МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение иформационных технологий

Специализация Программирование интернет-приложение

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Применение технологии составления отчётов базы данных MS SQL Server для задачи “Dean-helper”

Выполнил студент Шенец Полина Сергеевна

(Ф.И.О.)

Руководитель проекта доц. Блинова Е.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В .

(учен. степень, звание, должность, Ф.И.О., подпись)

Консультант: доц. Блинова Е.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер: доц. Блинова Е.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Минск 2018

Оглавление

[1. Введение 3](#_Toc533244382)

[2. Постановка задачи 4](#_Toc533244383)

[3. Разработка модели базы данных 5](#_Toc533244384)

[4. Разработка необходимых объектов 15](#_Toc533244385)

[4.1. Таблицы 15](#_Toc533244386)

[4.2. Пользователи 15](#_Toc533244387)

[4.3. Хранимые процедуры 15](#_Toc533244388)

[5. Описание процедур импорта и экспорта 16](#_Toc533244389)

[5.1. Описание процедуры экспорта 16](#_Toc533244390)

[5.2. Описание процедуры импорта 16](#_Toc533244391)

[6. Тестирование производительности 17](#_Toc533244392)

[7. Описание технологии 18](#_Toc533244393)

[8. Руководство пользователя 20](#_Toc533244394)

[9. Заключение 22](#_Toc533244395)

[10. Список используемых источников 23](#_Toc533244396)

[Приложение A 24](#_Toc533244397)

[Приложение Б 25](#_Toc533244398)

[Приложение В 32](#_Toc533244399)

[Приложение Г 38](#_Toc533244400)

[Приложение Д 106](#_Toc533244401)

[Приложение Е 109](#_Toc533244402)

# Введение

В данной записке приведено описание реляционной базы данных, разработанной в соответствии с заданием на курсовое проектирование по теме «Применение технологии составления отчётов базы данных MS SQL Server для задачи “Dean-helper”» по дисциплине «Базы данных».

База данных — представленная в объективной форме совокупность самостоятельных материалов (статей, расчётов, нормативных актов и иных подобных материалов), систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины.

Реляционная база данных — база данных, основанная на реляционной модели данных.

Для разработки и управления базой данных курсового проекта использовалась объектно-реляционная система управления базами данных «MS SQL Server».

Система управления базами данных (СУБД) – совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Основные функции СУБД:

* определение структуры создаваемой базы данных, ее инициализация и проведение начальной загрузки;
* предоставление пользователям возможности манипулирования данными (выборка необходимых данных, выполнение вычислений, разработка интерфейса ввода/вывода, визуализация);
* обеспечение логической и физической независимости данных;
* защита логической целостности базы данных;
* защита физической целостности;
* управление полномочиями пользователей на доступ к базе данных;
* синхронизация работы нескольких пользователей;
* управление ресурсами среды хранения;
* поддержка деятельности системного персонала. [1]

Примеры СУБД: Oracle, MS SQL Server, Microsoft Access, MySql, PostgreSQL, MongoDB и так далее.

# Постановка задачи

В ходе подготовки к разработке курсового проекта была проанализирована система хранения данных деканата о студентах, ведомостях, распоряжениях, аттестациях и другой информации. Было принято решение создать реляционную базу данных для оптимизирования работы с документацией. Соответственно в данном курсовом проекте требовалось реализовать следующие задачи:

* Добавление, изменение и удаление информации о студентах, старостах, оценках, преподавателях и пропусках;
* Поиск информации по заданным критериям;
* Запись информации в xml-файл;
* Считывание информации из xml-файла;
* Составление выписок из зачёток и отчётов о успеваемости;
* Составление списков групп;
* Оповещение об академических задолженностях.

Для реализации ПС были выбраны следующие технологии:

* Microsoft SQL Server – система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft;
* SQL Server Reporting Services – это спектр готовых к использованию средств и служб для создания и развертывания мобильных отчетов и отчетов с разбиением на страницы в локальной среде, а также управления ими. [2]

# 3. Разработка модели базы данных

Для базы данных Deanery было разработано 34 таблицы. В приложении А представлена диаграмма базы, на которой изображены все первичные и внешние ключи. База данных была разработана в СУБД MS SQL Server 2014.

Описание всех таблиц приведено ниже.

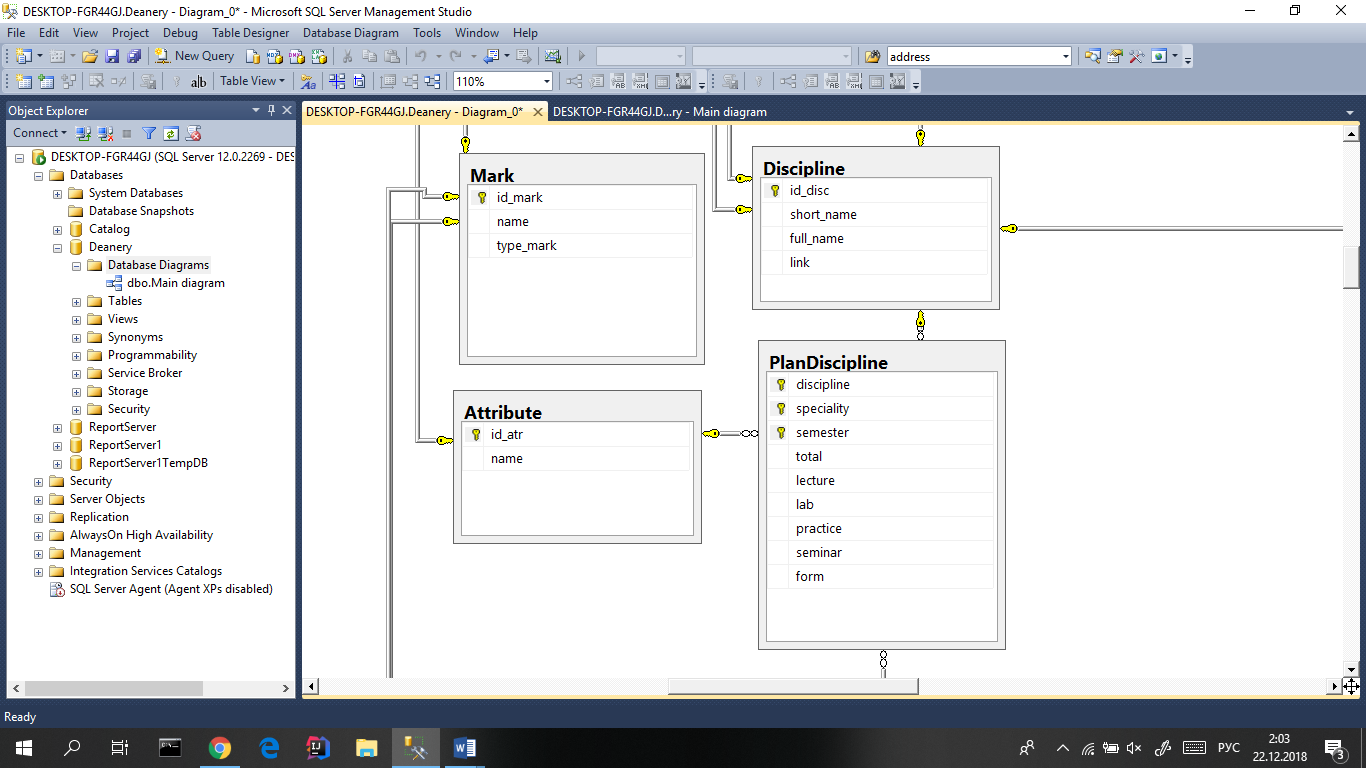


Рисунок 3.1 – Таблицы Mark, Attribute, Discipline, PlanDiscipline

Таблица **Mark** является **с**правочником оценок:

* Id\_Mark (первичный ключ) – код оценки, int;
* Name – наименование оценки (0-10, зачтено, н/а и другие), nvarchar(15) ;
* Type\_Mark – тип оценки (положительный-1, отрицательный-0), bit.

Таблица **Attribute** представляет собой справочник с признаком в зачётке:

* Id\_Atr (первичный ключ) – код признака, int;
* Name – наименование (зачёт, экзамен и другие), nvarchar(15).

Таблица **Discipline** представляет собой перечень учебных дисциплин:

* Id\_Disc (первичный ключ) – код дисциплины, int;
* Short\_Name – сокращённое название, nvarchar(20);
* Full\_Name – полное название, nvarchar(50);
* Link – ссылка на ресурс, nvarchar(500).

Таблица **PlanDiscipline** содержит план дисциплин:

* Discipline (первичный ключ, внешний ключ) – дисциплина, int;
* Speciality (первичный ключ, внешний ключ) – специальность, nvarchar(15);
* Semester (первичный ключ) – семестр, int;
* Total – всего часов, int;
* Lecture – количество лекционных часов, int;
* Lab – количество лабораторных часов, int;
* Practice – количество практических часов, int;
* Seminar – количество семинарских часов, int;
* Form (внешний ключ) – форма аттестации по окончании, int.

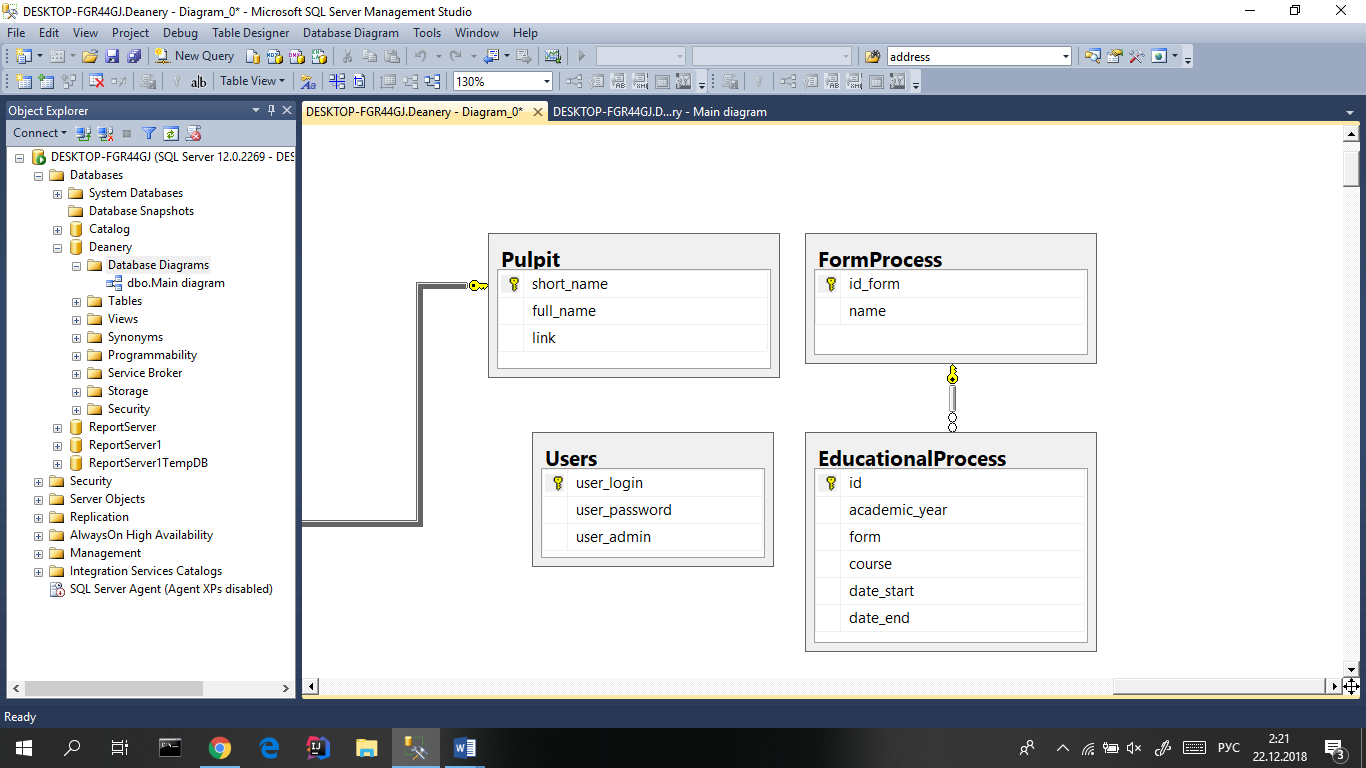


Рисунок 3.2 – Таблицы Pulpit, Users, FormProcess, EducationalProcess

Таблица **Pulpit** представляет собой перечень кафедр:

* Short\_Name (первичный ключ) – сокращённое название, nvarchar(10);
* Full\_Name – полное название, nvarchar(80);
* Link – ссылка на ресурс, nvarchar(500).

