopyTo(mem);

stream.Close();

mem.Position = 0;

DocumentStream = mem;

OriginalDoc = document;

if (document.ContentTypes.ContentType.Contains("по месту"))

{

CreateCertificate(document, document.List.First().Students);

}

else if (document.ContentTypes.Appointments.Appointment.Contains("группу"))

{

CreateCommandTransfer(document);

}

else if (document.ContentTypes.ContentType.Contains("неуспеваемость"))

{

CreateCommandDisabled(document);

}

}

}

catch

{

MessageManager.ShowError(Errors.ReportError, Errors.ReportErrorTitle);

}

}

public DocX CurrentDocument

{

get;

private set;

}

public Stream DocumentStream { get; private set; }

Documents OriginalDoc { get; set; }

void CreateCertificate(Documents document, Students stud)

{

if (DocumentStream != null)

{

try

{

DocX docx = DocX.Load(DocumentStream);

if (document != null)

{

docx.ReplaceText("UNIVERSITY\_NAME", AppSettings.Instance.UniversityName);

docx.ReplaceText("UNIVERSITY\_ADDRESS", AppSettings.Instance.UniversityAddress);

docx.ReplaceText("FACULTY\_NAME", stud.ExistingForms.Specialties.Departments.Faculties.Title);

docx.ReplaceText("DOC\_NUMBER", document.DocNumber.ToString());

docx.ReplaceText("DOC\_DATE", document.DocumentDate.ToShortDateString());

docx.ReplaceText("GROUP\_NAME", stud.Groups.Title);

if (stud.ExistingForms.Form.Contains("очная"))

docx

.ReplaceText("STUDY\_FORM", "очной (" + stud.StudyForm + "ной)");

else

docx

.ReplaceText("STUDY\_FORM", "заочной (" + stud.StudyForm + "ной)");

docx.ReplaceText("STUDENT\_FIO", GetStudentRightForm(stud));

docx.ReplaceText("FACULTY\_SKLON", GetFacultyRight(stud.ExistingForms.Specialties.Departments.Faculties.Title));

if (stud.Sex) docx.ReplaceText("что он", " что она");

int year = stud.Groups.YearOfFoundation;

string course = String.Empty;

if (DateTime.Now.Month > 8)

course = ((DateTime.Now.Year - year) + 1).ToString();

else course = (DateTime.Now.Year - year).ToString();

docx.ReplaceText("COURSE\_NUMBER", course);

docx.ReplaceText("DESTINATION", document.ContentTypes.ContentType);

docx.ReplaceText("FACULTY\_CODE", stud.ExistingForms.Specialties.Departments.Faculties.Chiper);

docx.ReplaceText("DEKAN\_FIO", stud.ExistingForms.Specialties.Departments.Faculties.Dean);

}

CurrentDocument = docx;

return;

}

catch

{

CurrentDocument = null;

return;

}

}

CurrentDocument = null;

return;

}

void CreateCommandDisabled(Documents document)

{

if (DocumentStream != null)

{

try

{

DocX docx = DocX.Load(DocumentStream);

if (document != null)

{

Students st = document.List.First().Students;

if (st != null)

{

docx.ReplaceText("UNIVERSITY\_NAME", AppSettings.Instance.UniversityName);

docx.ReplaceText("DOC\_NUMBER", document.DocNumber.ToString());

docx.ReplaceText("DATE", document.DocumentDate.ToShortDateString());

int i = 1;

foreach (List ls in document.List)

{

Docx

.InsertParagraph(i.ToString() + ". " + ls.Students.Surname.ToUpper()

+ " " + ls.Students.Name.ToUpper()

+ " " + ls.Students.Patronymic.ToUpper()).FontSize(14);

}

docx.InsertParagraph().FontSize(14);

if (document.DocumentDate.Month <= 8 || document.DocumentDate.Month >= 6)

docx.InsertParagraph("По результатам летней экзаменационной сессии за академическую неуспеваемость").FontSize(14);

else

docx.InsertParagraph("По результатам зимней экзаменационной сессии за академическую неуспеваемость").FontSize(14);

docx.InsertParagraph().FontSize(14);

docx.InsertParagraph().FontSize(14);

docx.InsertParagraph("Основание: представление декана").FontSize(14);

docx.InsertParagraph().FontSize(14);

docx.InsertParagraph().FontSize(14);

docx.InsertParagraph().FontSize(14);

docx.InsertParagraph().FontSize(14);

docx.InsertParagraph("Ректор университета"

+ " "

+ AppSettings.Instance.UniversityHead).FontSize(14);

}

}

CurrentDocument = docx;

return;

}

catch

{

CurrentDocument = null;

return;

}

}

CurrentDocument = null;

return;

}

void CreateCommandTransfer(Documents document)

{

if (DocumentStream != null)

{

try

{

DocX docx = DocX.Load(DocumentStream);

if (document != null)

{

Students st = document.List.First().Students;

if (st != null)

{

docx.ReplaceText("UNIVERSITY\_NAME", AppSettings.Instance.UniversityName);

docx.ReplaceText("FACULTY\_SKLON", GetFacultyRightTvor(st.ExistingForms.Specialties.Departments.Faculties.Title));

docx.ReplaceText("DOC\_NUMBER", document.DocNumber.ToString());

docx.ReplaceText("DATE", document.DocumentDate.ToShortDateString());

docx.InsertParagraph(String.Format("Перевести в группу {0} из группы {1} следующих студентов:",

document.AdditionalDocumentData.InPlace,

document.AdditionalDocumentData.FromPlace)).FontSize(14);

docx.InsertParagraph().FontSize(14);

foreach (List ls in document.List)

docx.InsertParagraph(" " + ls.Students.Surname

+ " " + ls.Students.Name

+ " " + ls.Students.Patronymic).FontSize(14);

docx.InsertParagraph().FontSize(14);

docx.InsertParagraph().FontSize(14);

docx.InsertParagraph().FontSize(14);

docx.InsertParagraph("Основание: " + document.ContentTypes.ContentType).FontSize(14);

docx.InsertParagraph().FontSize(14);

docx.InsertParagraph().FontSize(14);

docx.InsertParagraph().FontSize(14);

docx.InsertParagraph().FontSize(14);

docx.InsertParagraph("Декан " + GetFacultyRight(st.ExistingForms.Specialties.Departments.Faculties.Title)

+ " " +

st.ExistingForms.Specialties.Departments.Faculties.Dean).FontSize(14);

}

}

CurrentDocument = docx;

return;

}

catch

{

CurrentDocument = null;

return;

}

}

CurrentDocument = null;

return;

}

public Stream GetResourceTemplate(DocumentType type)

{

try

{

Assembly currentAssembly = Assembly.GetExecutingAssembly();

string[] arrResources = currentAssembly.GetManifestResourceNames();

switch (type)

{

case DocumentType.Certificate:

if (arrResources.Count(s => s.Contains("Справка.docx")) > 0)

{

Stream stream = currentAssembly.GetManifestResourceStream(arrResources

.Where(s => s.Contains("Справка.docx")).First());

return stream;

}

break;

case DocumentType.CommandTransfer:

if (arrResources.Count(s => s.Contains("Перевод.docx")) > 0)

{

Stream stream = currentAssembly.GetManifestResourceStream(arrResources

.Where(s => s.Contains("Перевод.docx")).First());

return stream;

}

break;

case DocumentType.CommandDisabled:

if (arrResources.Count(s => s.Contains("Отчисление.docx")) > 0)

{

Stream stream = currentAssembly.GetManifestResourceStream(arrResources

.Where(s => s.Contains("Отчисление.docx")).First());

return stream;

}

break;

}

return null;

}

catch

{

return null;

}

}

public void PrintDoc(DocX doc, bool isSaved)

{

Microsoft.Office.Interop.Word.Application app = new Microsoft.Office.Interop.Word.Application();

app.DisplayAlerts = Microsoft.Office.Interop.Word.WdAlertLevel.wdAlertsNone;

try

{

string save\_path = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "PrintTemp.docx");

doc.SaveAs(save\_path);

object filename = save\_path;

object missingValue = Type.Missing;

Microsoft.Office.Interop.Word.Document document = app.Documents.OpenOld(ref filename,

ref missingValue, ref missingValue,

ref missingValue, ref missingValue, ref missingValue,

ref missingValue, ref missingValue, ref missingValue, ref missingValue);

object myTrue = true;

object myFalse = false;

int dialogResult = app.Dialogs[Microsoft.Office.Interop.Word.WdWordDialog.wdDialogFilePrint].Show(ref missingValue);

if (dialogResult == 1)

{

app.ActiveDocument.PrintOutOld(ref myTrue, ref myFalse, ref missingValue, ref missingValue, ref missingValue, ref missingValue, ref missingValue, ref missingValue, ref missingValue, ref missingValue, ref myFalse, ref missingValue, ref missingValue, ref missingValue);

}

document.Close(ref missingValue, ref missingValue, ref missingValue);

if (isSaved)

doc.SaveAs(Path.Combine(AppSettings.Instance.DocumentsFolder,

OriginalDoc.DocumentForms.DocumentForm + " - "

+ OriginalDoc.DocumentDate.ToShortDateString() + ".docx"));

while (app.BackgroundPrintingStatus > 0)

{

System.Threading.Thread.Sleep(500);

}

app.Quit(ref missingValue, ref missingValue, ref missingValue);

try

{

File.Delete(save\_path);

}

catch

{

}

}

catch

{

MessageManager.ShowError(Errors.WordError, Errors.WordErrorTitle);

return;

}

}

string GetStudentRightForm(Students stud)

{

string retval = String.Empty;

if (stud.Sex)

{

if (stud.Surname.Substring(stud.Surname.Length - 1, 1).Contains("а"))

{

retval = stud.Surname.Substring(0, stud.Surname.Length - 1) + "ой";

}

else if (stud.Surname.Substring(stud.Surname.Length - 2, 2).Contains("ая"))

retval = stud.Surname.Substring(0, stud.Surname.Length - 2) + "ой";

else retval = stud.Surname;

if (stud.Name.Substring(stud.Name.Length - 1).Equals("я") || stud.Name.Substring(stud.Name.Length - 1).Equals("а"))

{

retval += " " + stud.Name.Substring(0, stud.Name.Length - 1) + "е";

}

else retval += " " + stud.Name;

retval +=

" " + stud.Patronymic.Substring(0, stud.Patronymic.Length - 1) + "е";

}

else

{

List<char> glasn = new List<char>()

{ 'а', 'у', 'о', 'е', 'и', 'ю', 'я', 'э', 'ё' };

if (stud.Surname.Substring(stud.Surname.Length - 1, 1).Contains("й"))

{

retval = stud.Surname.Substring(0, stud.Surname.Length - 2) + "ому";

}

else if (glasn.Contains(stud.Surname.Substring(stud.Surname.Length - 1).ToCharArray().First()))

retval = stud.Surname;

else retval = stud.Surname + "у";

if (stud.Name.Substring(stud.Name.Length - 1).Equals("я") || stud.Name.Substring(stud.Name.Length - 1).Equals("а"))

{

retval += " " + stud.Name.Substring(0, stud.Name.Length - 1) + "е";

}

if (stud.Name.Substring(stud.Name.Length - 1).Equals("й"))

{

retval += " " + stud.Name.Substring(0, stud.Name.Length - 1) + "ю";

}

else retval += " " + stud.Name + "у";

if (stud.Patronymic.Substring(stud.Patronymic.Length - 1).Equals("ч"))

{

retval += " " + stud.Patronymic + "у";

}

else retval += " " + stud.Patronymic;

}

return retval;

}

string GetFacultyRight(string faculty\_full)

{

string retval = String.Empty;

string[] word = faculty\_full.Split(' ');

if (word[0].ToLower().Contains("факультет"))

{

retval = "факультета";

foreach (string str in word)

if (!str.ToLower().Contains("факуль"))

retval += " " + str;

}

else

{

retval = word[0].Substring(0, word[0].Length - 2) + "ого" + "факультета";

}

return retval;

}

string GetFacultyRightTvor(string faculty\_full)

{

string retval = String.Empty;

string[] word = faculty\_full.Split(' ');

if (word[0].ToLower().Contains("факультет"))

{

retval = "факультету";

foreach (string str in word)

if (!str.ToLower().Contains("факуль"))

retval += " " + str;

}

else

{

retval = word[0].Substring(0, word[0].Length - 2) + "ому" + "факультету";

}

return retval;

}

public void SaveDoc(DocX doc)

{

SaveFileDialog dlg = new SaveFileDialog();

dlg.CheckPathExists = true;

dlg.DefaultExt = ".docx";

if (dlg.ShowDialog() == true)

{

try

{

doc.SaveAs(dlg.FileName);

}

catch

{

}

}

}

}

**Исходный код класса AppSettings**

[Serializable]

public class AppSettings

{

private AppSettings() { }

public string ServerName { get; set; }

public string DatabaseName { get; set; }

public string UserName { get; set; }

public string UserPassword { get; set; }

public bool SecureConnection { get; set; }

public string UniversityName { get; set; }

public string UniversityAddress { get; set; }

public string UniversityHead { get; set; }

public string DocumentsFolder { get; set; }

public bool isSavedPrinted { get; set; }

private static string SettingsFile { get { return Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "Settings.xml"); } }

private static AppSettings instance;

public bool isConnectionSetFill

{

get

{

return (ServerName != null && DatabaseName != null && UserName != null && UserPassword != null);

}

}

public bool isSaveFill

{

get

{

return (isSavedPrinted && DocumentsFolder != null);

}

}

public static AppSettings Instance

{

get

{

if (instance == null)

instance = AppSettings.LoadSettings();

return instance;

}

}

public static AppSettings LoadSettings()

{

if (!File.Exists(SettingsFile))

return new AppSettings();

using (Stream sr = File.Open(SettingsFile, FileMode.OpenOrCreate, FileAccess.ReadWrite))

{

try

{

XmlSerializer xs = new XmlSerializer(typeof(AppSettings));

object obj = xs.Deserialize(sr);

if (obj is AppSettings)

{

AppSettings app = obj as AppSettings;

return app;

}

}

catch

{

return new AppSettings();

}

}

return new AppSettings();

}

public void SaveSettings()

{

using (Stream sw = File.Open(SettingsFile, FileMode.Create, FileAccess.Write))

{

XmlSerializer xs = new XmlSerializer(typeof(AppSettings));

xs.Serialize(sw, this);

}

}

}

**Исходный код класса ListPage**

public partial class ListPage : Page

{

private bool isAssigned = false;

IQueryable<Students> FiltredCollection { get; set; }

IQueryable<Students> SearchableCollection { get; set; }

TreeViewItem UniverItem { get; set; }

public bool isSearch { get; set; }

public ListPage()

{

InitializeComponent();

this.Loaded += new RoutedEventHandler(ListPage\_Loaded);

}

void ListPage\_Loaded(object sender, RoutedEventArgs e)

{

if (!UniverTree.HasItems)

{

BackgroundWorker work = new BackgroundWorker();

work.DoWork += new DoWorkEventHandler(FillTree);

work.RunWorkerAsync();

}

}

void SetCounts()

{

try

{

AllCount.Text = FiltredCollection.ToList().Count.ToString();

FreeCount.Text = FiltredCollection.Count(s => s.StudyForm.Contains("бюджет")).ToString();

PaidCount.Text = FiltredCollection.Count(s => s.StudyForm.Contains("договор")).ToString();

}

catch { }

}

private void SetFullCollection()

{

StudentsGrid.ItemsSource = null;

FiltredCollection = DatabaseAgent.Instance.Database.Students.AsQueryable();

StudentsGrid.ItemsSource = FiltredCollection;

UniverItem.IsSelected = true;

}

void FillTree(object sender, DoWorkEventArgs e)

{

try

{

this.Dispatcher.BeginInvoke(DispatcherPriority.SystemIdle,

new Action(() =>

{

UniverTree.Items.Clear();

UniverItem = new TreeViewItem();

UniverItem.Header = new TreeItem("Университет", TreeTypes.Root);

UniverItem.IsExpanded = true;

UniverTree.Items.Add(UniverItem);

foreach (Faculties fac in DatabaseAgent.Instance.Database.Faculties.ToList())

{

TreeViewItem itm = new TreeViewItem() { Header = new TreeItem(fac, TreeTypes.Faculty) };

foreach (Departments dep in DatabaseAgent.Instance.Database.Departments.Where(s => s.IDFaculty == fac.ID).ToList())

{

TreeViewItem depitm = new TreeViewItem() { Header = new TreeItem(dep, TreeTypes.Department) };

foreach (Specialties spec in DatabaseAgent.Instance.Database.Specialties.Where(s => s.IDDepartment == dep.ID).ToList())

{

TreeViewItem sitm = new TreeViewItem() { Header = new TreeItem(spec, TreeTypes.Specialty) };

foreach (Specializations spz in DatabaseAgent.Instance.Database.Specializations.Where(s => s.IDSpecialty == spec.ID).ToList())

{

TreeViewItem szitm = new TreeViewItem() { Header = new TreeItem(spz, TreeTypes.Specialization) };

foreach (Groups gr in DatabaseAgent.Instance.Database.Groups.Where(s => s.IDSpecialization == spz.ID).ToList())

{

TreeViewItem gritm = new TreeViewItem() { Header = new TreeItem(gr, TreeTypes.Group) };

szitm.Items.Add(gritm);

}

sitm.Items.Add(szitm);

}

depitm.Items.Add(sitm);

}

itm.Items.Add(depitm);

}

UniverItem.Items.Add(itm);

}

if (!isAssigned)

{

SetFullCollection();

isAssigned = true;

}

}));

}

catch

{

MessageManager.ShowError(Errors.ConnectError, Errors.ConnectErrorTitle);

}

}

private void SetFirstStateList()

{

base\_combo.SelectedIndex = 0;

form\_combo.SelectedIndex = 0;

course\_combo.SelectedIndex = 0;

status\_combo.SelectedIndex = 0;

StudentsGrid.ItemsSource = null;

try

{

FiltredCollection = DatabaseAgent.Instance.Database.Students.AsQueryable();

}

catch { }

StudentsGrid.ItemsSource = FiltredCollection;

SetCounts();

UniverItem.IsSelected = true;

}

private void NewStudentWindow()

{

AddStudentWindow win = new AddStudentWindow();

win.Owner = MainWindow.Instance;

win.ShowDialog();

SetFirstStateList();

}

private void AddNewStructure()

{

AddStructureWindow win = new AddStructureWindow();

win.Owner = MainWindow.Instance;

win.ShowDialog();

MainWindow.Instance.FocusWindow();

FillTree(this, null);

}

private void orderDocument()

{

PreDocumentWindow pre = new PreDocumentWindow();

pre.Owner = MainWindow.Instance;

pre.ShowDialog();

SetFullCollection();

}

private void EditStudent()

{

if (StudentsGrid.SelectedItem != null)

{

AddStudentWindow st = new AddStudentWindow((Students)StudentsGrid.SelectedItem);

st.Owner = MainWindow.Instance;

st.ShowDialog();

SetFullCollection();

SetTreeFilter(this, null);

}

}

private void DeleteStudent()

{

if (StudentsGrid.SelectedItem != null)

{

MessageWindow mess = new MessageWindow(Messages.DeleteStudentMessage, Messages.DeleteStudentTitle);

mess.Owner = MainWindow.Instance;

mess.MessageOK += new Action(mess\_MessageOK);

mess.ShowDialog();

}

}

void mess\_MessageOK()

{

try

{

DatabaseAgent.Instance.Database.Students

.DeleteObject(DatabaseAgent.Instance.Database.Students

.First(s => s.ID == ((Students)StudentsGrid.SelectedItem).ID));

DatabaseAgent.Instance.Database.SaveChanges();

SetFullCollection();

}

catch

{

MessageManager.ShowError(Errors.DeleteError, Errors.DeleteErrorTitle);

}

}

private void MenuEdit(object sender, RoutedEventArgs e)

{

EditStudent();

}

private void MenuDelete(object sender, RoutedEventArgs e)

{

DeleteStudent();

}

private void Deletekey(object sender, KeyEventArgs e)

{

if (e.Key == Key.Delete)

if (StudentsGrid.SelectedItem != null)

DeleteStudent();

}

private void SetListFilter(object sender, SelectionChangedEventArgs e)

{

if (base\_combo != null && form\_combo != null && status\_combo != null && course\_combo != null && UniverTree.SelectedItem != null)

{

if (base\_combo.SelectedIndex == 0 && form\_combo.SelectedIndex == 0 &&

course\_combo.SelectedIndex == 0 && status\_combo.SelectedIndex == 0

&& UniverTree.SelectedItem == UniverItem)

{

SetFullCollection();

}

else if (base\_combo.SelectedIndex == 0 && form\_combo.SelectedIndex == 0 &&

course\_combo.SelectedIndex == 0 && status\_combo.SelectedIndex == 0

&& ((TreeItem)((TreeViewItem)UniverTree.SelectedItem).Header).ContentObject.GetType().ToString().Contains("ISD"))

{

StudentsGrid.ItemsSource = FiltredCollection;

}

else if (FiltredCollection != null && base\_combo != null && form\_combo != null

&& status\_combo != null && course\_combo != null)

{

string s\_base = null, s\_form = null, s\_status = null;

int? course = null;

if (course\_combo.SelectedIndex > 0) course = course\_combo.SelectedIndex;

if (base\_combo.SelectedIndex > 0) s\_base = (string)base\_combo.SelectedItem;

if (form\_combo.SelectedIndex > 0) s\_form = (string)form\_combo.SelectedItem;

if (status\_combo.SelectedIndex > 0) s\_status = (string)status\_combo.SelectedItem;

StudentsGrid.ItemsSource = null;

if (course.HasValue)

{

if (DateTime.Now.Month > 8) course = DateTime.Now.Year - course + 1;

else course = DateTime.Now.Year - course;

}

//0000

if (s\_base == null && s\_form == null && s\_status == null && !course.HasValue)

{

StudentsGrid.ItemsSource = FiltredCollection;

}

//0001

if (s\_base == null && s\_form == null && s\_status == null && course.HasValue)

{

StudentsGrid.ItemsSource = FiltredCollection

.Where(s => s.Groups.YearOfFoundation == course);

}

//0010

else if (s\_base == null && s\_form == null && s\_status != null && !course.HasValue)

{

StudentsGrid.ItemsSource = FiltredCollection

.Where(s => s.Status == s\_status);

}

//0011

else if (s\_base == null && s\_form == null && s\_status != null && course.HasValue)

{

StudentsGrid.ItemsSource = FiltredCollection

.Where(s => s.Status == s\_status && s.Groups.YearOfFoundation == course);

}

//0100

else if (s\_base == null && s\_form != null && s\_status == null && !course.HasValue)

{

StudentsGrid.ItemsSource = FiltredCollection

.Where(s => s.ExistingForms.Form == s\_form);

}

//0101

else if (s\_base == null && s\_form != null && s\_status == null && course.HasValue)

{

StudentsGrid.ItemsSource = FiltredCollection

.Where(s => s.ExistingForms.Form == s\_form && s.Groups.YearOfFoundation == course);

}

//0110

else if (s\_base == null && s\_form != null && s\_status != null && !course.HasValue)

{

StudentsGrid.ItemsSource = FiltredCollection

.Where(s => s.ExistingForms.Form == s\_form && s.Status == s\_status);

}

//0111

else if (s\_base == null && s\_form != null && s\_status != null && course.HasValue)

{

StudentsGrid.ItemsSource = FiltredCollection

.Where(s => s.ExistingForms.Form == s\_form

&& s.Groups.YearOfFoundation == course && s.Status == s\_status);

}

//1000

else if (s\_base != null && s\_form == null && s\_status == null && !course.HasValue)

{

StudentsGrid.ItemsSource = FiltredCollection

.Where(s => s.StudyForm == s\_base);

}

//1001

else if (s\_base != null && s\_form == null && s\_status == null && course.HasValue)

{

StudentsGrid.ItemsSource = FiltredCollection

.Where(s => s.StudyForm == s\_base && s.Groups.YearOfFoundation == course);

}

//1010

else if (s\_base != null && s\_form == null && s\_status != null && !course.HasValue)

{

StudentsGrid.ItemsSource = FiltredCollection

.Where(s => s.StudyForm == s\_base && s.Status == s\_status);

}

//1011

else if (s\_base != null && s\_form == null && s\_status != null && course.HasValue)

{

StudentsGrid.ItemsSource = FiltredCollection

.Where(s => s.StudyForm == s\_base && s.Status == s\_status && s.Groups.YearOfFoundation == course);

}

//1100

else if (s\_base != null && s\_form != null && s\_status == null && !course.HasValue)

{

StudentsGrid.ItemsSource = FiltredCollection

.Where(s => s.StudyForm == s\_base && s.ExistingForms.Form == s\_form);

}

//1101

else if (s\_base != null && s\_form != null && s\_status == null && course.HasValue)

{

StudentsGrid.ItemsSource = FiltredCollection

.Where(s => s.StudyForm == s\_base

&& s.ExistingForms.Form == s\_form && s.Groups.YearOfFoundation == course);

}

//1110

else if (s\_base != null && s\_form != null && s\_status != null && !course.HasValue)

{

StudentsGrid.ItemsSource = FiltredCollection

.Where(s => s.StudyForm == s\_base && s.ExistingForms.Form == s\_form && s.Status == s\_status);

}

//1111

else if (s\_base != null && s\_form != null && s\_status != null && course.HasValue)

{

StudentsGrid.ItemsSource = FiltredCollection

.Where(s => s.StudyForm == s\_base && s.ExistingForms.Form == s\_form

&& s.Status == s\_status && s.Groups.YearOfFoundation == course);

}

}

}

}

private void SetTreeFilter(object sender, RoutedPropertyChangedEventArgs<object> e)

{

if (UniverTree.SelectedItem != null)

{

TreeItem itm = ((TreeViewItem)UniverTree.SelectedItem).Header as TreeItem;

switch (itm.ItemType)

{

case TreeTypes.Root:

SetFullCollection();

SetListFilter(this, null);

break;

case TreeTypes.Faculty:

FiltredCollection = DatabaseAgent.Instance.Database.Students

.Where(s => s.ExistingForms.Specialties.Departments.Faculties.ID == ((Faculties)itm.ContentObject).ID);

if (isSearch) CancelSearch();