Таблица **Users** содержит информацию о пользователях:

* User\_Login (первичный ключ) – логин, nvarchar(16);
* User\_Password – пароль, nvarchar(20);
* User\_Admin – метка, что пользователь является админом, bit.

Таблица **FormProcess** содержит описания форм учебного процесса:

* Id\_Form (первичный ключ) – идентификатор формы, nvarchar(2);
* Name– название (осенний семестр, зимняя экзаменационная сессия, зимние каникулы и другие), nvarchar(30).

Таблица **EducationalProcess** содержит описание графика учебного процесса:

* Id (первичный ключ) – идентификатор, int;
* Academic\_Year – номер учебного года, char(9);
* Form (внешний ключ) – форма учебного процесса, nchar(2);
* Course – курс, int;
* Date\_Start – дата начала, date;
* Date\_End – дата конца, date.

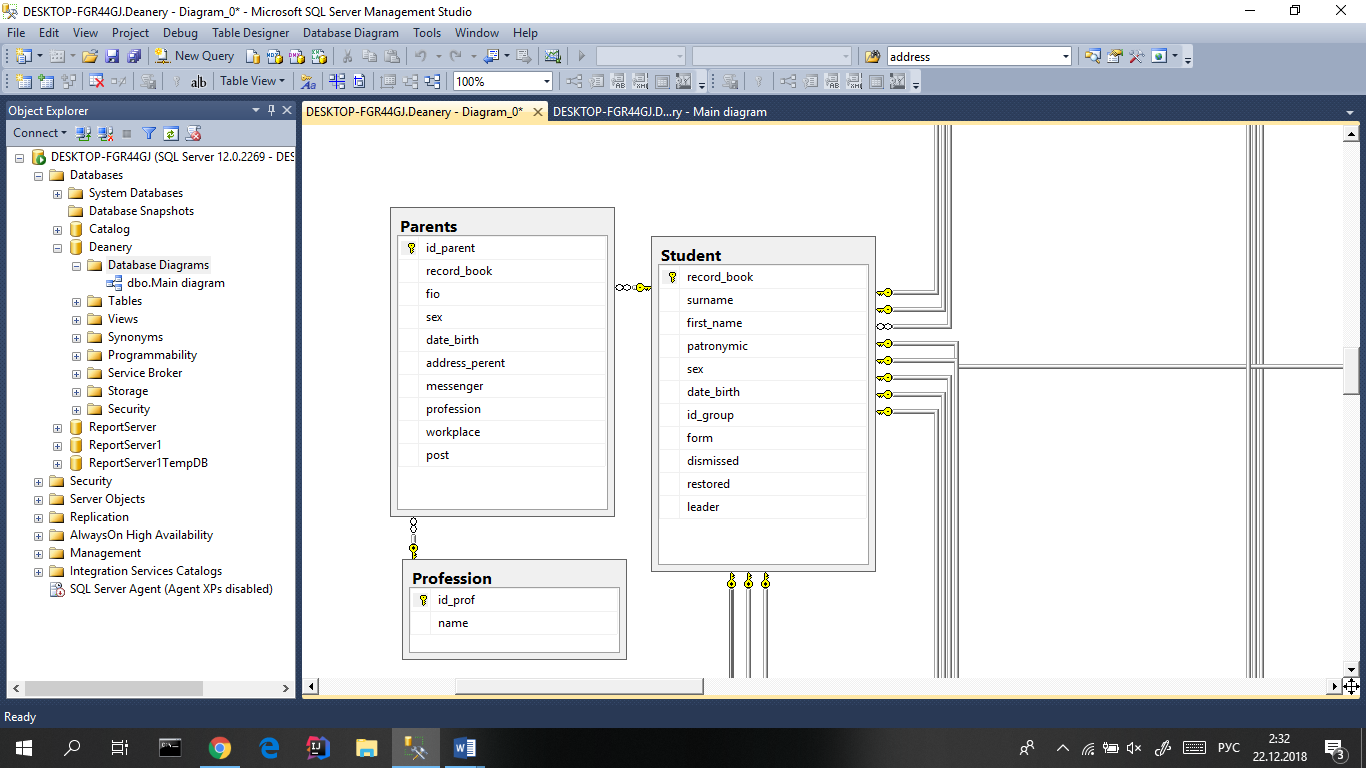


Рисунок 3.3 – Таблицы Pulpit, Users, FormProcess, EducationalProcess

Таблица **Student** хранит список всех студентов:

* Record\_Book (первичный ключ) – номер зачётной книжки, int;
* Surname – фамилия, nvarchar(15);
* First\_Name – имя, nvarchar(20);
* Patronymic – отчество, nvarchar(15);
* Sex – пол, nchar(1);
* Date\_Birth – дата рождения, date;
* Id\_Group (внешний ключ) – код группы, nvarchar(15);
* Form– форма обучения, nchar(1);
* Dismissed– был ли отчислен (поле, нарушающее нормализацию), bit;
* Restored– был ли восстановлен (поле, нарушающее нормализацию), bit;
* Leader– является ли старостой, bit.

Таблица **Parents** хранит список родителей студентов:

* Id\_Parent (первичный ключ) – id родителя, int;
* Record\_Book (внешний ключ) – номер зачётной книжки студента, int;
* FIO – фамилия, nvarchar(50);
* Sex – пол, nchar(1);
* Date\_Birth – дата рождения, date;
* Address\_Perent – адрес, nvarchar(150);
* Messenger– телефон(ы), nvarchar(50);
* Profession (внешний ключ) – номер профессии, int;
* Workplace – место работы, nvarchar(150);
* Post – должность, nvarchar(30).

Таблица **Profession** представляет собой справочник профессий:

* Id\_ Prof (первичный ключ) – код профессии, int;
* Name – наименование профессии, nvarchar (30).

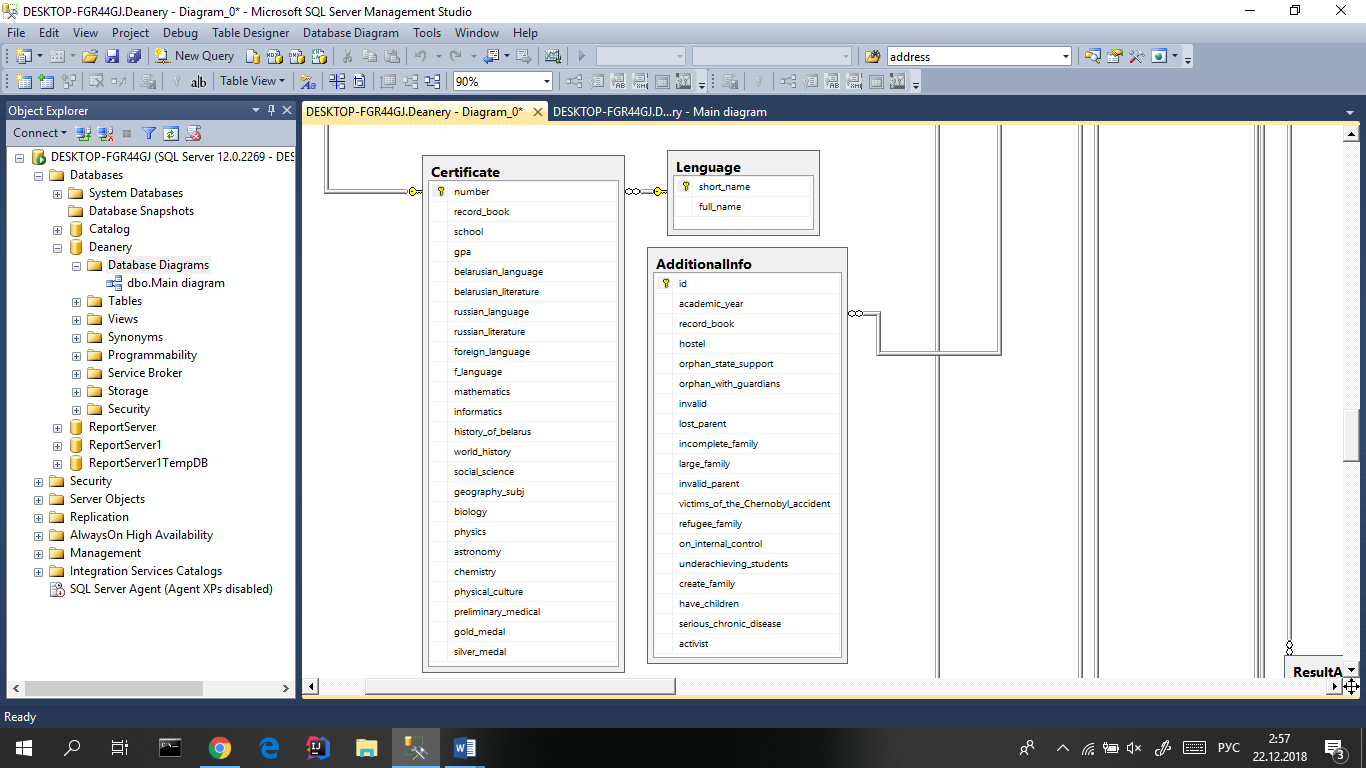


Рисунок 3.4 – Таблицы Lenguage, Certificate, AdditionalInfo

Таблица **Lenguage** представляет собой перечень языков:

* Short\_Name (первичный ключ) – сокращённое название, nvarchar(4);
* Full\_Name – полное название, nvarchar(20).

Таблица **Certificate** хранит в себе данные о аттестатах студентов:

* Number (первичный ключ) – регистрационный номер аттестата, int;
* Record\_Book (внешний ключ) – номер зачётной книжки, int;
* School– учереждение образования, nvarchar(50);
* GPA– средний балл, float;
* Belarusian\_Language – оценка по предмету белорусский язык, int;
* Belarusian\_Literature – по белорусской литературе, int;
* Russian\_Language – по русскому языку, int;
* Russian\_Literature – по русской литературе, int;
* Foreign\_Language – по второму иностранному языку, int;
* F\_Language (внешний ключ) – справочник языков, nvarchar(4);
* Mathematics – по математике, int;
* Informatics – по информатике, int;
* History\_Of\_Belarus – по истории Беларуси, int;
* World\_History – по всемирной истории, int;
* Social\_Science – по обществоведению , int;
* Geography\_Subj – по географии, int;
* Biology – по биологии, int;
* Physics – по физике, int;
* Astronomy – по астрономии, int;
* Chemistry – по химии, int;
* Physical\_Culture – по физической культуре, int;
* Preliminary\_Medical – по допризывной и медицинской подготовке, int;
* Gold\_Medal – наличие золотой медали, bit;
* Silver\_Medal – наличие серебряной медали, bit.

Таблица **AdditionalInfo** содержитдополнительную информацию о студентах:

* Id (первичный ключ) – идентификатор, int;
* Academic\_Year – номер учебного года, char(9);
* Record\_Book (внешний ключ) – номер зачётной книжки студента, int;
* Hostel – наличие общежития, bit;
* Orphan\_State\_Support– является ли сиротой на гос обеспечении, bit;
* Orphan\_With\_Guardians– сирота с опекунами, bit;
* Invalid – инвалид, bit;
* Lost\_Parent – потерял последнего из родителей во время учёбы, bit;
* Incomplete\_Family – неполная семья, bit;
* Large\_family – многодетная семья, bit;
* Invalid\_Parent – родитель инвалид, bit;
* Victims\_Of\_The\_Chernobyl\_Accident – пострадавший от аварии на Чернобыльской АЭС, bit;
* Refugee\_Family – семья беженцев, bit;
* On\_Internal\_Control – состоящие на внутреннем контроле, bit;
* Underachieving\_Students– неуспевающий студент, bit;
* Create\_Family – студент, создавший семью, bit;
* Have\_Children – имеет детей, bit;
* Serious\_Chronic\_Disease – тяжёлое хроническое заболевание, bit;
* Activist – активист молодёжных общественных организаций, bit.