StudentsGrid.ItemsSource = FiltredCollection;

SetListFilter(this, null);

break;

case TreeTypes.Department:

FiltredCollection = DatabaseAgent.Instance.Database.Students

.Where(s => s.ExistingForms.Specialties.Departments.ID == ((Departments)itm.ContentObject).ID);

StudentsGrid.ItemsSource = FiltredCollection;

if (isSearch) CancelSearch();

SetListFilter(this, null);

break;

case TreeTypes.Specialty:

FiltredCollection = DatabaseAgent.Instance.Database.Students

.Where(s => s.ExistingForms.Specialties.ID == ((Specialties)itm.ContentObject).ID);

StudentsGrid.ItemsSource = FiltredCollection;

if (isSearch) CancelSearch();

SetListFilter(this, null);

break;

case TreeTypes.Group:

FiltredCollection = DatabaseAgent.Instance.Database.Students

.Where(s => s.Groups.ID == ((Groups)itm.ContentObject).ID);

StudentsGrid.ItemsSource = FiltredCollection;

if (isSearch) CancelSearch();

SetListFilter(this, null);

break;

case TreeTypes.Specialization:

FiltredCollection = DatabaseAgent.Instance.Database.Students

.Where(s => s.Groups.Specializations.ID == ((Specializations)itm.ContentObject).ID);

StudentsGrid.ItemsSource = FiltredCollection;

if (isSearch) CancelSearch();

SetListFilter(this, null);

break;

}

SetCounts();

}

}

private void CancelSearch()

{

SearchPanel.SourceText = String.Empty;

isSearch = false;

}

private void SearchStudent(object sender, TextChangedEventArgs e)

{

if (SearchPanel.SourceText.Length > 0)

{

string[] words = SearchPanel.SourceText.Split(' ');

if (!isSearch)

{

status\_combo.SelectedIndex = 0;

form\_combo.SelectedIndex = 0;

base\_combo.SelectedIndex = 0;

course\_combo.SelectedIndex = 0;

UniverItem.IsSelected = true;

isSearch = true;

}

if (words.Length == 1)

{

string word = words[0];

SearchableCollection = FiltredCollection

.Where(s => s.Name.Contains(word)

|| s.Surname.Contains(word)

|| s.Patronymic.Contains(word));

StudentsGrid.ItemsSource = SearchableCollection;

}

else if (words.Length == 2)

{

string word = words[0], word1 = words[1];

SearchableCollection = FiltredCollection

.Where(s => s.Name.Contains(word)

|| s.Surname.Contains(word)

|| s.Patronymic.Contains(word)

|| s.Name.Contains(word1)

|| s.Surname.Contains(word1)

|| s.Patronymic.Contains(word1));

StudentsGrid.ItemsSource = SearchableCollection;

}

else if (words.Length > 2)

{

string word = words[0], word1 = words[1], word2 = words[2];

SearchableCollection = FiltredCollection

.Where(s => s.Name.Contains(word)

|| s.Surname.Contains(word)

|| s.Patronymic.Contains(word)

|| s.Name.Contains(word1)

|| s.Surname.Contains(word1)

|| s.Patronymic.Contains(word1)

|| s.Name.Contains(word2)

|| s.Surname.Contains(word2)

|| s.Patronymic.Contains(word2));

StudentsGrid.ItemsSource = SearchableCollection;

}

}

else

{

isSearch = false;

StudentsGrid.ItemsSource = FiltredCollection;

}

}

private void FormingGroup()

{

CreateGroupWindow grwin = new CreateGroupWindow();

grwin.Owner = MainWindow.Instance;

grwin.ShowDialog();

SetTreeFilter(this, null);

}

private void ShowAdditional()

{

if (StudentsGrid.SelectedItem != null)

{

SecretWindow sc = new SecretWindow(StudentsGrid.SelectedItem as Students);

sc.Owner = MainWindow.Instance;

MainWindow.Instance.UnfocusWindow();

sc.ShowDialog();

}

}

private void EditStructure(object sender, RoutedEventArgs e)

{

if (UniverTree.SelectedItem != null)

{

TreeItem itm = ((TreeViewItem)UniverTree.SelectedItem).Header as TreeItem;

AddStructureWindow win = null;

switch (itm.ItemType)

{

case TreeTypes.Faculty:

win = new AddStructureWindow(itm.ContentObject as Faculties);

break;

case TreeTypes.Department:

win = new AddStructureWindow(itm.ContentObject as Departments);

break;

case TreeTypes.Group:

win = new AddStructureWindow(itm.ContentObject as Groups);

break;

case TreeTypes.Specialization:

win = new AddStructureWindow(itm.ContentObject as Specializations);

break;

case TreeTypes.Specialty:

win = new AddStructureWindow(itm.ContentObject as Specialties);

break;

}

if (win != null)

{

win.Owner = MainWindow.Instance;

win.ShowDialog();

MainWindow.Instance.FocusWindow();

FillTree(this, null);

}

}

}

private void ShowDictionary()

{

DictionaryWindow dc = new DictionaryWindow();

dc.Owner = MainWindow.Instance;

dc.ShowDialog();

}

private void MenuExtended(object sender, RoutedEventArgs e)

{

if (StudentsGrid.SelectedItem != null)

{

SecretWindow sc = new SecretWindow(StudentsGrid.SelectedItem as Students);

sc.Owner = MainWindow.Instance;

MainWindow.Instance.UnfocusWindow();

sc.ShowDialog();

}

}

private void DeleteStructure(object sender, RoutedEventArgs e)

{

if (UniverTree.SelectedItem != null)

{

MessageWindow msg = new MessageWindow("Выдествительно хотите удалить данный элемент. При этом так же будут удалены все связанные структуры. Хотите продолжить?", "Удаление структуры");

msg.Owner = MainWindow.Instance;

msg.MessageOK += new Action(msg\_MessageOK);

msg.ShowDialog();

}

}

void msg\_MessageOK()

{

if (UniverTree.SelectedItem != null)

{

TreeItem itm = ((TreeViewItem)UniverTree.SelectedItem).Header as TreeItem;

try

{

switch (itm.ItemType)

{

case TreeTypes.Faculty:

DatabaseAgent.Instance.Database.Faculties

.DeleteObject(DatabaseAgent.Instance.Database.Faculties

.First(s => s.ID == ((Faculties)itm.ContentObject).ID));

break;

case TreeTypes.Department:

DatabaseAgent.Instance.Database.Departments

.DeleteObject(DatabaseAgent.Instance.Database.Departments

.First(s => s.ID == ((Departments)itm.ContentObject).ID));

break;

case TreeTypes.Group:

DatabaseAgent.Instance.Database.Groups

.DeleteObject(DatabaseAgent.Instance.Database.Groups

.First(s => s.ID == ((Groups)itm.ContentObject).ID));

break;

case TreeTypes.Specialization:

DatabaseAgent.Instance.Database.Specializations

.DeleteObject(DatabaseAgent.Instance.Database.Specializations

.First(s => s.ID == ((Specializations)itm.ContentObject).ID));

break;

case TreeTypes.Specialty:

DatabaseAgent.Instance.Database.Specialties

.DeleteObject(DatabaseAgent.Instance.Database.Specialties

.First(s => s.ID == ((Specialties)itm.ContentObject).ID));

break;

}

DatabaseAgent.Instance.Database.SaveChanges();

FillTree(this, null);

}

catch

{

MessageManager.ShowError(Errors.DeleteError, Errors.DeleteErrorTitle);

}

}

}

private void TrySort(object sender, MouseButtonEventArgs e)

{

DependencyObject dep = (DependencyObject)e.OriginalSource;

while ((dep != null) &&

!(dep is DataGridCell) &&

!(dep is System.Windows.Controls.Primitives.DataGridColumnHeader))

{

dep = System.Windows.Media.VisualTreeHelper.GetParent(dep);

}

if (dep == null)

return;

if (dep is System.Windows.Controls.Primitives.DataGridColumnHeader)

{

System.Windows.Controls.Primitives.DataGridColumnHeader columnHeader = dep as System.Windows.Controls.Primitives.DataGridColumnHeader;

switch (columnHeader.Column.DisplayIndex)

{

case 0:

StudentsGrid.ItemsSource =

FiltredCollection.OrderByDescending(s => s.Surname);

break;

case 1:

StudentsGrid.ItemsSource =

FiltredCollection.OrderByDescending(s => s.Name);

break;

case 2:

StudentsGrid.ItemsSource =

FiltredCollection.OrderByDescending(s => s.Patronymic);

break;

}

}

}

}

**Исходный код класса DocumentsPage**

public partial class DocumentsPage

{

IQueryable<Documents> DocumentList { get; set; }

BackgroundWorker Worker { get; set; }

public DocumentsPage()

{

this.InitializeComponent();

}

void LoadedInit()

{

try

{

DocumentList = DatabaseAgent.Instance.Database.Documents;

this.Dispatcher.BeginInvoke(DispatcherPriority.Background, new Action(() =>

{

SetTextLabels(DocumentList);

DocumentGrid.ItemsSource = DocumentList;

FormaBox.ItemsSource = DatabaseAgent.Instance.Database.DocumentForms;

}));

}

catch

{

MessageManager.ShowError(Errors.ConnectError, Errors.ConnectErrorTitle);

return;

}

}

void SetTextLabels(IQueryable<Documents> list)

{

var lst = list.ToList();

AllCountText.Text = lst.Count.ToString();

DateTime begindate, enddate;

if (DateTime.Now.Month < 9)

{

begindate = new DateTime(DateTime.Now.Year - 1, 9, 1);

enddate = new DateTime(DateTime.Now.Year, 8, 31);

}

else

{

begindate = new DateTime(DateTime.Now.Year, 9, 1);

enddate = DateTime.Now;

}

ThisCountText.Text = lst

.Where(s => s.DocumentDate >= begindate && s.DocumentDate <= enddate).ToList().Count.ToString();

}

void SetData()

{

SetTextLabels(DocumentList);

DocumentGrid.ItemsSource = DocumentList;

}

private void FormChange(object sender, SelectionChangedEventArgs e)

{

if (FormaBox.SelectedItem != null)

{

AllBox.IsChecked = false;

try

{

AppBox.ItemsSource = DatabaseAgent.Instance.Database.Appointments

.Where(s => s.MergeForms.Select(m => m.IDDocumentForm)

.Contains(((DocumentForms)FormaBox.SelectedItem).ID));

DocumentList = DatabaseAgent.Instance.Database.Documents

.Where(s => s.IDDocumentForma == ((DocumentForms)FormaBox.SelectedItem).ID);

SetData();

}

catch

{

}

}

else

{

AppBox.ItemsSource = null;

SetData();

}

}

private void AppChange(object sender, SelectionChangedEventArgs e)

{

if (AppBox.SelectedItem != null)

{

AllBox.IsChecked = false;

try

{

ContentBox.ItemsSource = DatabaseAgent.Instance.Database.ContentTypes

.Where(s => s.IDAppointment == ((Appointments)AppBox.SelectedItem).ID);

DocumentList = DatabaseAgent.Instance.Database.Documents

.Where(s => s.IDAppointment == ((Appointments)AppBox.SelectedItem).ID

&& s.IDDocumentForma == ((DocumentForms)FormaBox.SelectedItem).ID);

SetData();

}

catch

{

}

}

else

{

ContentBox.ItemsSource = null;

SetData();

}

}

private void ContentChange(object sender, SelectionChangedEventArgs e)

{

if (ContentBox.SelectedItem != null)

{

AllBox.IsChecked = false;

try

{

DocumentList = DatabaseAgent.Instance.Database.Documents

.Where(s => s.IDAppointment == ((Appointments)AppBox.SelectedItem).ID

&& s.IDDocumentForma == ((DocumentForms)FormaBox.SelectedItem).ID

&& s.IDContentType == ((ContentTypes)ContentBox.SelectedItem).IDContentType);

SetData();

}

catch

{

}

}

else SetData();

}

private void SetAllDocuments(object sender, RoutedEventArgs e)

{

try

{

StudentsList.ItemsSource = DatabaseAgent.Instance.Database.Documents.ToList();

DocumentList = DatabaseAgent.Instance.Database.Documents.AsQueryable();

FormaBox.SelectedItem = null;

SetData();

}

catch

{

}

}

private void SetStudentsList(object sender, SelectionChangedEventArgs e)

{

if (DocumentGrid.SelectedItem != null)

{

Documents doc = DocumentGrid.SelectedItem as Documents;

StudentsList.ItemsSource = (DocumentGrid.SelectedItem as Documents).List;

if (doc.ContentTypes.Appointments.Appointment.Contains("другую группу") ||

doc.ContentTypes.ContentType.Contains("по месту") ||

doc.ContentTypes.ContentType.Contains("неуспеваемость"))

{

print\_btn.IsActive = true;

Save\_doc\_btn.IsActive = true;

}

else

{

print\_btn.IsActive = false;

Save\_doc\_btn.IsActive = false;

}

}

else

{

StudentsList.ItemsSource = null;

}

}

private void NewDocument()

{

PreDocumentWindow pr = new PreDocumentWindow();

pr.Owner = MainWindow.Instance;

pr.ShowDialog();