Таблица **District** представляет собой список областей:

* Id\_ Distr (первичный ключ) – код области, int;
* Name – наименование области, nvarchar (12).

Таблица **Region** представляет собой перечень районов:

* Id\_ reg (первичный ключ) – код района, int;
* Name – наименование района, nvarchar (20);
* District (внешний ключ) – код области, int.

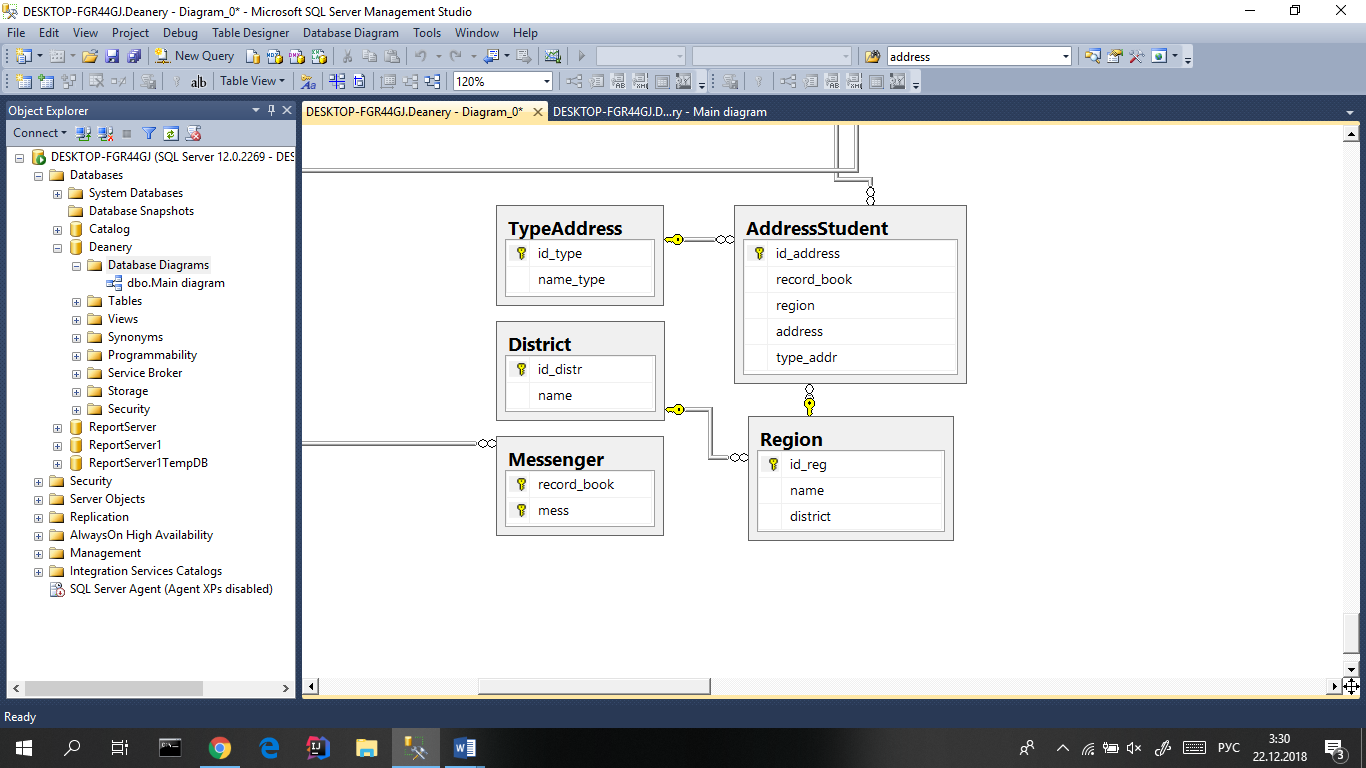


Рисунок 3.5 – Таблицы District, Region, TypeAddress, AddressStudent, Messenger

Таблица **TypeAddress** представляет собой справочник для типов адресов:

* Id\_Type (первичный ключ) – код типа, int;
* Name\_Type– наименование типа, nvarchar (12).

Таблица **AddressStudent** хранит cписок адресов:

* Id\_Address (первичный ключ) – id, int;
* Record\_Book (внешний ключ) – номер зачётной книжки, int;
* Region (внешний ключ) – район, int;
* Address – адрес, nvarchar(15);
* Type\_Addr (первичный ключ) – тип адреса, int.

Таблица **Messenger** представляет собой перечень мессенджеров:

* Record\_Book (первичный и внешний ключ) – номер зачётной книжки, int;
* Mess (первичный ключ) – наименование типа, nvarchar (100).

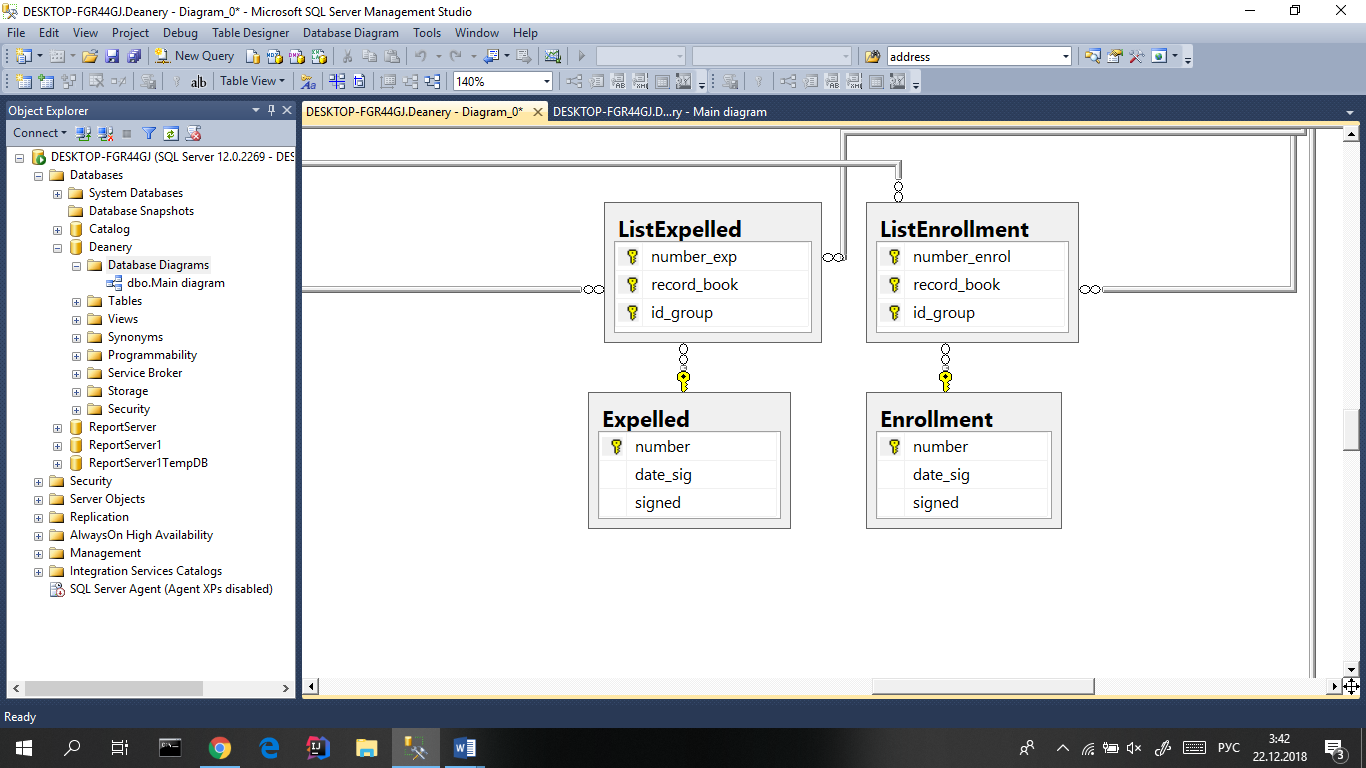


Рисунок 3.6 – Таблицы Enrollment, Expelled, ListEnrollment, ListExpelled

Таблица **Enrollment** хранит в себе распоряжения о зачислениях:

* Number (первичный ключ) – номер распоряжения, varchar(15);
* Date\_Sig – дата подписи, date;
* Signed – кто подписал, nvarchar(15).

Таблица **Expelled** предназначена для хранения распоряжений об отчислениях:

* Number (первичный ключ) – номер распоряжения, varchar(15);
* Date\_Sig – дата подписи, date;
* Signed – кто подписал, nvarchar(15).

Таблица **ListEnrollment** хранит список зачисленных студентов:

* Number\_Enrol (первичный и внешний ключ) – номер распоряжения, varchar(15);
* Record\_Book (первичный и внешний ключ) – номер зачётной книжки, int;
* Id\_Group (первичный ключ, внешний ключ) – код группы, nvarchar(15).

Таблица **ListExpelled** хранит список отчисленных студентов:

* Number\_ Exp (первичный и внешний ключ) – номер распоряжения, varchar(15);
* Record\_Book (первичный и внешний ключ) – номер зачётной книжки, int;
* Id\_Group (первичный и внешний ключ) – код группы, nvarchar(15).

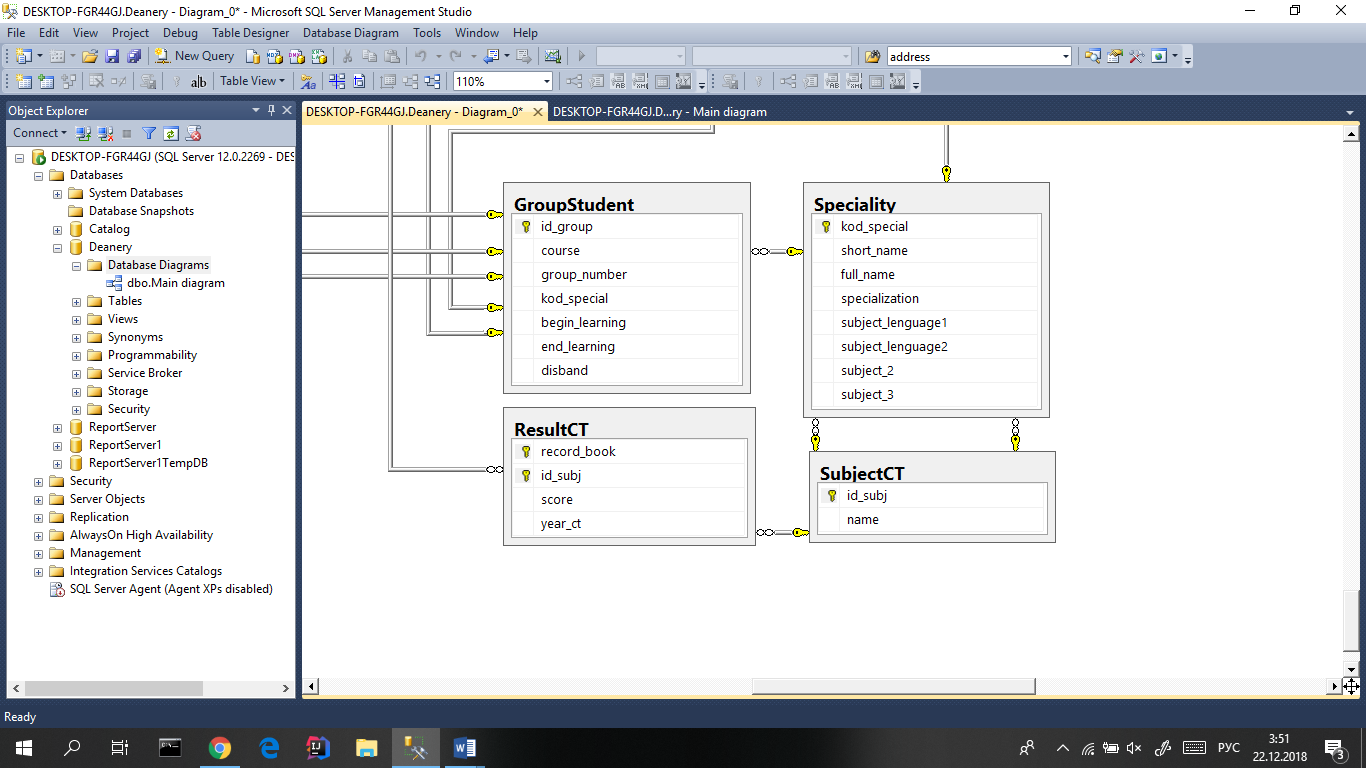


Рисунок 3.7 – Таблицы SubjectCT, Speciality, GroupStudent, ResultCT

Таблица **SubjectCT** представляет собой перечень предметов ЦТ:

* id\_Subj (первичный ключ) – код предмета, int;
* Name – наименование предмета, nvarchar (12).