SetAllDocuments(this, null);

}

private void DeleteDocument()

{

if (DocumentGrid.SelectedItem != null)

{

MessageWindow msg = new MessageWindow(Messages.DeleteDocumentMessage, Messages.DeleteDocumentTitle);

msg.Owner = MainWindow.Instance;

msg.MessageOK += new Action(msg\_MessageOK);

msg.ShowDialog();

}

}

void msg\_MessageOK()

{

try

{

if (DocumentGrid.SelectedItem != null)

{

Documents doc = DocumentGrid.SelectedItem as Documents;

DatabaseAgent.Instance.Database.Documents.DeleteObject(doc);

DatabaseAgent.Instance.Database.SaveChanges();

FormChange(this, null);

}

}

catch

{

MessageManager.ShowError(Errors.DeleteError, Errors.DeleteErrorTitle);

return;

}

}

private void Refresh(object sender, RoutedEventArgs e)

{

LoadedInit();

}

private void PrintDocument()

{

if (DocumentGrid.SelectedItem != null)

{

Documents doc = DocumentGrid.SelectedItem as Documents;

if (doc.ContentTypes.Appointments.Appointment.Contains("другую группу"))

{

DocumentsManager manager = new DocumentsManager(DocumentType.CommandTransfer, doc);

manager.PrintDoc(manager.CurrentDocument, AppSettings.Instance.isSaveFill);

}

else if (doc.ContentTypes.ContentType.Contains("по месту"))

{

DocumentsManager manager = new DocumentsManager(DocumentType.Certificate, doc);

manager.PrintDoc(manager.CurrentDocument, AppSettings.Instance.isSaveFill);

}

else if (doc.ContentTypes.ContentType.Contains("неуспеваемость"))

{

DocumentsManager manager = new DocumentsManager(DocumentType.CommandDisabled, doc);

manager.PrintDoc(manager.CurrentDocument, AppSettings.Instance.isSaveFill);

}

}

}

private void SaveDocument()

{

if (DocumentGrid.SelectedItem != null)

{

Documents doc = DocumentGrid.SelectedItem as Documents;

if (doc.ContentTypes.Appointments.Appointment.Contains("другую группу"))

{

DocumentsManager manager = new DocumentsManager(DocumentType.CommandTransfer, doc);

manager.SaveDoc(manager.CurrentDocument);

}

else if (doc.ContentTypes.ContentType.Contains("по месту"))

{

DocumentsManager manager = new DocumentsManager(DocumentType.Certificate, doc);

manager.SaveDoc(manager.CurrentDocument);

}

else if (doc.ContentTypes.ContentType.Contains("неуспеваемость"))

{

DocumentsManager manager = new DocumentsManager(DocumentType.CommandDisabled, doc);

manager.SaveDoc(manager.CurrentDocument);

}

}

}

}

**Исходный код класса StatisticsPage**

public partial class StatisticsPage

{

public class TempData

{

public TempData(){}

public string Name { get; set; }

public string Surname { get; set; }

public string Patronymic { get; set; }

public DateTime DateOfBirth { get; set; }

public string Status { get; set; }

public Groups Group { get; set; }

public Documents Doc { get; set; }

}

IQueryable<TempData> StudentsList { get; set; }

IQueryable<TempData> FiltredDate { get; set; }

BackgroundWorker Worker { get; set; }

public StatisticsPage()

{

this.InitializeComponent();

}

void Worker\_DoWork(object sender, DoWorkEventArgs e)

{

try

{

StudentsList = DatabaseAgent.Instance.Database.Students.Join(DatabaseAgent.Instance.Database.List, s => s.ID, t => t.IDStudent, (s, t) =>

new TempData

{

Name = s.Name,

Surname = s.Surname,

Patronymic = s.Patronymic,

DateOfBirth = s.DateOfBirth,

Status = s.Status,

Group = s.Groups,

Doc = t.Documents

});

this.Dispatcher.BeginInvoke(DispatcherPriority.Background,

new Action(() =>

{

SetTextLabels(StudentsList);

StudentsGrid.ItemsSource = StudentsList;

MethodChange(this, null);

}));

}

catch

{

MessageManager.ShowError(Errors.ConnectError, Errors.ConnectErrorTitle);

return;

}

}

void SetTextLabels(IQueryable<TempData> list)

{

List<TempData> lst = list.ToList();

AllCountText.Text = lst.Count.ToString();

DateTime begindate, enddate;

if (DateTime.Now.Month < 9)

{

begindate = new DateTime(DateTime.Now.Year - 1, 9, 1);

enddate = new DateTime(DateTime.Now.Year, 8, 31);

}

else

{

begindate = new DateTime(DateTime.Now.Year, 9, 1);

enddate = DateTime.Now;

}

ThisCountText.Text = lst

.Where(t => t.Doc.DocumentDate >= begindate && t.Doc.DocumentDate <= enddate).ToList().Count.ToString();

}

void SetData(IQueryable<TempData> list)

{

SetTextLabels(list);

StudentsGrid.ItemsSource = list;

}

private void MethodChange(object sender, SelectionChangedEventArgs e)

{

if (MethodBox.SelectedItem != null)

{

ThisYearBox.IsChecked = false;

ThisSemestrBox.IsChecked = false;

StatBox.IsChecked = false;

try

{

if (MethodBox.SelectedIndex == 0)

{

StudentsList = DatabaseAgent.Instance.Database.Students

.Join(DatabaseAgent.Instance.Database.List, s => s.ID, t => t.IDStudent, (s, t) => new TempData

{

Name = s.Name,

Surname = s.Surname,

Patronymic = s.Patronymic,

DateOfBirth = s.DateOfBirth,

Status = s.Status,

Group = s.Groups,

Doc = t.Documents

});

}

else if (MethodBox.SelectedIndex == 1)

{

StudentsList = DatabaseAgent.Instance.Database.Students

.Join(DatabaseAgent.Instance.Database.List

.Where(l => l.Documents.Appointments.Appointment.Contains("зачисление")),

s => s.ID,

t => t.IDStudent,

(s, t) => new TempData

{

Name = s.Name,

Surname = s.Surname,

Patronymic = s.Patronymic,

DateOfBirth = s.DateOfBirth,

Status = s.Status,

Group = s.Groups,

Doc = t.Documents

});

}

else if (MethodBox.SelectedIndex == 2)

{

StudentsList = DatabaseAgent.Instance.Database.Students

.Join(DatabaseAgent.Instance.Database.List

.Where(l => l.Documents.Appointments.Appointment.Contains("отчисление")),

s => s.ID,

t => t.IDStudent,

(s, t) => new TempData

{

Name = s.Name,

Surname = s.Surname,

Patronymic = s.Patronymic,

DateOfBirth = s.DateOfBirth,

Status = s.Status,

Group = s.Groups,

Doc = t.Documents

});

}

else if (MethodBox.SelectedIndex == 3)

{

StudentsList = DatabaseAgent.Instance.Database.Students

.Join(DatabaseAgent.Instance.Database.List

.Where(l => l.Documents.ContentTypes.ContentType.Contains("другого факультета")),

s => s.ID,

t => t.IDStudent,

(s, t) => new TempData

{

Name = s.Name,

Surname = s.Surname,

Patronymic = s.Patronymic,

DateOfBirth = s.DateOfBirth,

Status = s.Status,

Group = s.Groups,

Doc = t.Documents

});

}

else if (MethodBox.SelectedIndex == 4)

{

StudentsList = DatabaseAgent.Instance.Database.Students

.Join(DatabaseAgent.Instance.Database.List

.Where(l => l.Documents.ContentTypes.ContentType.Contains("другого ВУЗа")),

s => s.ID,

t => t.IDStudent,

(s, t) => new TempData

{

Name = s.Name,

Surname = s.Surname,

Patronymic = s.Patronymic,

DateOfBirth = s.DateOfBirth,

Status = s.Status,

Group = s.Groups,

Doc = t.Documents

});

}

else if (MethodBox.SelectedIndex == 5)

{

StudentsList = DatabaseAgent.Instance.Database.Students

.Join(DatabaseAgent.Instance.Database.List

.Where(l => l.Documents.Appointments.Appointment.Contains("восстановление")),

s => s.ID,

t => t.IDStudent,

(s, t) => new TempData

{

Name = s.Name,

Surname = s.Surname,

Patronymic = s.Patronymic,

DateOfBirth = s.DateOfBirth,

Status = s.Status,

Group = s.Groups,

Doc = t.Documents

});

}

else if (MethodBox.SelectedIndex == 6)

{

StudentsList = DatabaseAgent.Instance.Database.Students

.Join(DatabaseAgent.Instance.Database.List

.Where(l => l.Documents.Appointments.Appointment.Contains("смена фамилии")),

s => s.ID,

t => t.IDStudent,

(s, t) => new TempData

{

Name = s.Name,

Surname = s.Surname,

Patronymic = s.Patronymic,

DateOfBirth = s.DateOfBirth,

Status = s.Status,

Group = s.Groups,

Doc = t.Documents

});

}

else if (MethodBox.SelectedIndex == 7)

{

StudentsList = DatabaseAgent.Instance.Database.Students

.Join(DatabaseAgent.Instance.Database.List

.Where(l => l.Documents.ContentTypes.ContentType.Contains("выговор")),

s => s.ID,

t => t.IDStudent,

(s, t) => new TempData

{

Name = s.Name,

Surname = s.Surname,

Patronymic = s.Patronymic,

DateOfBirth = s.DateOfBirth,

Status = s.Status,

Group = s.Groups,

Doc = t.Documents

});

}

else if (MethodBox.SelectedIndex == 8)

{

StudentsList = DatabaseAgent.Instance.Database.Students.Where(s => s.Status.Contains("отпуске"))

.Join(DatabaseAgent.Instance.Database.List

.Where(l => l.Documents.Appointments.Appointment.Contains("академический")),

s => s.ID,

t => t.IDStudent,

(s, t) => new TempData

{

Name = s.Name,

Surname = s.Surname,

Patronymic = s.Patronymic,

DateOfBirth = s.DateOfBirth,

Status = s.Status,

Group = s.Groups,

Doc = t.Documents

});

}

else if (MethodBox.SelectedIndex == 9)

{

StudentsList = DatabaseAgent.Instance.Database.Students

.Join(DatabaseAgent.Instance.Database.List

.Where(l => l.Documents.Appointments.Appointment.Contains("академический")

&& l.Documents.ContentTypes.ContentType.Contains("выход")),

s => s.ID,

t => t.IDStudent,

(s, t) => new TempData

{

Name = s.Name,

Surname = s.Surname,

Patronymic = s.Patronymic,

DateOfBirth = s.DateOfBirth,

Status = s.Status,

Group = s.Groups,

Doc = t.Documents

});

}

else if (MethodBox.SelectedIndex == 10)

{

StudentsList = DatabaseAgent.Instance.Database.Students

.Join(DatabaseAgent.Instance.Database.List

.Where(l => l.Documents.Appointments.Appointment.Contains("предупреждение")),

s => s.ID,

t => t.IDStudent,

(s, t) => new TempData

{

Name = s.Name,

Surname = s.Surname,

Patronymic = s.Patronymic,

DateOfBirth = s.DateOfBirth,

Status = s.Status,

Group = s.Groups,

Doc = t.Documents

});

}

ChangeDate(this, null);

}

catch

{

}

}

else

{

ChangeDate(this, null);

}

}

private void ChangeDate(object sender, SelectionChangedEventArgs e)

{

try

{

if (BeginDate.SelectedDate != null && EndDate.SelectedDate != null)

{

FiltredDate = StudentsList

.Where(s => s.Doc.DocumentDate >= BeginDate.SelectedDate && s.Doc.DocumentDate <= EndDate.SelectedDate);

}

else if (BeginDate.SelectedDate == null && EndDate.SelectedDate != null)

{

FiltredDate = StudentsList

.Where(s => s.Doc.DocumentDate <= EndDate.SelectedDate);

}

else if (BeginDate.SelectedDate != null && EndDate.SelectedDate == null)

{

FiltredDate = StudentsList

.Where(s => s.Doc.DocumentDate >= BeginDate.SelectedDate);

}

else

{

FiltredDate = StudentsList;

}

SetData(FiltredDate);

}

catch

{

MessageManager.ShowError(Errors.ConnectError, Errors.ConnectErrorTitle);

}

}

private void FilterByCurrentYear(object sender, RoutedEventArgs e)

{

DateTime begindate, enddate;

ThisSemestrBox.IsChecked = false;

StatBox.IsChecked = false;

if (DateTime.Now.Month < 8)

{

begindate = new DateTime(DateTime.Now.Year - 1, 8, 1);

enddate = new DateTime(DateTime.Now.Year, 7, 31);

}

else

{

begindate = new DateTime(DateTime.Now.Year, 8, 1);

enddate = DateTime.Now;

}

StudentsGrid.ItemsSource = FiltredDate

.Where(s => s.Doc.DocumentDate >= begindate && s.Doc.DocumentDate <= enddate);

}

private void FilterByCurrentSemestr(object sender, RoutedEventArgs e)

{

DateTime begindate, enddate;

ThisYearBox.IsChecked = false;