Таблица **Speciality** хранит перечень специальностей:

* Kod\_Special (первичный ключ) – код специальности, nvarchar(13);
* Short\_Name – сокращённое название, nvarchar(10);
* Full\_Name – полное название, nvarchar(90);
* Specialization – специализация, nvarchar(35);
* Subject\_Lenguage1 (внешний ключ) – предмет ЦТ язык, int;
* Subject\_Lenguage2 (внешний ключ) – предмет ЦТ язык, int;
* Subject\_2 (внешний ключ) – предмет ЦТ (приоритет), int;
* Subject\_3 (внешний ключ) – предмет ЦТ, int.

Таблица **GroupStudent** хранит список групп:

* Id\_Group (первичный ключ) – код группы, nvarchar(18);
* Course – курс, int;
* Group\_Number – номер группы, int;
* Kod\_Special (внешний ключ) – код специальности, nvarchar(13);
* Begin\_Learning – начало обучения, date;
* End\_Learning – конец обучения, date;
* Disband – расформирована ли группа, bit.

Таблица **ResultCT** хранит результаты ЦТ каждого студента:

* Record\_Book (первичный ключ) – номер зачётной книжки, int;
* Id\_Subj (первичный ключ) – код предмета, int;
* Score (внешний ключ) – балл, int;
* Year\_Ct (внешний ключ) – год, int.

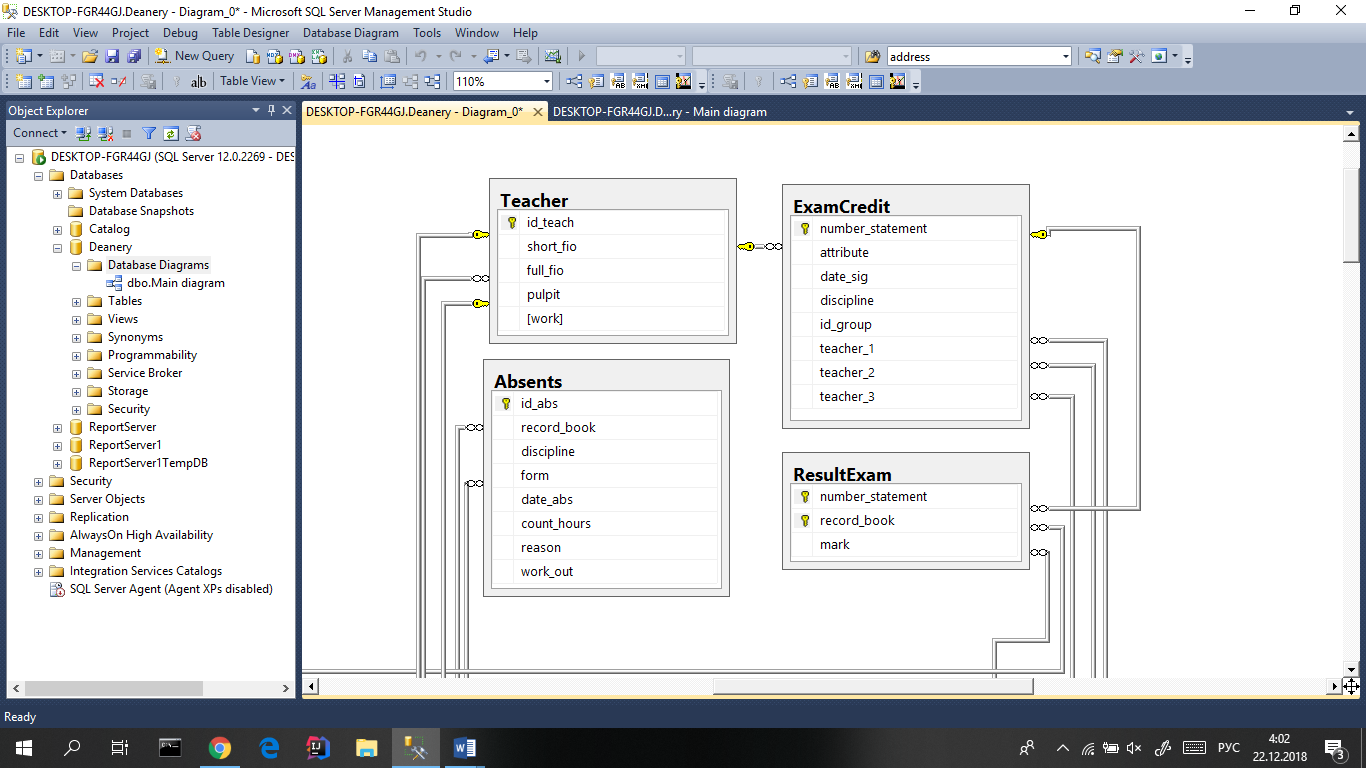


Рисунок 3.8 – Таблицы Teacher, ExamCredit, ResultExam, Absents

Таблица **Teacher** представляет собой перечень всех преподавателей:

* Id\_ Teach (первичный ключ) – код преподавателя, int;
* Short\_Fio – сокращённое имя, nvarchar(20);
* Full\_Fio – полное имя, nvarchar(50);
* Pulpit (внешний ключ) – кафедра, nvarchar(10);
* Work – работает ли ещё (0-нет, 1-да), bit.

Таблица **ExamCredit** хранит ведомости экзаменов и зачётов:

* Number\_Statement (первичный ключ) – номер ведомости, varchar(10);
* Attribute (внешний ключ) – признак, int;
* Date\_Sig – дата проведения, date;
* Discipline (внешний ключ) – код дисциплины, int;
* Id\_Group (внешний ключ) – код группы, nvarchar(13);
* Teacher\_1 (внешний ключ) – код преподавателя, int;
* Teacher\_2 (внешний ключ) – код второго преподавателя, int;
* Teacher\_3 (внешний ключ) – код третьего преподавателя, int.

Таблица **ResultExam** хранит Результаты экзаменов и зачётов:

* Number\_Statement (первичный и внешний ключ) – номер распоряжения, varchar(10);
* Record\_Book (первичный и внешний ключ) – номер зачётной книжки, int;
* Mark (внешний ключ) – код группы, int.

Таблица **Absents** содержит пропуски занятий:

* Id\_Abs (первичный ключ) – id, bigint;
* Record\_Book (внешний ключ) – номер зачётной книжки, int;
* Discipline (внешний ключ) – код дисциплины, int;
* Form – форма, nchar(2);
* Date\_Abs – дата, date;
* Count\_Hours – количество часов, int;
* Reason – причина пропуска, int;
* Work\_Out – отработано ли занятие, int.

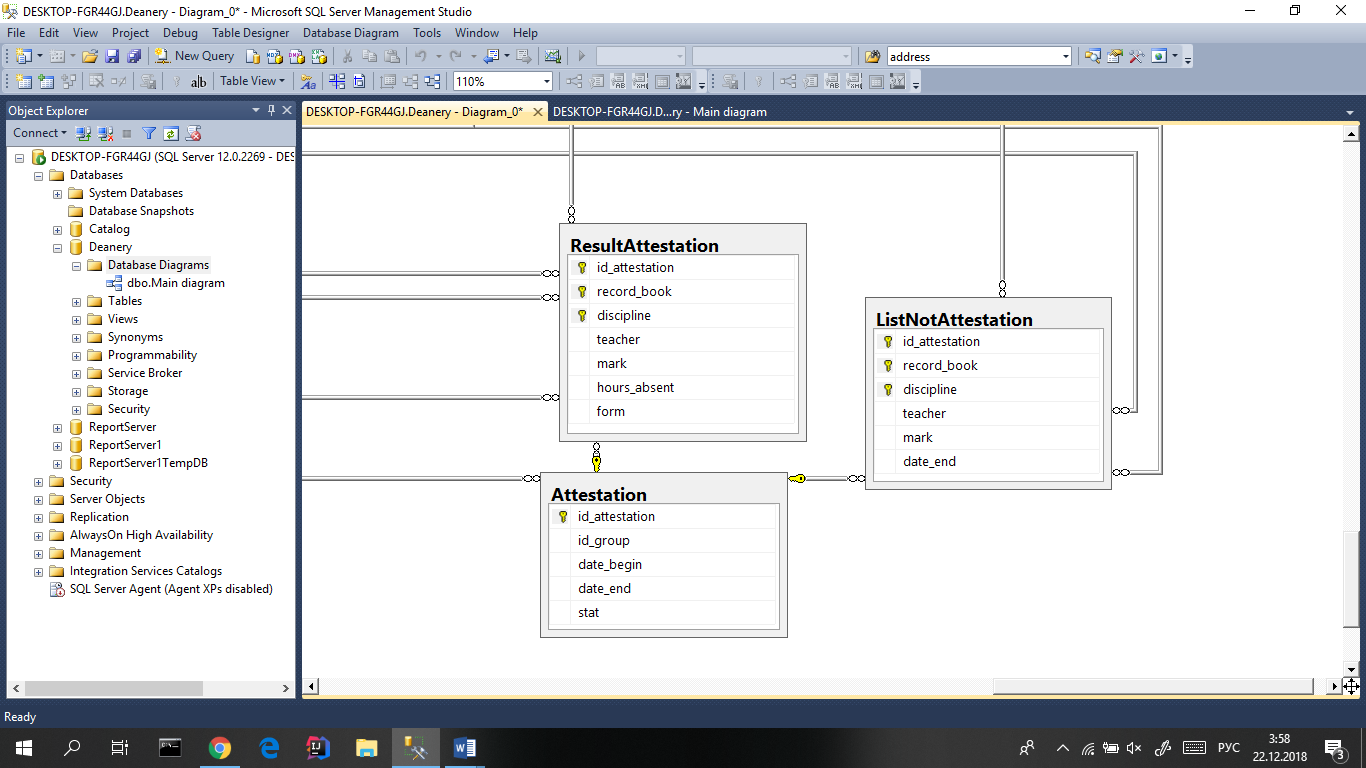


Рисунок 3.9 – Таблицы **Attestation**, **ResultAttestation**, **ListNotAttestation**

Таблица **Attestation** описывает листы аттестации:

* Id\_Attestation (первичный ключ) – номер аттестации, int;
* Id\_Group (внешний ключ) – код группы, nvarchar(18);
* Date\_Begin – начало аттестации, date;
* Date\_End – конец аттестации, date;
* Stat – статус аттестации, bit.

Таблица **ResultAttestation** содержит результаты аттестации:

* Id\_Attestation (первичный ключ, внешний ключ) – номер аттестации, int;
* Record\_Book (первичный ключ, внешний ключ) – номер зачётной книжки, Int;
* Discipline (первичный ключ, внешний ключ) – код дисциплины, int;
* Teacher (внешний ключ) – код преподавателя, int;
* Mark (внешний ключ) – код оценки, int;
* Hours\_Absent – часы пропуска, int;
* Form – номер аттестации, nchar(2).