StatBox.IsChecked = false;

if (DateTime.Now.Month < 9 && DateTime.Now.Month > 1)

{

begindate = new DateTime(DateTime.Now.Year, 2, 1);

enddate = new DateTime(DateTime.Now.Year, 8, 31);

}

else

{

begindate = new DateTime(DateTime.Now.Year, 9, 1);

enddate = DateTime.Now;

}

StudentsGrid.ItemsSource = FiltredDate

.Where(s => s.Doc.DocumentDate >= begindate && s.Doc.DocumentDate <= enddate);

}

private void CalculateStat(object sender, RoutedEventArgs e)

{

StudentsGrid.Visibility = Visibility.Hidden;

StatChart.Visibility = Visibility.Visible;

StatChart.DataContext = GetStatInfo();

}

List<KeyValuePair<string, int>> GetStatInfo()

{

List<KeyValuePair<string, int>> valueList = new List<KeyValuePair<string, int>>();

try

{

int minus\_year;

if (DateTime.Now.Month < 9) minus\_year = DateTime.Now.Year;

else minus\_year = DateTime.Now.Year + 1;

for (int i = 1; i < 6; i++)

{

int val = DatabaseAgent.Instance.Database.Students

.Count(s => (s.Status.Contains("обучается") || s.Status.Contains("отпуске"))

&& minus\_year - s.Groups.YearOfFoundation == i);

valueList.Add(new KeyValuePair<string, int>(i.ToString() + " курс(усп.)", val));

//первый курс неуспевающие

val = DatabaseAgent.Instance.Database.Students

.Join(DatabaseAgent.Instance.Database.List

.Where(s => s.Documents.ContentTypes.ContentType.Contains("аттестационных")),

s => s.ID, t => t.IDStudent, (s, t) => s)

.Count(s => (s.Status.Contains("обучается") || s.Status.Contains("отпуске"))

&& minus\_year - s.Groups.YearOfFoundation == i);

valueList.Add(new KeyValuePair<string, int>(i.ToString() + " курс(неусп.)", val));

}

return valueList;

}

catch

{

MessageManager.ShowError(Errors.ConnectError, Errors.ConnectErrorTitle);

return null;

}

}

private void hideStat(object sender, RoutedEventArgs e)

{

StudentsGrid.Visibility = Visibility.Visible;

StatChart.Visibility = Visibility.Hidden;

}

private void Init(object sender, RoutedEventArgs e)

{

Worker = new BackgroundWorker();

Worker.DoWork += new DoWorkEventHandler(Worker\_DoWork);

Worker.RunWorkerAsync();

}

}

**Исходный код класса AddStudentWindow**

public partial class AddStudentWindow : Window

{

Students CurrentStudent { get; set; }

bool EditMode;

CurrentIDs DatabseIDs;

List<Relationships> DeletedRelationships { get; set; }

List<Relationships> AddedRelationships { get; set; }

List<Citizen> DeletedCitizen { get; set; }

List<Citizen> AddedCitizen { get; set; }

List<Citizen> Countries { get; set; }

List<Relationships> Relation { get; set; }

private BitmapImage BImage;

public AddStudentWindow()

{

this.InitializeComponent();

MainWindow.Instance.UnfocusWindow();

EditMode = false;

Countries = new List<Citizen>();

Relation = new List<Relationships>();

RelativeList.ItemsSource = Relation;

CountryList.ItemsSource = Countries;

}

public AddStudentWindow(Students stud)

{

this.InitializeComponent();

MainWindow.Instance.UnfocusWindow();

EditMode = true;

CurrentStudent = stud;

BackgroundWorker wk = new BackgroundWorker();

wk.DoWork += new DoWorkEventHandler(SetEditData);

wk.RunWorkerAsync();

DeletedCitizen = new List<Citizen>();

DeletedRelationships = new List<Relationships>();

AddedRelationships = new List<Relationships>();

AddedCitizen = new List<Citizen>();

}

private void SetEditData(object sender, DoWorkEventArgs e)

{

this.Dispatcher.BeginInvoke(DispatcherPriority.Background, new Action(() =>

{

this.StudentName.Text = CurrentStudent.Name;

this.StudentSurname.Text = CurrentStudent.Surname;

this.StudentPatron.Text = CurrentStudent.Patronymic;

this.Nationality.SelectedItem = CurrentStudent.Nationality;

this.DateBirth.SelectedDate = CurrentStudent.DateOfBirth;

this.Sex.SelectedIndex = CurrentStudent.Sex ? 1 : 0;

this.Countries = CurrentStudent.Citizen.ToList();

this.CountryList.ItemsSource = Countries;

this.NumberRecord.Text = CurrentStudent.NumberOfRecordBook.ToString();

this.NUmberBilet.Text = CurrentStudent.NumberOfStudetnID.ToString();

this.Languages.Text = CurrentStudent.ForeignLanguage;

this.PSerie.Text = CurrentStudent.Passports.Series;

this.PNumber.Text = CurrentStudent.Passports.Number.ToString();

this.PWho.Text = CurrentStudent.Passports.Issued;

this.PDate.Text = CurrentStudent.Passports.Date.ToShortDateString();

if (CurrentStudent.StudentPhoto != null)

this.StudentImage.Source = ImageFromBytes(CurrentStudent.StudentPhoto.Photo);

List<Residence> res = CurrentStudent.Residence.ToList();

Address adr = res[0].Address;

this.ACountry.Text = adr.Country;

this.ALocality.Text = adr.Locality;

this.ADistrict.Text = adr.District;

this.AHouse.Text = adr.House;

this.AIndex.Text = adr.Index.ToString();

this.AKorpus.Text = adr.Housing;

this.AApartment.Text = adr.Apartment;

this.AStreet.Text = adr.Street;

this.APhone.Text = adr.Telephone;

this.AisForeignBox.IsChecked = adr.IsForeign;

this.AisRegistrationBox.IsChecked = adr.IsRegistration;

if (res.Count > 1)

{

Address addr = res[1].Address;

this.ARCountry.Text = addr.Country;

this.ARLocality.Text = addr.Locality;

this.ARDistrict.Text = addr.District;

this.ARHouse.Text = addr.House;

this.ARIndex.Text = addr.Index.ToString();

this.ARKorpus.Text = addr.Housing;

this.ARApartment.Text = addr.Apartment;

this.ARStreet.Text = addr.Street;

this.ARPhone.Text = addr.Telephone;

this.ARisForeign.IsChecked = addr.IsForeign;

}

this.EName.Text = CurrentStudent.Education.Title;

this.ENumber.Text = CurrentStudent.Education.DocNumber;

this.ESpeciality.Text = CurrentStudent.Education.Speciality;

this.EType.SelectedItem = CurrentStudent.Education.Type;

this.EYear.Text = CurrentStudent.Education.YearEnd.ToString();

if (CurrentStudent.Education.Ball.HasValue)

this.EBall.Text = CurrentStudent.Education.Ball.Value.ToString();

if (CurrentStudent.Work != null)

{

this.WExperience.Text = CurrentStudent.Work.Experience;

this.WPlace.Text = CurrentStudent.Work.PlaceName;

this.WPost.Text = CurrentStudent.Work.Post;

}

this.FisInvalid.IsChecked = CurrentStudent.MaritalStatus.Disabled;

this.FisMarried.IsChecked = CurrentStudent.MaritalStatus.Married;

this.FisOpeka.IsChecked = CurrentStudent.MaritalStatus.Guardianship;

this.FisPosob.IsChecked = CurrentStudent.MaritalStatus.Allowance;

this.Relation = CurrentStudent.MaritalStatus.Relationships.ToList();

this.RelativeList.ItemsSource = Relation;

}));

}

private BitmapImage ImageFromBytes(byte[] buff)

{

BImage = new BitmapImage();

BImage.BeginInit();

MemoryStream ms = new MemoryStream(buff);

BImage.StreamSource = ms;

BImage.EndInit();

return BImage;

}

private void ImageDrop(object sender, DragEventArgs e)

{

if (e.Data.GetDataPresent(DataFormats.FileDrop))

{

string[] fileNames = e.Data.GetData(DataFormats.FileDrop, true) as string[];

BImage = new BitmapImage();

BImage.BeginInit();

BImage.StreamSource = File.Open(fileNames[0], FileMode.Open, FileAccess.Read, FileShare.Read);

BImage.EndInit();

StudentImage.Source = BImage;

}

e.Handled = true;

}

private void CloseWindow()

{

MainWindow.Instance.FocusWindow();

this.Close();

}

private Address GetSecondAddressInfo()

{

bool isDamaged = false;

if (ARCountry.Text.Length > 0 && ARLocality.Text.Length > 0 && ARHouse.Text.Length > 0 && ARIndex.Text.Length > 0)

{

Address curAddr = new Address();

curAddr.Country = ARCountry.Text;

curAddr.Locality = ARLocality.Text;

int ind;

if (int.TryParse(ARIndex.Text, out ind)) { curAddr.Index = ind; }

else isDamaged = true;

curAddr.House = ARHouse.Text;

if (ARDistrict.Text.Length > 0) curAddr.District = ARDistrict.Text;

if (ARStreet.Text.Length > 0) curAddr.Street = ARStreet.Text;

if (ARPhone.Text.Length > 0) curAddr.Telephone = ARPhone.Text;

if (ARApartment.Text.Length > 0) curAddr.Apartment = ARApartment.Text;

if (ARKorpus.Text.Length > 0) curAddr.Housing = ARKorpus.Text;

if (ARisForeign.IsChecked.Value) curAddr.IsForeign = true;

curAddr.IsRegistration = true;

curAddr.IsCurrentAddress = false;

if (isDamaged) return null;

else return curAddr;

}

return null;

}

private List<Address> GetAddressInfo()

{

bool isDamaged = false;

if (ACountry.Text.Length > 0 && ALocality.Text.Length > 0 && AHouse.Text.Length > 0 && AIndex.Text.Length > 0)

{

Address curAddr = new Address();

curAddr.Country = ACountry.Text;

curAddr.Locality = ALocality.Text;

int ind;

if (int.TryParse(AIndex.Text, out ind)) { curAddr.Index = ind; }

else isDamaged = true;

curAddr.House = AHouse.Text;

if (ADistrict.Text.Length > 0) curAddr.District = ADistrict.Text;

if (AStreet.Text.Length > 0) curAddr.Street = AStreet.Text;

if (APhone.Text.Length > 0) curAddr.Telephone = APhone.Text;

if (AApartment.Text.Length > 0) curAddr.Apartment = AApartment.Text;

if (AKorpus.Text.Length > 0) curAddr.Housing = AKorpus.Text;

if (AisForeignBox.IsChecked.Value) curAddr.IsForeign = true;

curAddr.IsCurrentAddress = true;

if (AisRegistrationBox.IsChecked.Value) { curAddr.IsRegistration = true; }

if (isDamaged) return null;

else

{

if (curAddr.IsRegistration)

{

return new List<Address>() { curAddr };

}

else

{

Address twoAddr = GetSecondAddressInfo();

if (twoAddr != null)

{

return new List<Address>() { curAddr, twoAddr };

}

else return null;

}

}

}

return null;

}

private Work GetWorkInfo()

{

if (WPlace.Text.Length > 0 && WPost.Text.Length > 0)

{

Work wk = new Work();

wk.PlaceName = WPlace.Text;

wk.Post = WPost.Text;

if (WExperience.Text.Length > 0) wk.Experience = WExperience.Text;

return wk;

}

else return null;

}

private Education GetEducationInfo()

{

if (EName.Text.Length > 0 && EType.SelectedItem != null && EYear.Text.Length > 0 && ENumber.Text.Length > 0)

{

bool isDamaged = false;

Education ed = new Education();

ed.Title = EName.Text;

ed.Type = (string)EType.SelectedItem;

try { ed.YearEnd = short.Parse(EYear.Text); }

catch { isDamaged = true; }

ed.DocNumber = ENumber.Text;

if (EBall.Text.Length > 0)

try { ed.Ball = decimal.Parse(EBall.Text.Replace(".", ",")); }

catch { isDamaged = true; }

if (ESpeciality.Text.Length > 0) ed.Speciality = ESpeciality.Text;

if (!isDamaged) return ed;

else return null;

}

else return null;

}

private MaritalStatus GetMaritalStatus()

{

MaritalStatus mr = new MaritalStatus();

if (FisInvalid.IsChecked.Value) mr.Disabled = true;

if (FisMarried.IsChecked.Value) mr.Married = true;

if (FisOpeka.IsChecked.Value) mr.Guardianship = true;

if (FisPosob.IsChecked.Value) mr.Allowance = true;

return mr;

}

private Passports GetPassportInfo()

{

if (PSerie.Text.Length > 0 && PNumber.Text.Length > 0 &&

PWho.Text.Length > 0 && PDate.SelectedDate != null)

{

Passports ps = new Passports();

ps.Date = PDate.SelectedDate.Value;

ps.Issued = PWho.Text;

ps.Series = PSerie.Text;

int numb;

if (Int32.TryParse(PNumber.Text, out numb)) ps.Number = numb;

else return null;

return ps;

}

return null;

}

private StudentPhoto GetStudentPhoto()

{

if (StudentImage.Source != null)

{

try

{

StudentPhoto ph = new StudentPhoto();

Stream img = BImage.StreamSource;

img.Position = 0;

MemoryStream mems = new MemoryStream();

img.CopyTo(mems);

ph.Photo = mems.ToArray();

return ph;

}

catch

{

return null;

}

}

return null;

}

private void GetStudentData()

{

if (!EditMode)

{

NewStudent();

}

else

{

EditStudent();

}

}

private void NewStudent()

{

Students stud = GetStudent();

StudentPhoto photo = GetStudentPhoto();

MaritalStatus marital = GetMaritalStatus();

Work work = GetWorkInfo();

Passports passport = GetPassportInfo();

Education education = GetEducationInfo();

List<Address> addr = GetAddressInfo();

List<Residence> residence = new List<Residence>();

if (stud != null && marital != null && education != null &&

passport != null && addr != null && DatabseIDs != null)

{

stud.Status = Enums.StudentStatus.Last();

stud.ID = DatabseIDs.StudentID;

marital.ID = stud.ID;

if (work != null) work.ID = stud.ID;

passport.ID = stud.ID;

education.ID = stud.ID;

try

{

if (!DatabaseAgent.Instance.Database.Students.Any(i => (i.Name == stud.Name && i.Surname == stud.Surname && i.Patronymic == stud.Patronymic && i.DateOfBirth == stud.DateOfBirth)))

DatabaseAgent.Instance.Database.Students.AddObject(stud);

else

{

return;

}

if (photo != null)

{

photo.ID = stud.ID;

DatabaseAgent.Instance.Database.StudentPhoto.AddObject(photo);

}

DatabaseAgent.Instance.Database.Education.AddObject(education);

if (work != null) DatabaseAgent.Instance.Database.Work.AddObject(work);

DatabaseAgent.Instance.Database.Passports.AddObject(passport);

DatabaseAgent.Instance.Database.MaritalStatus.AddObject(marital);

foreach (Address adr in addr)

{

adr.ID = DatabseIDs.AddressID;

DatabaseAgent.Instance.Database.Address.AddObject(adr);

Residence rs = new Residence();

rs.IDAddress = adr.ID;

rs.IDStudent = stud.ID;

rs.ID = DatabseIDs.ResidenceID;

DatabaseAgent.Instance.Database.Residence.AddObject(rs);

residence.Add(rs);

DatabseIDs.ResidenceID++;

DatabseIDs.AddressID++;

}

foreach (Citizen cz in Countries)

{

cz.IDStudent = stud.ID;

cz.ID = DatabseIDs.CitizenID;

DatabaseAgent.Instance.Database.Citizen.AddObject(cz);

DatabseIDs.CitizenID++;

}

foreach (Relationships rl in Relation)

{

rl.IDMartialStatus = marital.ID;

rl.Relatives.ID = DatabseIDs.RelativeID;

rl.ID = DatabseIDs.RelationshipID;

DatabaseAgent.Instance.Database.Relationships.AddObject(rl);

DatabseIDs.RelationshipID++;

DatabseIDs.RelativeID++;

}

DatabaseAgent.Instance.Database.SaveChanges();

MainWindow.Instance.FocusWindow();

this.Close();

// Success

}

catch // undo changes

{

try

{

DatabseIDs = new CurrentIDs();

DatabaseAgent.Instance.Database.Students.Detach(stud);

if (work != null) DatabaseAgent.Instance.Database.Work.Detach(work);

DatabaseAgent.Instance.Database.Passports.Detach(passport);

DatabaseAgent.Instance.Database.Education.Detach(education);

DatabaseAgent.Instance.Database.StudentPhoto.Detach(photo);

DatabaseAgent.Instance.Database.MaritalStatus.Detach(marital);

foreach (Relationships rl in Relation)

{

DatabaseAgent.Instance.Database.Relatives.Detach(rl.Relatives);

DatabaseAgent.Instance.Database.Relationships.Detach(rl);

}

foreach (Citizen cz in Countries)

DatabaseAgent.Instance.Database.Citizen.Detach(cz);

foreach (Residence rs in residence)

{

DatabaseAgent.Instance.Database.Residence.Detach(rs);

}

foreach (Address adr in addr)

{

DatabaseAgent.Instance.Database.Address.Detach(adr);

}

return;

}

catch

{

}

}

}

else

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

private void EditStudent()

{

Students stud = GetStudent();

StudentPhoto photo = GetStudentPhoto();

MaritalStatus marital = GetMaritalStatus();

Work work = GetWorkInfo();

Passports passport = GetPassportInfo();

Education education = GetEducationInfo();

List<Address> addr = GetAddressInfo();

if (stud != null && marital != null && education != null &&

passport != null && addr != null && DatabseIDs != null)

{

bool isEdited = false;

try //изменения личных данных студента

{

Students EditableStudent = DatabaseAgent.Instance.Database.Students.First(s => s.ID == CurrentStudent.ID);

if (stud.Name != CurrentStudent.Name) { isEdited = true; EditableStudent.Name = stud.Name; }

if (stud.Surname != CurrentStudent.Surname) { isEdited = true; EditableStudent.Surname = stud.Surname; }

if (stud.Patronymic != CurrentStudent.Patronymic) { isEdited = true; EditableStudent.Patronymic = stud.Patronymic; }

if (stud.Sex != CurrentStudent.Sex) { isEdited = true; EditableStudent.Sex = stud.Sex; }

if (stud.NumberOfRecordBook != CurrentStudent.NumberOfRecordBook) { isEdited = true; EditableStudent.NumberOfRecordBook = stud.NumberOfRecordBook; }

if (stud.NumberOfStudetnID != CurrentStudent.NumberOfStudetnID) { isEdited = true; EditableStudent.NumberOfStudetnID = stud.NumberOfStudetnID; }

if (stud.DateOfBirth != CurrentStudent.DateOfBirth) { isEdited = true; EditableStudent.DateOfBirth = stud.DateOfBirth; }

if (stud.ForeignLanguage != CurrentStudent.ForeignLanguage) { isEdited = true; EditableStudent.ForeignLanguage = stud.ForeignLanguage; }

}

catch

{

MessageManager.ShowError(Errors.ConnectError, Errors.ConnectErrorTitle);

return;

}

try // изменения паспортных данных студента

{

Passports EditablePassport = DatabaseAgent.Instance.Database.Passports.First(s => s.ID == CurrentStudent.ID);

if (passport.Issued != CurrentStudent.Passports.Issued) { isEdited = true; EditablePassport.Issued = passport.Issued; }

if (passport.Number != CurrentStudent.Passports.Number) { isEdited = true; EditablePassport.Number = passport.Number; }

if (passport.Date != CurrentStudent.Passports.Date) { isEdited = true; EditablePassport.Date = passport.Date; }

if (passport.Series != CurrentStudent.Passports.Series) { isEdited = true; EditablePassport.Series = passport.Series; }

}

catch

{

MessageManager.ShowError(Errors.ConnectError, Errors.ConnectErrorTitle);

return;

}

try // изменения образования студента

{

Education EditableEducation = DatabaseAgent.Instance.Database.Education.First(s => s.ID == CurrentStudent.ID);

if (education.Ball != CurrentStudent.Education.Ball) { isEdited = true; EditableEducation.Ball = education.Ball; }

if (education.DocNumber != CurrentStudent.Education.DocNumber) { isEdited = true; EditableEducation.DocNumber = education.DocNumber; }

if (education.Speciality != CurrentStudent.Education.Speciality) { isEdited = true; EditableEducation.Speciality = education.Speciality; }

if (education.Title != CurrentStudent.Education.Title) { isEdited = true; EditableEducation.Title = education.Title; }

if (education.Type != CurrentStudent.Education.Type) { isEdited = true; EditableEducation.Type = education.Type; }

if (education.YearEnd != CurrentStudent.Education.YearEnd) { isEdited = true; EditableEducation.YearEnd = education.YearEnd; }

}

catch

{

MessageManager.ShowError(Errors.ConnectError, Errors.ConnectErrorTitle);

return;

}

if (work != null)

{

if (CurrentStudent.Work == null)

{

work.ID = CurrentStudent.ID;

DatabaseAgent.Instance.Database.Work.AddObject(work);

isEdited = true;

}

else

{

try // изменения работы студента

{

Work EditableWork = DatabaseAgent.Instance.Database.Work.First(s => s.ID == CurrentStudent.ID);

if (work.Experience != CurrentStudent.Work.Experience) { isEdited = true; EditableWork.Experience = work.Experience; }

if (work.PlaceName != CurrentStudent.Work.PlaceName) { isEdited = true; EditableWork.PlaceName = work.PlaceName; }

if (work.Post != CurrentStudent.Work.Post) { isEdited = true; EditableWork.Post = work.Post; }

}

catch

{

MessageManager.ShowError(Errors.ConnectError, Errors.ConnectErrorTitle);

return;

}

}

}

else

{

if (CurrentStudent.Work != null)

DatabaseAgent.Instance.Database.Work.DeleteObject(DatabaseAgent.Instance.Database.Work

.First(s => s.ID == CurrentStudent.ID));

isEdited = true;

}

// изменение фотографии студента ==========================

if (photo != null)

{

if (CurrentStudent.StudentPhoto == null)

{

photo.ID = CurrentStudent.ID;

DatabaseAgent.Instance.Database.StudentPhoto.AddObject(photo);

isEdited = true;

}

else

{

StudentPhoto EditablePhoto = DatabaseAgent.Instance.Database.StudentPhoto.First(s => s.ID == CurrentStudent.ID);

if (CurrentStudent.StudentPhoto.Photo.Length != photo.Photo.Length) { isEdited = true; EditablePhoto.Photo = photo.Photo; }

}

}

else

{

if (CurrentStudent.StudentPhoto != null)

DatabaseAgent.Instance.Database.StudentPhoto.DeleteObject(DatabaseAgent.Instance.Database.StudentPhoto

.First(s => s.ID == CurrentStudent.ID));

isEdited = true;

}

try // изменения семейного положения

{

MaritalStatus EditableMarital = DatabaseAgent.Instance.Database.MaritalStatus.First(s => s.ID == CurrentStudent.ID);

if (marital.Allowance != CurrentStudent.MaritalStatus.Allowance) { isEdited = true; EditableMarital.Allowance = marital.Allowance; }

if (marital.Disabled != CurrentStudent.MaritalStatus.Disabled) { isEdited = true; EditableMarital.Disabled = marital.Disabled; }

if (marital.Guardianship != CurrentStudent.MaritalStatus.Guardianship) { isEdited = true; EditableMarital.Guardianship = marital.Guardianship; }

if (marital.Married != CurrentStudent.MaritalStatus.Married) { isEdited = true; EditableMarital.Married = marital.Married; }

}

catch

{

MessageManager.ShowError(Errors.ConnectError, Errors.ConnectErrorTitle);

return;

}

// изменение адресов студента

try

{

List<Residence> currRes = CurrentStudent.Residence.ToList();

Address currAddr = currRes[0].Address;

Address regAddr = null;

if (currRes.Count > 1) regAddr = currRes[1].Address;

Address newAddr = addr[0];

Address EditableAddress = DatabaseAgent.Instance.Database.Address.First(s => s.ID == currAddr.ID);

if (newAddr.Country != currAddr.Country) { isEdited = true; EditableAddress.Country = newAddr.Country; }

if (newAddr.Apartment != currAddr.Apartment) { isEdited = true; EditableAddress.Apartment = newAddr.Apartment; }

if (newAddr.District != currAddr.District) { isEdited = true; EditableAddress.District = newAddr.District; }

if (newAddr.House != currAddr.House) { isEdited = true; EditableAddress.House = newAddr.House; }

if (newAddr.Housing != currAddr.Housing) { isEdited = true; EditableAddress.Housing = newAddr.Housing; }

if (newAddr.Index != currAddr.Index) { isEdited = true; EditableAddress.Index = newAddr.Index; }

if (newAddr.IsCurrentAddress != currAddr.IsCurrentAddress) { isEdited = true; EditableAddress.IsCurrentAddress = newAddr.IsCurrentAddress; }

if (newAddr.IsForeign != currAddr.IsForeign) { isEdited = true; EditableAddress.IsForeign = newAddr.IsForeign; }

if (newAddr.IsRegistration != currAddr.IsRegistration) { isEdited = true; EditableAddress.IsRegistration = newAddr.IsRegistration; }

if (newAddr.Locality != currAddr.Locality) { isEdited = true; EditableAddress.Locality = newAddr.Locality; }

if (newAddr.Street != currAddr.Street) { isEdited = true; EditableAddress.Street = newAddr.Street; }

if (newAddr.Telephone != currAddr.Telephone) { isEdited = true; EditableAddress.Telephone = newAddr.Telephone; }

if (currRes.Count == 1 && addr.Count == 2)

{

Address newReg = addr[1];

newReg.ID = DatabseIDs.AddressID;

Residence res = new Residence();

res.IDAddress = newReg.ID;

res.IDStudent = CurrentStudent.ID;

res.ID = DatabseIDs.ResidenceID;

DatabaseAgent.Instance.Database.Address.AddObject(newReg);

DatabaseAgent.Instance.Database.Residence.AddObject(res);

isEdited = true;

}

if (currRes.Count == 2 & addr.Count == 1)