Таблица **ListNotAttestation** содержитинформацию об отработки неаттестаций:

* Id\_Attestation (первичный ключ, внешний ключ) – номер аттестации, int;
* Record\_Book (первичный ключ) – номер зачётной книжки студента, int;
* Discipline (первичный ключ, внешний ключ) – код дисциплины, int;
* Teacher (внешний ключ) – код преподавателя, int;
* Mark (внешний ключ) – код оценки, int;
* Date\_End– дата сдачи листа, date;

Скрипт создания таблиц приведён в приложении Б.

# 4. Разработка необходимых объектов

При разработке базы данных курсового проекта понадобилось создать следующие объекты:

* Таблицы
* Пользователи
* Процедуры

## 4.1. Таблицы

Таблицы являются основой любой базы данных, именно в них хранится вся информация. При проектировании базы данных было создано 34 таблицы, которые подробно описаны ранее в разделе 3, а SQL-скрипты для их создания находятся в Приложении Б.

## 4.2. Пользователи

Пользователь базы данных – это учётная запись, которая имеет доступ к базе данных и пользуется услугами информационной системы для получения необходимой информации. В базе данных данного курсового проекта имеется 4 пользователь.

* Student – студент, привилегии которого ограничены ролью Student\_Role, позволяющей выполнять подключение к базе данных, а также вызов всех разрешённых этой роли процедур;
* Leader – староста, привилегии которого ограничены ролью Leader \_Role;
* Admin – администратор базы данных, привилегии которого ограничены ролью Admin \_Role, владеющий всеми процедурами.

Код создания пользователей и ролей приведен в приложении В.

## 4.3. Хранимые процедуры

Хранимая процедура – объект базы данных, представляющий собой набор SQL–инструкций, который компилируется один раз и хранится на сервере. В данном курсовом проекте они используются для предоставления доступа пользователю к данным, а также для экспорта и импорта в XML. Код хранимых процедур находится в приложении Г. Код хранимых процедур импорта и экспорта находится в приложении Д.

# 5. Описание процедур импорта и экспорта

Процедуры импорта и экспорта реализованы для таблицы Users.

## 5.1. Описание процедуры экспорта

В данной курсовой работе для того, чтобы провести экспорт данных из таблиц Users необходимо выполнить SELECT запрос и использовать FOR XML PATH (‘User’), ROOT (‘User’) для генерации XML–документа, более удобного для восприятия человеком.

## 5.2. Описание процедуры импорта

Для того, чтобы провести процедуру импорта из XML, в данном курсовом проекте были использованы ключевые функции и процедуры:

* openrowset, где указывается путь к файлу и в результате xml документ помещается в специальную переменную типа xml;
* sp\_xml\_preparedocument – стандартная процедура – парсер, в результате которой мы получаем документ, готовый к обработке;
* openxml, куда мы передаем хендл XML–документа, полученный после выполнения процедуры, описанной выше, XPATH, используемый для идентификации узлов документа и тип считывания данных (1 – считываем аттрибуты, 2 – считываем значения элементов, 8 – считываем аттрибуты и значения элементов).[2]

Так же были разработаны функции для импорта и экспорта в Excel с помощью Microsoft.ACE.OLEDB.12.0.

Код процедур импорта и экспорта данных представлен в приложении В. XML–документ, полученный в результате выполнения соответствующей процедуры представлен в приложении Е.

# 6. Тестирование производительности

Так как в данном проекте не было разработано приложения, то оптимизация производительности не тестировалась. Однако стоит заметить, что при дальнейшей работе с этой базой данных и разработке приложения, следует создать индексы для тех полей и таблиц по которым будет осуществляться какой-либо поиск.

# 7. Описание технологии

Службы SQL Server Reporting Services — это решение, развертываемое клиентами в их локальной среде для создания, публикации отчетов и управления ими, а также для их доставки нужным пользователям разнообразными способами, например, посредством просмотра в веб-браузере, на мобильном устройстве или в папке входящих сообщений электронной почты. [3]

SSRS реализован как web-служба и ее администрирование, а также управление отчетами производится через веб интерфейс. SSRS предоставляет возможность интегрировать разработанные отчеты в сторонние приложения, т.е. имеется некий API функционал. Для того чтобы сервер отчетов имел возможность хранить опубликованные отчеты, модели отчетов и иерархию папок требуется база данных SQL Server. SSRS предоставляют возможность подписки на публикуемые отчеты, т.е. например, по расписанию пользователь будет получать по почте или в общую папку свежие отчеты.

Все отчеты, созданные с помощью служб Reporting Services можно экспортировать в разные форматы (с сохранением визуального оформления), например:

* Excel;
* Word;
* PDF;
* CSV;
* XML;
* TIFF;
* MHTML (Web Archive).

Службами Reporting Services поддерживаются следующие типы источников данных для отчетов:

* Microsoft SQL Server;
* Microsoft SQL Azure;
* Microsoft SQL Server Analysis Services (поддержка многомерных баз данных);
* OLE DB;
* ODBC – соответственно поддерживаются файлы dBase, Excel, Access (MDB) и другие;
* Параллельные хранилища данных Microsoft;
* XML;
* Oracle;
* Teradata;
* Hyperion Essbase;
* Список Microsoft SharePoint;
* Sap NetWeaver BI;
* Report Server Model.

SQL Server Reporting Services (SSRS) – это очень мощное, а главное комплексное решение создания системы отчетности. Основные возможности SSRS:

* Создание, как простых табличных отчетов, так и сложных интерактивных;
* Экспорт отчетов в разные популярные форматы (Excel, Word, PDF и др.);
* Поддержка всевозможных источников данных для отчетов (SQL Server, SQL Azure, OLE DB, ODBC, Oracle и др.);
* Мощный конструктор Business Intelligence Development Studio;
* Подписка на отчеты;
* Кэширование отчетов;
* Интеграция с SharePoint;
* API интерфейс для интеграции с другими сторонними приложениями. [4]

# 8. Руководство пользователя

Для базы данных Deanery было разработано 5 отчётов SSRS.

На рисунке 8.1 представлен отчёт – “Список группы”. Входным параметром для этого отчёта является номер зачётки студента. В результате, студент получит список своей группы.

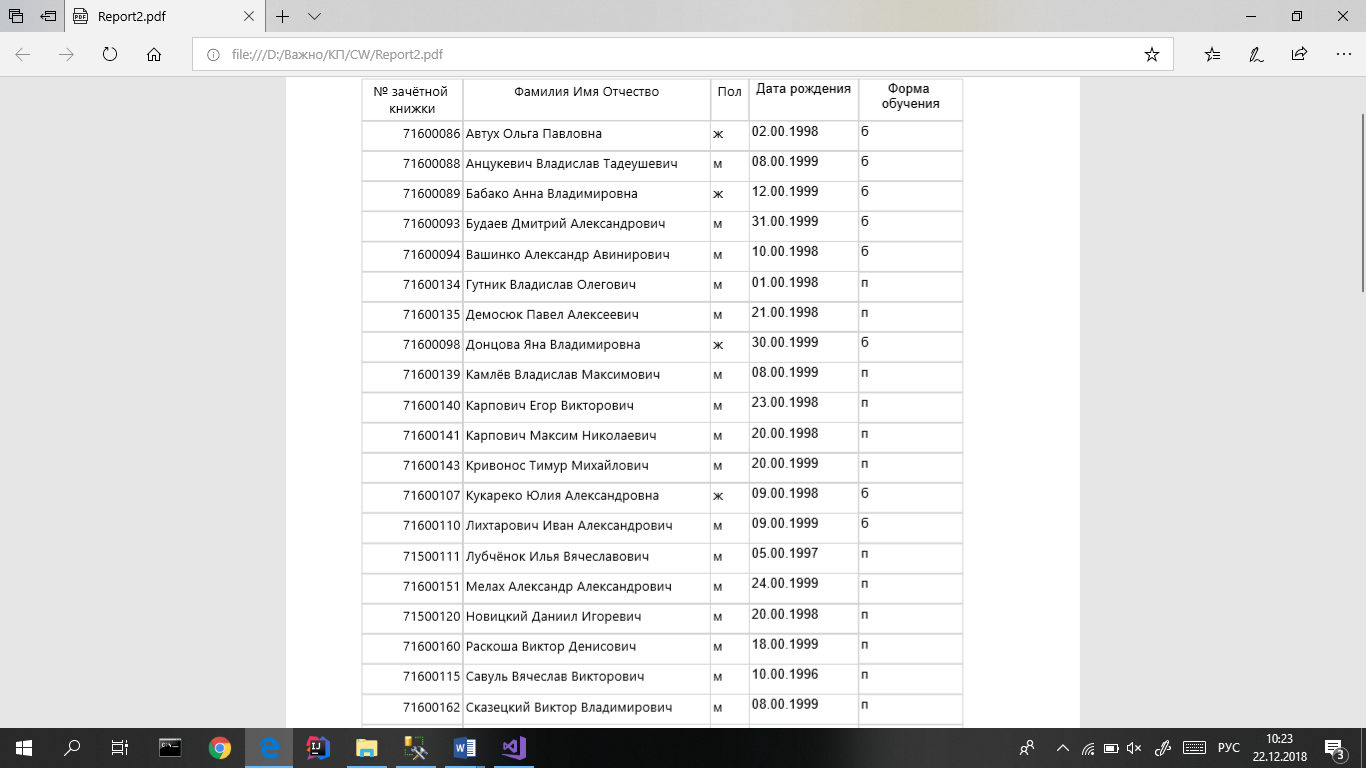


Рисунок 8.1 – Список студентов группы

На рисунке 8.2 представлен отчёт – “Аттестация”. Входным параметром для этого отчёта является номер аттестации. После выполнения отчёта, можно получит результат этой аттестации, в которой будет видна оценка по предмету и количество часов пропуска.

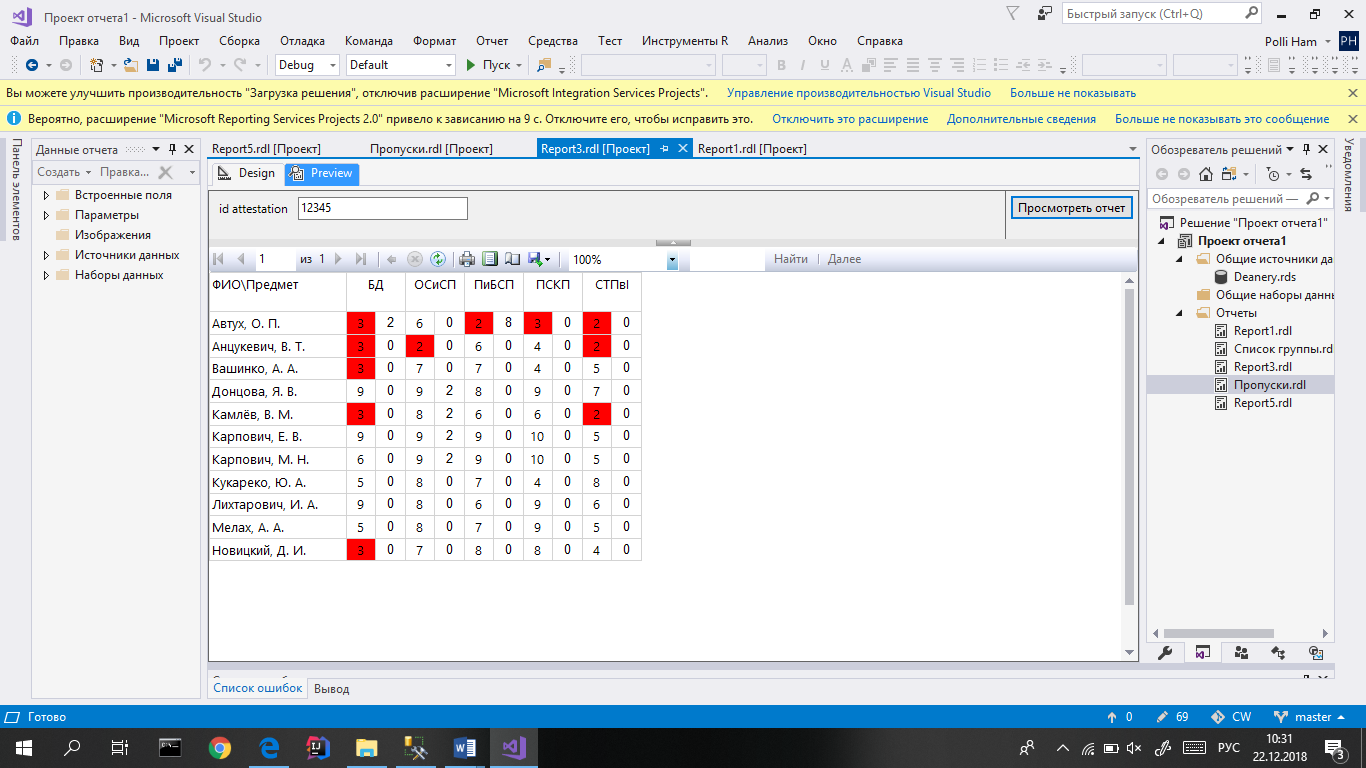


Рисунок 8.2 – Аттестация

На рисунке 8.3 представлен отчёт – “Пропуски”. Входным параметром для этого отчёта является номер курса, номер группы, дата начала подсчёта пропусков и дата конца. В результате, получаем количество часов пропусков по уважительной причине и по неуважительной. Отчёт будет особенно полезен для старост при заполнении сводных ведомостей пропусков.

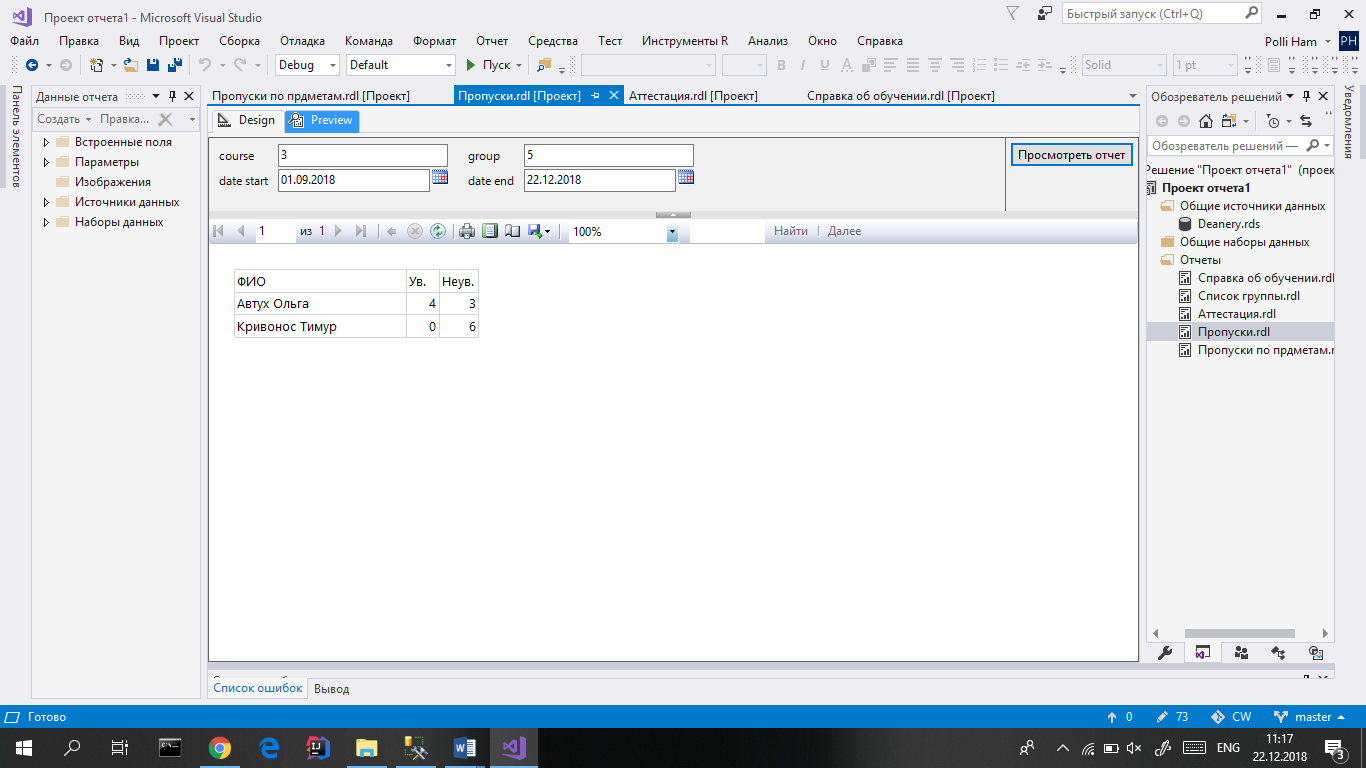


Рисунок 8.3 – Пропуски

Более подробную информацию о пропусках можно посмотреть в отчёте– “Пропуски по предметам” (рисунок 8.4). Входным параметром для этого отчёта является id группы, дата начала подсчёта пропусков и дата конца. В результате, получаем количество часов пропусков по определённым предметам. Отчёт будет особенно полезен для старост при заполнении сводных ведомостей пропусков.

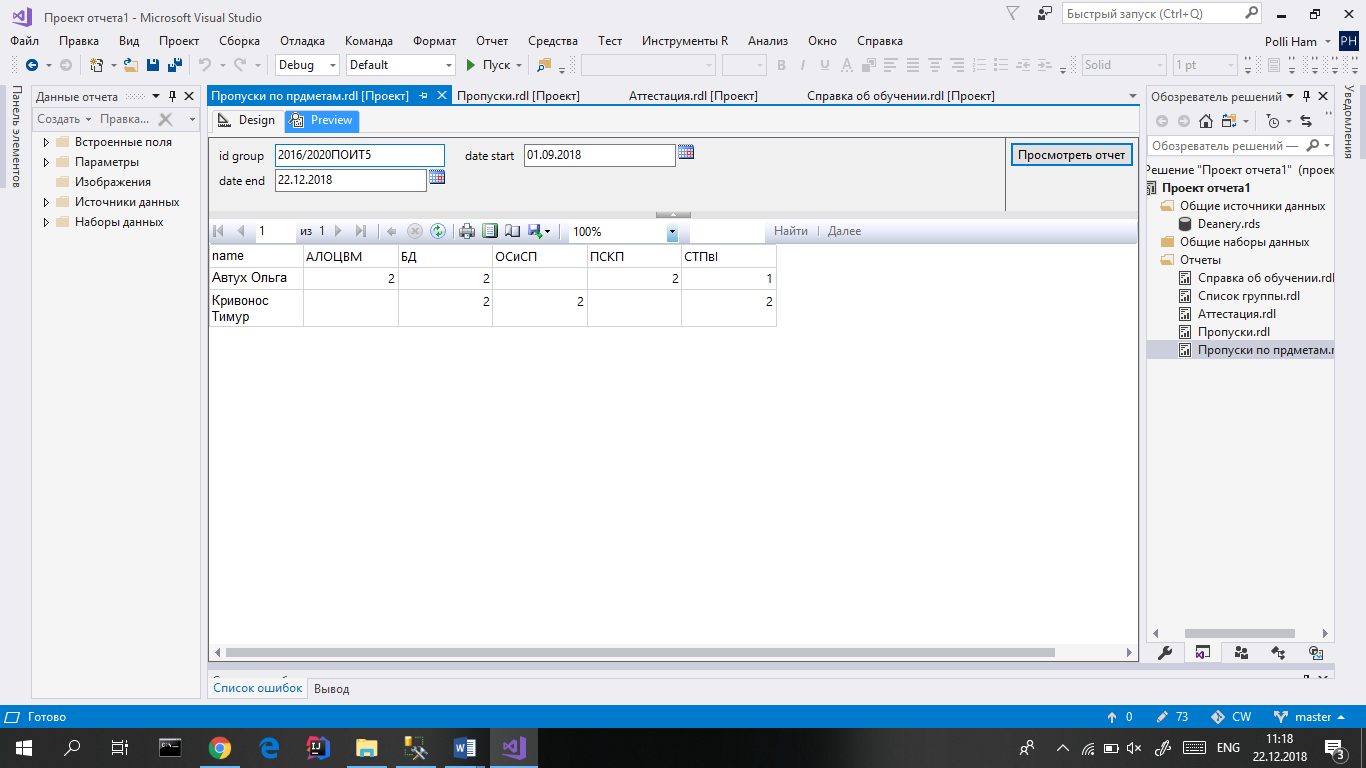


Рисунок 8.4 – Пропуски по предметам

Также для старост будет чрезвычайно полезен отчёт “Справка об обучении” (рисунке 8.5), который по входному параметру (номеру зачётки), формирует справку о том, что студент является обучающимся в БГТУ. Можно экспортировать этот отчёт в текстовый документ или сразу распечатать.

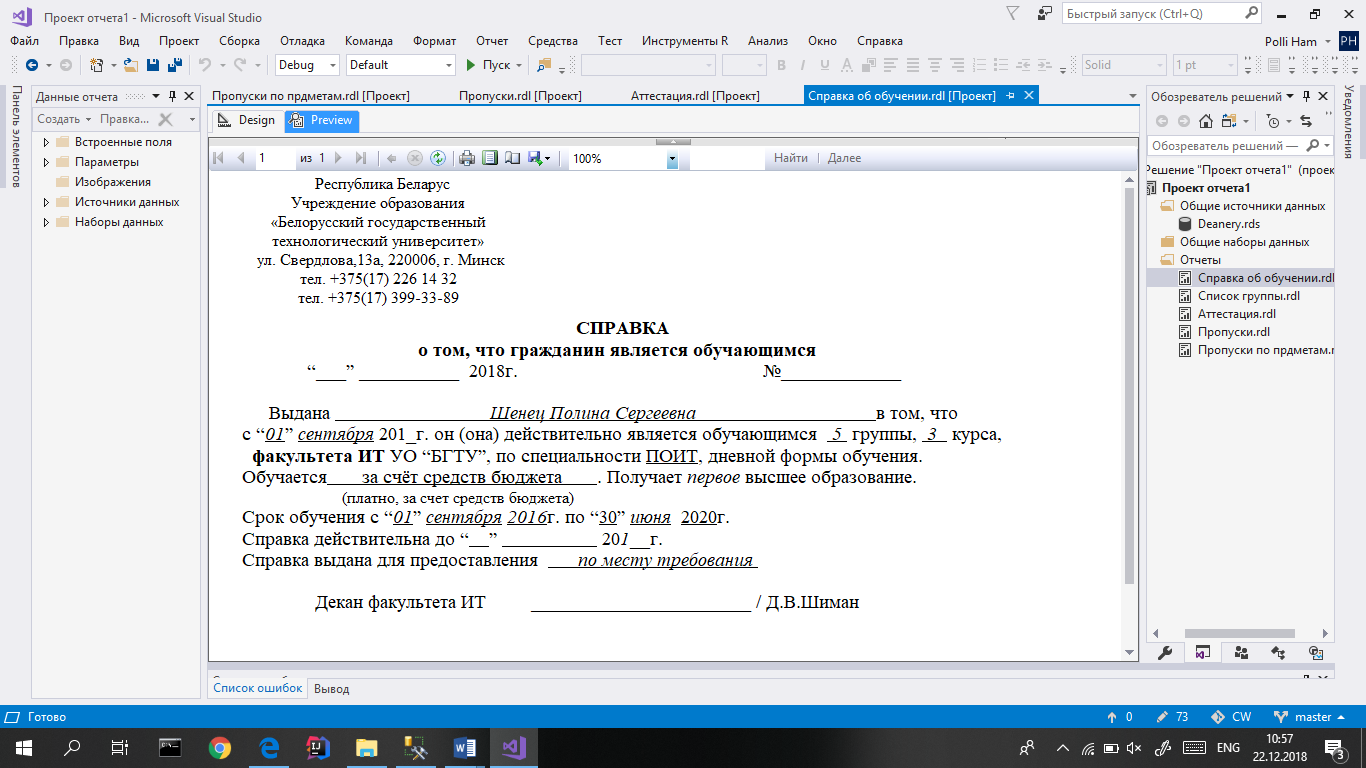


Рисунок 8.5– Справка об обучении

# 9. Заключение

В данном курсовом проекте была разработана база данных деканата, хранимые процедуры, а также настроены пользователь и администратор и их права на выполнение тех или иных процедур.