{

DatabaseAgent.Instance.Database.Residence

.DeleteObject(DatabaseAgent.Instance.Database.Residence.First(s => s.ID == currRes[1].ID));

isEdited = true;

}

if (currRes.Count == 2 && addr.Count == 2)

{

Address RegA = addr[1];

Address EditableAddress2 = DatabaseAgent.Instance.Database.Address.First(s => s.ID == regAddr.ID);

if (RegA.Country != regAddr.Country) { isEdited = true; EditableAddress2.Country = RegA.Country; }

if (RegA.Apartment != regAddr.Apartment) { isEdited = true; EditableAddress2.Apartment = RegA.Apartment; }

if (RegA.District != regAddr.District) { isEdited = true; EditableAddress2.District = RegA.District; }

if (RegA.House != regAddr.House) { isEdited = true; EditableAddress2.House = RegA.House; }

if (RegA.Housing != regAddr.Housing) { isEdited = true; EditableAddress2.Housing = RegA.Housing; }

if (RegA.Index != regAddr.Index) { isEdited = true; EditableAddress2.Index = RegA.Index; }

if (RegA.IsCurrentAddress != regAddr.IsCurrentAddress) { isEdited = true; EditableAddress2.IsCurrentAddress = RegA.IsCurrentAddress; }

if (RegA.IsForeign != regAddr.IsForeign) { isEdited = true; EditableAddress2.IsForeign = RegA.IsForeign; }

if (RegA.IsRegistration != regAddr.IsRegistration) { isEdited = true; EditableAddress2.IsRegistration = RegA.IsRegistration; }

if (RegA.Locality != regAddr.Locality) { isEdited = true; EditableAddress2.Locality = RegA.Locality; }

if (RegA.Street != regAddr.Street) { isEdited = true; EditableAddress2.Street = RegA.Street; }

if (RegA.Telephone != regAddr.Telephone) { isEdited = true; EditableAddress2.Telephone = RegA.Telephone; }

}

}

catch

{

}

//изменение родственников студента

try

{

if (DeletedRelationships.Count > 0)

{

foreach (Relationships rl in DeletedRelationships)

{

DatabaseAgent.Instance.Database.Relationships

.DeleteObject(DatabaseAgent.Instance.Database.Relationships.First(s => s.ID == rl.ID));

}

isEdited = true;

}

if (AddedRelationships.Count > 0)

{

foreach (Relationships rl in AddedRelationships)

{

rl.Relatives.ID = DatabseIDs.RelativeID;

rl.IDRelative = DatabseIDs.RelativeID;

DatabseIDs.RelativeID++;

rl.ID = DatabseIDs.RelationshipID;

DatabseIDs.RelationshipID++;

rl.IDMartialStatus = CurrentStudent.ID;

DatabaseAgent.Instance.Database.Relatives.AddObject(rl.Relatives);

DatabaseAgent.Instance.Database.Relationships.AddObject(rl);

}

isEdited = true;

}

}

catch

{

MessageManager.ShowError(Errors.ConnectError, Errors.ConnectErrorTitle);

}

//изменение гражданства студента

try

{

if (DeletedCitizen.Count > 0)

{

foreach (Citizen cz in DeletedCitizen)

DatabaseAgent.Instance.Database.Citizen

.DeleteObject(DatabaseAgent.Instance.Database.Citizen.First(s => s.ID == cz.ID));

isEdited = true;

}

if (AddedCitizen.Count > 0)

{

foreach (Citizen cz in AddedCitizen)

{

cz.ID = DatabseIDs.CitizenID;

cz.IDStudent = CurrentStudent.ID;

DatabseIDs.CitizenID++;

DatabaseAgent.Instance.Database.Citizen.AddObject(cz);

}

isEdited = true;

}

}

catch

{

MessageManager.ShowError(Errors.ConnectError, Errors.ConnectErrorTitle);

}

if (isEdited) DatabaseAgent.Instance.Database.SaveChanges();

MainWindow.Instance.FocusWindow();

this.Close();

}

else

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

return;

}

private Students GetStudent()

{

if (StudentName.Text.Length > 0 && StudentSurname.Text.Length > 0 && StudentPatron.Text.Length > 0 &&

DateBirth.SelectedDate != null && Sex.SelectedItem != null && Nationality.SelectedItem != null &&

NUmberBilet.Text.Length > 0 && NumberRecord.Text.Length > 0)

{

Students st = new Students();

st.Name = StudentName.Text;

st.Surname = StudentSurname.Text;

st.Patronymic = StudentPatron.Text;

st.DateOfBirth = DateBirth.SelectedDate.Value;

st.Sex = Sex.SelectedIndex == 0 ? false : true;

st.IDNationality = ((Nationality)Nationality.SelectedItem).ID;

int rn;

if (int.TryParse(NumberRecord.Text, out rn)) st.NumberOfRecordBook = rn;

else return null;

int bn;

if (int.TryParse(NUmberBilet.Text, out bn)) st.NumberOfStudetnID = bn;

else return null;

if (Languages.Text.Length > 0) st.ForeignLanguage = Languages.Text;

return st;

}

return null;

}

private void CreateRelationship(object sender, RoutedEventArgs e)

{

if (FFIO.Text.Length > 0 && FType.SelectedItem != null)

{

Relatives rt = new Relatives();

rt.Name = FFIO.Text;

if (FDate.SelectedDate != null && FDate.SelectedDate < DateTime.Now)

rt.DateOfBirth = FDate.SelectedDate;

if (FPlace.Text.Length > 0) rt.WorkPlace = FPlace.Text;

if (FPhone.Text.Length > 0) rt.Telephone = FPhone.Text;

Relationships rl = new Relationships();

rl.Relatives = rt;

rl.Relationship = (string)FType.SelectedItem;

if (EditMode) AddedRelationships.Add(rl);

Relation.Add(rl);

RelativeList.ItemsSource = null;

RelativeList.ItemsSource = Relation;

}

}

private void BrowsePhoto(object sender, RoutedEventArgs e)

{

OpenFileDialog od = new OpenFileDialog();

od.CheckFileExists = true;

od.CheckPathExists = true;

od.Title = "Выберите фотографию студента";

if (od.ShowDialog().Value)

{

BImage = new BitmapImage();

BImage.BeginInit();

BImage.StreamSource = File.Open(od.FileName, FileMode.Open, FileAccess.Read, FileShare.Read);

BImage.EndInit();

StudentImage.Source = BImage;

}

}

private void ClearPhoto(object sender, RoutedEventArgs e)

{

StudentImage.Source = null;

BImage = null;

}

private void AddCountry(object sender, RoutedEventArgs e)

{

if (StudentCountry.SelectedItem != null)

{

Citizen cz = new Citizen();

cz.IDCitizenship = ((Citizenship)StudentCountry.SelectedItem).ID;

if (EditMode) AddedCitizen.Add(cz);

Countries.Add(cz);

StudentCountry.SelectedItem = null;

CountryList.ItemsSource = null;

CountryList.ItemsSource = Countries;

}

}

private void DeleteCountry(object sender, KeyEventArgs e)

{

if (CountryList.SelectedItem != null)

{

if (e.Key == Key.Delete)

{

if (EditMode && ((Citizen)CountryList.SelectedItem).ID != 0)

DeletedCitizen.Add((Citizen)CountryList.SelectedItem);

Countries.Remove((Citizen)CountryList.SelectedItem);

CountryList.ItemsSource = null;

CountryList.ItemsSource = Countries;

}

}

}

private void InitializeState(object sender, RoutedEventArgs e)

{

try

{

DatabseIDs = new CurrentIDs();

Nationality.ItemsSource = DatabaseAgent.Instance.Database.Nationality.ToList();

StudentCountry.ItemsSource = DatabaseAgent.Instance.Database.Citizenship.ToList();

if (CurrentStudent != null)

HistoryGrid.ItemsSource = DatabaseAgent.Instance.Database.Documents

.Join(DatabaseAgent.Instance.Database.List

.Where(s => s.IDStudent == CurrentStudent.ID),

s => s.ID, t => t.IDDocument, (s, t) => s);

}

catch

{

MessageManager.ShowError(Errors.ConnectError, Errors.ConnectErrorTitle);

}

}

private void RelativeDelete\_Click(object sender, RoutedEventArgs e)

{

if (RelativeList.SelectedItem != null)

{

if (EditMode && ((Relationships)RelativeList.SelectedItem).ID != 0)

DeletedRelationships.Add((Relationships)RelativeList.SelectedItem);

Relation.Remove((Relationships)RelativeList.SelectedItem);

RelativeList.ItemsSource = null;

RelativeList.ItemsSource = Relation;

}

}

}

**Исходный код класса AddStructureWindow**

public partial class AddStructureWindow : Window

{

bool EditMode;

CurrentIDs IDs;

public AddStructureWindow()

{

this.InitializeComponent();

MainWindow.Instance.UnfocusWindow();

DatabaseAgent.Instance.Database.SavingChanges += new EventHandler(Database\_SavingChanges);

}

void Database\_SavingChanges(object sender, EventArgs e)

{

Initialize(this, null); }

Faculties EditFaculty { get; set; }

Departments EditDepartment { get; set; }

Specialties EditSpecialty { get; set; }

Specializations EditSpecialization { get; set; }

Groups EditGroup { get; set; }

public AddStructureWindow(Faculties fac)

{

this.InitializeComponent();

MainWindow.Instance.UnfocusWindow();

AddButton.IconText = "сохранить";

EditMode = true;

EKafedra.IsEnabled = false;

EGroup.IsEnabled = false;

ESpecialization.IsEnabled = false;

ESpecialty.IsEnabled = false;

EditFaculty = fac;

FDekan.Text = EditFaculty.Dean;

FName.Text = EditFaculty.Title;

FPhone.Text = EditFaculty.Telephone;

FCode.Text = EditFaculty.Chiper;

FEmail.Text = EditFaculty.Email;

EFaculty.IsExpanded = true;

}

public AddStructureWindow(Departments dep)

{

this.InitializeComponent();

MainWindow.Instance.UnfocusWindow();

AddButton.IconText = "сохранить";

EditMode = true;

EditDepartment = dep;

KCode.Text = EditDepartment.Chiper;

KZaved.Text = EditDepartment.Head;

KPhone.Text = EditDepartment.Telephone;

KName.Text = EditDepartment.Title;

try

{

KFaculty.ItemsSource = DatabaseAgent.Instance.Database.Faculties;

}

catch { }

EKafedra.IsExpanded = true;

}

public AddStructureWindow(Specialties spec)

{

EditMode = true;

this.InitializeComponent();

MainWindow.Instance.UnfocusWindow();

AddButton.IconText = "сохранить";

EditSpecialty = spec;

SName.Text = spec.Title;

SCode.Text = spec.Cipher;

if (spec.Years != null)

SYears.Text = spec.Years.ToString();

try

{

SFaculty.ItemsSource = DatabaseAgent.Instance.Database.Faculties;

SFormsList.ItemsSource = DatabaseAgent.Instance.Database.ExistingForms

.Where(s => s.IDSpecialty == spec.ID);

}

catch { }

ESpecialty.IsExpanded = true;

}

public AddStructureWindow(Specializations spec)

{

EditMode = true;

this.InitializeComponent();

MainWindow.Instance.UnfocusWindow();

AddButton.IconText = "сохранить";

EditSpecialization = spec;

SPName.Text = spec.Title;

SPCode.Text = spec.Chiper;

SPQualificat.Text = spec.Qualification;

try

{

SPFaculty.ItemsSource = DatabaseAgent.Instance.Database.Faculties;

}

catch { }

ESpecialization.IsExpanded = true;

}

public AddStructureWindow(Groups gr)

{

EditMode = true;

this.InitializeComponent();

MainWindow.Instance.UnfocusWindow();

AddButton.IconText = "сохранить";

EditGroup = gr;

GName.Text = gr.Title;

GYear.Text = gr.YearOfFoundation.ToString();

try

{

GFaculty.ItemsSource = DatabaseAgent.Instance.Database.Faculties;

}

catch { }

EGroup.IsExpanded = true;

}

private void ChangeExpand(object sender, RoutedEventArgs e)

{

if ((Expander)sender == EFaculty)

{

ESpecialization.IsExpanded = false;

ESpecialty.IsExpanded = false;

EGroup.IsExpanded = false;

EKafedra.IsExpanded = false;

}

else if ((Expander)sender == EKafedra)

{

ESpecialization.IsExpanded = false;

ESpecialty.IsExpanded = false;

EGroup.IsExpanded = false;

EFaculty.IsExpanded = false;

}

else if ((Expander)sender == ESpecialty)

{

ESpecialization.IsExpanded = false;

EKafedra.IsExpanded = false;

EGroup.IsExpanded = false;

EFaculty.IsExpanded = false;

}

else if ((Expander)sender == ESpecialization)

{

ESpecialty.IsExpanded = false;

ESpecialty.IsExpanded = false;

EGroup.IsExpanded = false;

EFaculty.IsExpanded = false;

}

else if ((Expander)sender == EGroup)

{

ESpecialization.IsExpanded = false;

ESpecialty.IsExpanded = false;

EKafedra.IsExpanded = false;

EFaculty.IsExpanded = false;

}

}

private Faculties GetFaculty()