Были разработаны процедуры для реализации следующих целей:

* Добавление, изменение и удаление информации о студентах, старостах, оценках, преподавателях и пропусках;
* Поиск информации по заданным критериям;
* Запись информации в xml-файл;
* Считывание информации из xml-файла;
* Составление выписок из зачёток и отчётов о успеваемости;
* Составление списков групп;
* Оповещение об академических задолженностях.

Доступ к данным, а также манипулирование ими осуществляется посредством процедур.

# 10. Список используемых источников

1. Понятие и функции системы управления базами данных [Электронный ресурс]. — Режим доступа: <https://vuzlit.ru/998046/ponyatie_funktsii_sistemy_upravleniya_bazami_dannyh>. Дата доступа: 13.12.2018.
2. Примеры массового импорта и экспорта XML-документов (SQL Server) [Электронный ресурс]. — Режим доступа: [https://docs.microsoft.com/ru-ru/sql/relatio nal-databases/import-export/examples-of-bulk-import-and-export-of-xml-documents-sql-server](https://docs.microsoft.com/ru-ru/sql/relatio%20nal-databases/import-export/examples-of-bulk-import-and-export-of-xml-documents-sql-server). Дата доступа: 14.12.2018.
3. Что такое службы SQL Server Reporting Services (SSRS)? [Электронный ресурс]. — Режим доступа: <https://docs.microsoft.com/ru-ru/sql/reporting-services/create-deploy-and-manage-mobile-and-paginated-reports?view=sql-server-2017>. Дата доступа: 18.12.2018.
4. Что такое SSRS? Описание SQL Server Reporting Services. [Электронный ресурс]. — Режим доступа: https://info-comp.ru/softprodobes/478-ssrs-sql-server-reporting-services.html. Дата доступа: 20.12.2018.

# Приложение A

# Приложение Б

use Deanery;

--1 Распоряжения о зачислении

CREATE TABLE Enrollment(

number VARCHAR(15) NOT NULL, --номер распоряжения

date\_sig DATE NOT NULL, --дата подписи

signed NVARCHAR(15) NOT NULL, --кто подписал

CONSTRAINT PK\_EnrollmentNumber PRIMARY KEY (number)

);

--2 Распоряжения об отчислении

CREATE TABLE Expelled(

number VARCHAR(15) NOT NULL, --номер распоряжения

date\_sig DATE NOT NULL, --дата подписи

signed NVARCHAR(15) NOT NULL, --кто подписал

CONSTRAINT PK\_ExpelledNumber PRIMARY KEY (number)

);

--3 Справочник оценок

CREATE TABLE Mark(

id\_mark INT IDENTITY(0, 1), --код оценки

name NVARCHAR(15) NOT NULL, --наименование(0-10, зачтено)

type\_mark BIT NOT NULL, --тип (положительный-1, отрицательный-0)

CONSTRAINT PK\_MarkId PRIMARY KEY (id\_mark)

);

--4 Справочник с признаком в зачётке

CREATE TABLE Attribute(

id\_atr INT IDENTITY(1, 1), --код признака

name NVARCHAR(15) NOT NULL, --наименование(зачёт,экзамен, диф зачёт, курсач)

CONSTRAINT PK\_AttributeId PRIMARY KEY (id\_atr)

);

--5 Справочник профессий

CREATE TABLE Profession(

id\_prof INT IDENTITY(1, 1), --код профессии

name NVARCHAR(30) UNIQUE NOT NULL, --наименование

CONSTRAINT PK\_ProfessionId PRIMARY KEY (id\_prof)

);

--6 Перечень учебных дисциплин

CREATE TABLE Discipline(

id\_disc INT IDENTITY(1, 1), --код дисциплины

short\_name NVARCHAR(20) UNIQUE NOT NULL, --сокращённое название

full\_name NVARCHAR(50) UNIQUE NOT NULL, --полное название

link NVARCHAR(500) --ссылки на ресурсы

CONSTRAINT PK\_DisciplineId PRIMARY KEY (id\_disc)

);

--7 Перечень кафедр

CREATE TABLE Pulpit(

short\_name NVARCHAR(10) NOT NULL, --сокращённое название

full\_name NVARCHAR(80) UNIQUE NOT NULL, --полное название

link NVARCHAR(500) --ссылки на ресурсы

CONSTRAINT PK\_PulpitId PRIMARY KEY (short\_name)

);

--8 Перечень всех преподвавтелей

CREATE TABLE Teacher(

id\_teach INT IDENTITY(1, 1), --код преподавателя

short\_fio NVARCHAR(20) NOT NULL, --сокращённое имя

full\_fio NVARCHAR(50) NOT NULL, --полное имя

pulpit NVARCHAR(10), --кафедра

work BIT NOT NULL, --работает ли ещё(0-нет, 1-да)

CONSTRAINT PK\_TeacherId PRIMARY KEY (id\_teach),

CONSTRAINT FK\_Teacher\_To\_Pulpit FOREIGN KEY (pulpit) REFERENCES Pulpit (short\_name) ON DELETE SET NULL

);

--10 Список областей

CREATE TABLE District(

id\_distr INT IDENTITY(1, 1), --код области

name NVARCHAR(12) UNIQUE, --наименование

CONSTRAINT PK\_DistrictId PRIMARY KEY (id\_distr)

);

--11 Перечень районов

CREATE TABLE Region(

id\_reg INT IDENTITY(1, 1), --код области

name NVARCHAR(20), --наименование

district INT, --область

CONSTRAINT PK\_RegionId PRIMARY KEY (id\_reg),

CONSTRAINT FK\_Region\_To\_District FOREIGN KEY (district) REFERENCES District (id\_distr)

);

--12 Справочник для адресов (тип адреса)

CREATE TABLE TypeAddress(

id\_type INT IDENTITY(1, 1), --код типа

name\_type NVARCHAR(30) UNIQUE NOT NULL, --наименование типа

CONSTRAINT PK\_TypeAddressId PRIMARY KEY (id\_type)

);

--13 Перечень предметов ЦТ

CREATE TABLE SubjectCT(

id\_subj INT IDENTITY(1, 1), --код предмета

name NVARCHAR(20) UNIQUE NOT NULL, --наименование CONSTRAINT PK\_SubjectCTId PRIMARY KEY (id\_subj)

);

--14 Speciality

CREATE TABLE Speciality(

kod\_special NVARCHAR(13) NOT NULL, --код специальности

short\_name NVARCHAR(10) UNIQUE NOT NULL, --сокращённое название

full\_name NVARCHAR(90) UNIQUE NOT NULL, --полное название

specialization NVARCHAR(35) NOT NULL, --специализация

subject\_lenguage1 INT NOT NULL , --предмет ЦТ язык

subject\_lenguage2 INT NOT NULL , --предмет ЦТ язык

subject\_2 INT NOT NULL, --предмет ЦТ (приоритет) subject\_3 INT NOT NULL, --предмет ЦТ

CONSTRAINT PK\_SpecialityId PRIMARY KEY (kod\_special),

CONSTRAINT FK\_SpecialityLen1\_To\_SubjectCT FOREIGN KEY (subject\_lenguage1) REFERENCES SubjectCT (id\_subj),

CONSTRAINT FK\_SpecialityLen2\_To\_SubjectCT FOREIGN KEY (subject\_lenguage2) REFERENCES SubjectCT (id\_subj),

CONSTRAINT FK\_Speciality2\_To\_SubjectCT FOREIGN KEY (subject\_2) REFERENCES SubjectCT (id\_subj),

CONSTRAINT FK\_Speciality3\_To\_SubjectCT FOREIGN KEY (subject\_3) REFERENCES SubjectCT (id\_subj)

);

--15 Список групп

CREATE TABLE GroupStudent(

id\_group NVARCHAR(18) NOT NULL, --код группы

course INT NOT NULL, --курс

group\_number INT NOT NULL, --группа

kod\_special NVARCHAR(13) NOT NULL, --код специальности

begin\_learning DATE NOT NULL, --начало обучения

end\_learning DATE NOT NULL, --конец обучения

disband BIT NOT NULL DEFAULT 0 --расформирована? (0-существует, 1-расформирована)

CONSTRAINT PK\_GroupId PRIMARY KEY (id\_group),

CONSTRAINT FK\_Group\_To\_SubjectCT FOREIGN KEY (kod\_special) REFERENCES Speciality (kod\_special)

);

--9 Ведомости экзаменов и зачётов

CREATE TABLE ExamCredit(

number\_statement VARCHAR(20) NOT NULL, --номер ведомости

attribute INT NOT NULL, --признак

date\_sig DATE NOT NULL, --дата проведения

discipline INT NOT NULL, --код дисциплины

id\_group NVARCHAR(15) NOT NULL, --код группы

teacher\_1 INT NOT NULL, --код преподавателя

teacher\_2 INT DEFAULT NULL, --код второго преподавателя

teacher\_3 INT DEFAULT NULL, --код третьего преподавателя

CONSTRAINT Pk\_ExamCreditNum PRIMARY KEY (number\_statement),

CONSTRAINT FK\_ExamCredit\_To\_Attribute FOREIGN KEY (attribute) REFERENCES Attribute (id\_atr),

CONSTRAINT FK\_ExamCredit\_To\_Discipline FOREIGN KEY (discipline) REFERENCES Discipline (id\_disc),

CONSTRAINT FK\_ExamCredit\_To\_Group FOREIGN KEY (id\_group) REFERENCES GroupStudent (id\_group),

CONSTRAINT FK\_ExamCredit\_To\_Teacher1 FOREIGN KEY (teacher\_1) REFERENCES Teacher (id\_teach),

CONSTRAINT FK\_ExamCredit\_To\_Teacher2 FOREIGN KEY (teacher\_2) REFERENCES Teacher (id\_teach),

CONSTRAINT FK\_ExamCredit\_To\_Teacher3 FOREIGN KEY (teacher\_3) REFERENCES Teacher (id\_teach)

);

--16 Список всех студентов

CREATE TABLE Student(

record\_book INT NOT NULL, --номер зачётной книжки

surname NVARCHAR(15) NOT NULL, --фамилия

first\_name NVARCHAR(20) NOT NULL, --имя

patronymic NVARCHAR(15), --отчество

sex NCHAR(1) CHECK (sex in('Ж','М')) NOT NULL, --пол

date\_birth DATE NOT NULL, --дата рождения

id\_group NVARCHAR(15) NOT NULL, --код группы

form NCHAR(1) CHECK (form in('П','Б')) NOT NULL, --форма обучения

dismissed BIT NOT NULL DEFAULT 0, --был ли отчислен (0-нет, 1-да)

restored BIT NOT NULL DEFAULT 0, --был ли восстановлен (0-нет, 1-да)

leader BIT NOT NULL DEFAULT 0, --является ли старостой (0-нет, 1-да)

CONSTRAINT PK\_StudentId PRIMARY KEY (record\_book),

CONSTRAINT FK\_Student\_To\_Group FOREIGN KEY (id\_group) REFERENCES GroupStudent (id\_group)

);

--17 Результаты ЦТ каждого студента

CREATE TABLE ResultCT(

record\_book INT NOT NULL, --номер зачётной книжки

id\_subj INT NOT NULL, --код предмета ЦТ

score INT CHECK(score between 0 and 100), --балл

year\_ct INT NOT NULL, --год

CONSTRAINT PK\_ResultCT PRIMARY KEY (record\_book,id\_subj),

CONSTRAINT FK\_ResultCT\_To\_Student FOREIGN KEY (record\_book) REFERENCES Student (record\_book),

CONSTRAINT FK\_ResultCT\_To\_SubjectCT FOREIGN KEY (id\_subj) REFERENCES SubjectCT (id\_subj)

);

--18 Список зачисленных студентов по распоряжениям

CREATE TABLE ListEnrollment(

number\_enrol VARCHAR(15) NOT NULL, --номер распоряжения

record\_book INT NOT NULL, --номер зачётной книжки

id\_group NVARCHAR(15) NOT NULL, --код группы

CONSTRAINT PK\_ListEnrollment PRIMARY KEY (number\_enrol,record\_book,id\_group),