{

if (FName.Text.Length > 0 && FDekan.Text.Length > 0)

{

Faculties fac = new Faculties();

fac.Title = FName.Text;

fac.Dean = FDekan.Text;

if (FCode.Text.Length > 0) fac.Chiper = FCode.Text;

if (FPhone.Text.Length > 0) fac.Telephone = FPhone.Text;

if (FEmail.Text.Length > 0) fac.Email = FEmail.Text;

return fac;

}

return null;

}

private Departments GetDepartment()

{

if (KName.Text.Length > 0 && KFaculty.SelectedItem != null && KZaved.Text.Length > 0)

{

Departments dep = new Departments();

dep.Title = KName.Text;

dep.Head = KZaved.Text;

dep.IDFaculty = ((Faculties)KFaculty.SelectedItem).ID;

if (KCode.Text.Length > 0) dep.Chiper = KCode.Text;

if (KPhone.Text.Length > 0) dep.Telephone = KPhone.Text;

return dep;

}

return null;

}

private Specialties GetSpecialty()

{

if (SName.Text.Length > 0 && SFaculty.SelectedItem != null

&& SKafedra.SelectedItem != null && SCode.Text.Length > 0 && SFormsList.Items.Count > 0)

{

Specialties sp = new Specialties();

sp.Title = SName.Text;

sp.Cipher = SCode.Text;

sp.IDDepartment = ((Departments)SKafedra.SelectedItem).ID;

if (SYears.Text.Length > 0)

{

short i;

if (short.TryParse(SYears.Text, out i)) sp.Years = i;

}

return sp;

}

return null;

}

private List<ExistingForms> GetForms()

{

if (SFormsList.Items.Count > 0)

{

List<ExistingForms> lst = new List<ExistingForms>();

foreach (string str in SFormsList.Items)

{

ExistingForms exs = new ExistingForms();

exs.Form = str;

lst.Add(exs);

}

return lst;

}

return null;

}

private Specializations GetSpecialization()

{

if (SPName.Text.Length > 0 && SPQualificat.Text.Length > 0 &&

SPSpec.SelectedItem != null)

{

Specializations spec = new Specializations();

spec.Title = SPName.Text;

spec.Qualification = SPQualificat.Text;

if (SPCode.Text.Length > 0) spec.Chiper = SPCode.Text;

spec.IDSpecialty = ((Specialties)SPSpec.SelectedItem).ID;

return spec;

}

return null;

}

//редактирование

void EditStructure()

{

if (EFaculty.IsExpanded)

{

Faculties fac = GetFaculty();

if (fac != null)

{

try

{

Faculties fc = DatabaseAgent.Instance.Database.Faculties.First(s => s.ID == EditFaculty.ID);

fc.Chiper = fac.Chiper;

fc.Dean = fac.Dean;

fc.Email = fac.Email;

fc.Telephone = fac.Telephone;

fc.Title = fac.Title;

}

catch

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else if (EKafedra.IsExpanded)

{

Departments dep = GetDepartment();

if (dep != null)

{

try

{

Departments dp = DatabaseAgent.Instance.Database.Departments.First(s => s.ID == EditDepartment.ID);

dp.Chiper = dep.Chiper;

dp.Head = dep.Head;

dp.IDFaculty = dep.IDFaculty;

dp.Telephone = dep.Telephone;

dp.Title = dep.Title;

}

catch

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else if (ESpecialty.IsExpanded)

{

Specialties spec = GetSpecialty();

List<ExistingForms> lst = GetForms();

if (spec != null && lst != null)

{

try

{

Specialties sp = DatabaseAgent.Instance.Database.Specialties.First(s => s.ID == spec.ID);

foreach (ExistingForms ex in lst)

{

if (EditSpecialty.ExistingForms.First(s => s.Form == ex.Form) == null)

{

ex.ID = IDs.ExistingFormID;

IDs.ExistingFormID++;

ex.IDSpecialty = sp.ID;

DatabaseAgent.Instance.Database.ExistingForms.AddObject(ex);

}

}

foreach (ExistingForms ex in sp.ExistingForms)

{

if (lst.First(s => s.Form == ex.Form) == null)

{

DatabaseAgent.Instance.Database.ExistingForms

.DeleteObject(DatabaseAgent.Instance.Database.ExistingForms.First(s => s.ID == ex.ID));

}

}

sp.Cipher = spec.Cipher;

sp.IDDepartment = spec.IDDepartment;

sp.Title = spec.Title;

sp.Years = spec.Years;

}

catch

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else if (ESpecialization.IsExpanded)

{

Specializations spec = GetSpecialization();

if (spec != null)

{

try

{

Specializations sp = DatabaseAgent.Instance.Database.Specializations

.First(s => s.ID == EditSpecialization.ID);

sp.Chiper = spec.Chiper;

sp.IDSpecialty = spec.IDSpecialty;

sp.Qualification = spec.Qualification;

sp.Title = spec.Title;

}

catch

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else if (EGroup.IsExpanded)

{

Groups group = GetGroup();

if (group != null)

{

try

{

Groups gr = DatabaseAgent.Instance.Database.Groups.First(s => s.ID == EditGroup.ID);

gr.IDForm = group.IDForm;

gr.IDSpecialization = group.IDSpecialization;

gr.Title = group.Title;

gr.YearOfFoundation = group.YearOfFoundation;

}

catch

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

}

private void AddStructureToDatabase()

{

if (!EditMode)

{

NewStructure();

}

else

{

EditStructure();

}

try

{

DatabaseAgent.Instance.Database.SaveChanges();

if (!EditMode)

{

MessageWindow msg = new MessageWindow(Messages.StructureAdded, Messages.StructureAddedTitle);

msg.Owner = this;

msg.ShowDialog();

this.Close();

}

else

{

MessageWindow msg = new MessageWindow("Данные были успешно изменены.", "Изменение данных");

msg.Owner = this;

msg.ShowDialog();

this.Close();

}

}

catch

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

private Groups GetGroup()

{

if (GName.Text.Length > 0 && GYear.Text.Length > 0 && GForma.SelectedItem != null && GSpecializat.SelectedItem != null)

{

Groups gr = new Groups();

gr.Title = GName.Text;

int year;

if (Int32.TryParse(GYear.Text, out year))

gr.YearOfFoundation = year;

else return null;

gr.IDSpecialization = ((Specializations)GSpecializat.SelectedItem).ID;

gr.IDForm = ((ExistingForms)GForma.SelectedItem).ID;

return gr;

}

return null;

}

private void NewStructure()

{

if (EFaculty.IsExpanded)

{

Faculties fac = GetFaculty();

if (fac != null)

{

try

{

if (!DatabaseAgent.Instance.Database.Faculties.Any(s => s.Title == fac.Title))

{

fac.ID = IDs.FacultyID;

try

{

DatabaseAgent.Instance.Database.Faculties.AddObject(fac);

}

catch

{

DatabaseAgent.Instance.Database.Faculties.Detach(fac);

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else MessageManager.ShowError(Errors.DuplicatError, Errors.DuplicateErrorTitle);

}

catch

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else if (EKafedra.IsExpanded)

{

Departments dep = GetDepartment();

if (dep != null)

{

try

{

if (!DatabaseAgent.Instance.Database.Departments.Any(s => s.Title == dep.Title))

{

dep.ID = IDs.DepartmentID;

try

{

DatabaseAgent.Instance.Database.Departments.AddObject(dep);

}

catch

{

DatabaseAgent.Instance.Database.Departments.Detach(dep);

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else MessageManager.ShowError(Errors.DuplicatError, Errors.DuplicateErrorTitle);

}

catch

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else if (ESpecialty.IsExpanded)

{

Specialties spec = GetSpecialty();

List<ExistingForms> lst = GetForms();

if (spec != null && lst != null)

{

try

{

if (!DatabaseAgent.Instance.Database.Specialties.Any(s => s.Title == spec.Title))

{

spec.ID = IDs.SpecialtyID;

int fid = IDs.ExistingFormID;

foreach (ExistingForms fm in lst)

{

fm.IDSpecialty = spec.ID;

fm.ID = fid;

fid++;

}

try

{

DatabaseAgent.Instance.Database.Specialties.AddObject(spec);

foreach (ExistingForms fm in lst)

DatabaseAgent.Instance.Database.ExistingForms.AddObject(fm);

}

catch

{

foreach (ExistingForms fm in lst)

DatabaseAgent.Instance.Database.ExistingForms.Detach(fm);

DatabaseAgent.Instance.Database.Specialties.Detach(spec);

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else MessageManager.ShowError(Errors.DuplicatError, Errors.DuplicateErrorTitle);

}

catch

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else if (ESpecialization.IsExpanded)

{

Specializations spec = GetSpecialization();

if (spec != null)

{

try

{

if (!DatabaseAgent.Instance.Database.Specializations.Any(s => s.Title == spec.Title))

{

spec.ID = IDs.SpecializatioinID;

try

{

DatabaseAgent.Instance.Database.Specializations.AddObject(spec);

}

catch

{

DatabaseAgent.Instance.Database.Specializations.Detach(spec);

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else MessageManager.ShowError(Errors.DuplicatError, Errors.DuplicateErrorTitle);

}

catch

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else if (EGroup.IsExpanded)

{

Groups group = GetGroup();

if (group != null)

{

try

{

if (!DatabaseAgent.Instance.Database.Groups.Any(s => s.Title == group.Title))

{

group.ID = IDs.GroupID;

try

{

DatabaseAgent.Instance.Database.Groups.AddObject(group);

}

catch

{

DatabaseAgent.Instance.Database.Groups.Detach(group);

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else MessageManager.ShowError(Errors.DuplicatError, Errors.DuplicateErrorTitle);

}

catch

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

else

{

MessageManager.ShowError(Errors.AddingError, Errors.AddingErrorTitle);

}

}

}

private void Initialize(object sender, RoutedEventArgs e)

{

BackgroundWorker work = new BackgroundWorker();

work.DoWork += new DoWorkEventHandler(SetFirstState);

work.RunWorkerAsync();

}

void SetFirstState(object sender, DoWorkEventArgs e)

{

this.Dispatcher.BeginInvoke(new AsyncDelegate(GetData), null);

}

void GetData()

{

try

{

IDs = new CurrentIDs();

KFaculty.ItemsSource = null;

SFaculty.ItemsSource = null;

SPFaculty.ItemsSource = null;

GFaculty.ItemsSource = null;

KFaculty.ItemsSource = DatabaseAgent.Instance.Database.Faculties;

SFaculty.ItemsSource = DatabaseAgent.Instance.Database.Faculties;

SPFaculty.ItemsSource = DatabaseAgent.Instance.Database.Faculties;

GFaculty.ItemsSource = DatabaseAgent.Instance.Database.Faculties;

}

catch { }

}

private void FacultyChange(object sender, SelectionChangedEventArgs e)

{

try

{

SKafedra.ItemsSource = DatabaseAgent.Instance.Database.Departments.

Where(s => s.IDFaculty == ((Faculties)SFaculty.SelectedItem).ID);

}

catch { }

}

private void AddForm(object sender, RoutedEventArgs e)

{

if (SForma.SelectedItem != null)

if (!SFormsList.Items.Contains((string)SForma.SelectedItem))

{

SFormsList.Items.Add((string)SForma.SelectedItem);

}

}

private void DeleteForm(object sender, KeyEventArgs e)

{

if (e.Key == Key.Delete)

if (SFormsList.SelectedItem != null)

{

SFormsList.Items.Remove(SFormsList.SelectedItem);

}

}

private void SPFacultyChange(object sender, SelectionChangedEventArgs e)

{

try

{

SPKafedra.ItemsSource = DatabaseAgent.Instance.Database.Departments.

Where(s => s.IDFaculty == ((Faculties)SPFaculty.SelectedItem).ID);

}

catch { }

}

private void SPDepartmentChange(object sender, SelectionChangedEventArgs e)

{

try

{

SPSpec.ItemsSource = DatabaseAgent.Instance.Database.Specialties.

Where(s => s.IDDepartment == ((Departments)SPKafedra.SelectedItem).ID);

}

catch { }

}

private void GroupFaultyChange(object sender, SelectionChangedEventArgs e)

{

try

{

GKafedra.ItemsSource = DatabaseAgent.Instance.Database.Departments.

Where(s => s.IDFaculty == ((Faculties)GFaculty.SelectedItem).ID);

}

catch { }

}

private void GroupDepartmentChange(object sender, SelectionChangedEventArgs e)

{

try

{

GSpec.ItemsSource = DatabaseAgent.Instance.Database.Specialties.

Where(s => s.IDDepartment == ((Departments)GKafedra.SelectedItem).ID);

}

catch { }

}

private void GroupSpecChange(object sender, SelectionChangedEventArgs e)

{

try

{

GSpecializat.ItemsSource = DatabaseAgent.Instance.Database.Specializations.

Where(s => s.IDSpecialty == ((Specialties)GSpec.SelectedItem).ID);

GForma.ItemsSource = DatabaseAgent.Instance.Database.ExistingForms.

Where(s => s.IDSpecialty == ((Specialties)GSpec.SelectedItem).ID);

}

catch { }

}

private void CloseWindow()

{

MainWindow.Instance.FocusWindow();

this.Close();

}

}