CONSTRAINT FK\_ListEnrollment\_To\_SubjectCT FOREIGN KEY (number\_enrol) REFERENCES Enrollment (number),

CONSTRAINT FK\_ListEnrollment\_To\_Student FOREIGN KEY (record\_book) REFERENCES Student (record\_book),

CONSTRAINT FK\_ListEnrollment\_To\_GroupStudent FOREIGN KEY (id\_group) REFERENCES GroupStudent (id\_group)

);

--19 Список отчисленных студентов по распоряжениям

CREATE TABLE ListExpelled(

number\_exp VARCHAR(15) NOT NULL, --номер распоряжения

record\_book INT NOT NULL, --номер зачётной книжки

id\_group NVARCHAR(15) NOT NULL, --код группы

CONSTRAINT PK\_ListExpelled PRIMARY KEY (number\_exp,record\_book,id\_group),

CONSTRAINT FK\_ListExpelled\_To\_SubjectCT FOREIGN KEY (number\_exp) REFERENCES Expelled (number),

CONSTRAINT FK\_ListExpelled\_To\_Student FOREIGN KEY (record\_book) REFERENCES Student (record\_book),

CONSTRAINT FK\_ListExpelled\_To\_GroupStudent FOREIGN KEY (id\_group) REFERENCES GroupStudent (id\_group)

);

--20 Список адресов

CREATE TABLE AddressStudent(

id\_address INT IDENTITY(1, 1), --id

record\_book INT NOT NULL, --номер зачётной книжки

region INT NOT NULL, --район

[address] NVARCHAR(150) NOT NULL, --адресс

type\_addr INT NOT NULL, --тип адреса

CONSTRAINT PK\_AddressStudent PRIMARY KEY (id\_address),

CONSTRAINT FK\_AddressStudent\_To\_Student FOREIGN KEY (record\_book) REFERENCES Student (record\_book),

CONSTRAINT FK\_AddressStudent\_To\_Region FOREIGN KEY (region) REFERENCES Region (id\_reg),

CONSTRAINT FK\_AddressStudent\_To\_TypeAddress FOREIGN KEY (type\_addr) REFERENCES TypeAddress (id\_type)

);

--21 План дисциплин

CREATE TABLE PlanDiscipline(

discipline INT NOT NULL, --дисциплина

speciality NVARCHAR(13) NOT NULL, --специальность

semester INT CHECK(semester between 1 and 10), --семестр

total INT NOT NULL CHECK(total >0) , --всего часов

lecture INT NOT NULL DEFAULT 0 CHECK(lecture >0), --лекции

lab INT NOT NULL DEFAULT 0 CHECK(lab >0 ), --лабораторные

practice INT NOT NULL DEFAULT 0 CHECK(practice >0 ), --практические

seminar INT NOT NULL DEFAULT 0 CHECK(seminar >0 ), --семинары

form INT NOT NULL, --форма аттестации по окончании(экзамен, зачёт)

CONSTRAINT PK\_PlanDiscipline PRIMARY KEY (discipline,speciality,semester),

CONSTRAINT FK\_PlanDiscipline\_To\_Discipline FOREIGN KEY (discipline) REFERENCES Discipline (id\_disc),

CONSTRAINT FK\_PlanDiscipline\_To\_Speciality FOREIGN KEY (speciality) REFERENCES Speciality (kod\_special),

CONSTRAINT FK\_PlanDiscipline\_To\_Attribute FOREIGN KEY (form) REFERENCES Attribute (id\_atr)

);

--22 Результаты экзаменов и зачётов

CREATE TABLE ResultExam(

number\_statement VARCHAR(20) NOT NULL, --номер ведомости,

record\_book INT NOT NULL, --номер зачётной книжки

mark INT NOT NULL, --код оценки

CONSTRAINT PK\_ResultExam PRIMARY KEY (number\_statement, record\_book),

CONSTRAINT FK\_ResultExam\_To\_ExamCredit FOREIGN KEY (number\_statement) REFERENCES ExamCredit (number\_statement),

CONSTRAINT FK\_ResultExam\_To\_Student FOREIGN KEY (record\_book) REFERENCES Student (record\_book),

CONSTRAINT FK\_ResultExam\_To\_Mark FOREIGN KEY (mark) REFERENCES Mark (id\_mark)

);

--23 Лист аттестации

CREATE TABLE Attestation(

id\_attestation INT NOT NULL, --номер аттестации

id\_group NVARCHAR(15) NOT NULL, --код группы

date\_begin DATE NOT NULL, --начало аттестации

date\_end DATE NOT NULL, --конец аттестации

stat BIT NOT NULL DEFAULT 0, --статус аттестации( завершена 1, в процессе 0)

CONSTRAINT PK\_Attestation PRIMARY KEY (id\_attestation ),

CONSTRAINT FK\_Attestation\_To\_GroupStudent FOREIGN KEY (id\_group) REFERENCES GroupStudent (id\_group)

);

--24 Результаты аттестации

CREATE TABLE ResultAttestation(

id\_attestation INT NOT NULL, --номер аттестации

record\_book INT NOT NULL, --номер зачётной книжки

discipline INT NOT NULL, --код дисциплины

teacher INT NOT NULL, --код преподавателя

mark INT NOT NULL, --код оценки

hours\_absent INT NOT NULL CHECK(hours\_absent >=0), --часы пропуска

form NCHAR(2) CHECK(form IN('ЛЗ','ПЗ','КП')), --форма

CONSTRAINT PK\_ResultAttestation PRIMARY KEY (id\_attestation, record\_book, discipline),

CONSTRAINT FK\_ResultAttestation\_To\_Attestation FOREIGN KEY (id\_attestation) REFERENCES Attestation(id\_attestation),

CONSTRAINT FK\_ResultAttestation\_To\_Student FOREIGN KEY (record\_book) REFERENCES Student (record\_book),

CONSTRAINT FK\_ResultAttestation\_To\_Teacher FOREIGN KEY (teacher) REFERENCES Teacher (id\_teach),

CONSTRAINT FK\_ResultAttestation\_To\_Discipline FOREIGN KEY (discipline) REFERENCES Discipline (id\_disc),

CONSTRAINT FK\_ResultAttestation\_To\_Mark FOREIGN KEY (mark) REFERENCES Mark (id\_mark)

);

--25 Пропуски занятий сводная таблица

CREATE TABLE Absents(

id\_abs BIGINT IDENTITY(1,1), --id

record\_book INT NOT NULL, --номер зачётной книжки

discipline INT NOT NULL, --код дисциплины

form NCHAR(2) CHECK(form IN('ЛК','ЛЗ','ПЗ','КП')) NOT NULL, --форма

date\_abs DATE NOT NULL, --дата

count\_hours INT NOT NULL CHECK(count\_hours IN(1,2)), --количество часов

reason BIT NOT NULL DEFAULT 0, --причина пропуска(0-неуважительная, 1-уважительная)

work\_out BIT NOT NULL DEFAULT 0, --отработано ли занятие(0-нет, 1-да)

CONSTRAINT PK\_Absenteeism PRIMARY KEY (id\_abs),

CONSTRAINT FK\_Absenteeism\_To\_Student FOREIGN KEY (record\_book) REFERENCES Student (record\_book),

CONSTRAINT FK\_Absenteeism\_To\_Discipline FOREIGN KEY (discipline) REFERENCES Discipline (id\_disc)

);

--26 Мессенджеры

CREATE TABLE Messenger(

record\_book INT NOT NULL, --номер зачётной книжки

mess NVARCHAR(100) NOT NULL, --мессенджер

CONSTRAINT PK\_Messenger PRIMARY KEY (record\_book,mess),

CONSTRAINT FK\_Messenger\_To\_Student FOREIGN KEY (record\_book) REFERENCES Student (record\_book)

);

--27 Справочник языков

CREATE TABLE Lenguage(

short\_name NVARCHAR(4) UNIQUE NOT NULL , --сокращение

full\_name NVARCHAR(20) UNIQUE NOT NULL, --полное название

CONSTRAINT PK\_Lenguage PRIMARY KEY (short\_name)

);

--28 Аттестат

CREATE TABLE [Certificate](

number INT NOT NULL, --регистрационный номер аттестата

record\_book INT NOT NULL UNIQUE, --номер зачётной книжки

school NVARCHAR(50) NOT NULL, --учереждение образования

gpa FLOAT NOT NULL, --средний балл

belarusian\_language INT CHECK(belarusian\_language between 0 and 10) NOT NULL,

belarusian\_literature INT CHECK(belarusian\_literature between 0 and 10) NOT NULL,

russian\_language INT CHECK(russian\_language between 0 and 10) NOT NULL,

russian\_literature INT CHECK(russian\_literature between 0 and 10) NOT NULL,

foreign\_language INT CHECK(foreign\_language between 0 and 10) NOT NULL,

f\_language NVARCHAR(4) NOT NULL, --справочник языков

mathematics INT CHECK(mathematics between 0 and 10) NOT NULL,

informatics INT CHECK(informatics between 0 and 10) NOT NULL,

history\_of\_belarus INT CHECK(history\_of\_belarus between 0 and 10) NOT NULL,

world\_history INT CHECK(world\_history between 0 and 10) NOT NULL,

social\_science INT CHECK(social\_science between 0 and 10) NOT NULL,

geography\_subj INT CHECK(geography\_subj between 0 and 10) NOT NULL,

biology INT CHECK(biology between 0 and 10) NOT NULL,

physics INT CHECK(physics between 0 and 10) NOT NULL,

astronomy INT CHECK(astronomy between 0 and 10) NOT NULL,

chemistry INT CHECK(chemistry between 0 and 10) NOT NULL,

physical\_culture INT CHECK(physical\_culture between 0 and 10) NOT NULL,

preliminary\_medical INT CHECK(preliminary\_medical between 0 and 10) NOT NULL,

gold\_medal BIT NOT NULL DEFAULT 0, --наличие золотой медали(0-нет, 1-есть)

silver\_medal BIT NOT NULL DEFAULT 0, --наличие серебряной медали(0-нет, 1-есть)

CONSTRAINT PK\_CertificateSt PRIMARY KEY (number),

CONSTRAINT FK\_CertificateSt\_To\_Student FOREIGN KEY (record\_book) REFERENCES Student (record\_book),

CONSTRAINT FK\_CertificateSt\_To\_Lenguage FOREIGN KEY (f\_language) REFERENCES Lenguage (short\_name)

);

--29 Список родителей студентов

CREATE TABLE Parents(

id\_parent INT IDENTITY(1, 1), --id родителя

record\_book INT NOT NULL, --номер зачётной книжки студента

fio NVARCHAR(50) NOT NULL, --фамилия имя отчество

sex NCHAR(1) CHECK (sex in('Ж','М')), --пол

date\_birth DATE NOT NULL, --дата рождения

address\_perent NVARCHAR(150) NOT NULL, --адрес

messenger NVARCHAR(50) NOT NULL, --телефон(ы)

profession INT NOT NULL, --номер профессии

workplace NVARCHAR(150) NOT NULL, --место работы

post NVARCHAR(30) NOT NULL, --должность

CONSTRAINT PK\_ParentsId PRIMARY KEY (id\_parent),

CONSTRAINT FK\_Parents\_To\_Student FOREIGN KEY (record\_book) REFERENCES Student (record\_book),

CONSTRAINT FK\_Parents\_To\_Profession FOREIGN KEY (proffesion) REFERENCES Profession (id\_prof)

);

--30 Дополнительная инфа о студентах

CREATE TABLE AdditionalInfo(

id INT IDENTITY(1, 1), --id

academic\_year CHAR(9) NOT NULL, --номер учебного года

record\_book INT NOT NULL, --номер зачётной книжки студента

hostel BIT NOT NULL DEFAULT 0, --общежитие (0-нет, 1-есть) orphan\_state\_support BIT NOT NULL DEFAULT 0,-сирота на гос обеспечении (0-нет, 1-есть)

orphan\_with\_guardians BIT NOT NULL DEFAULT 0, -сирота с опекунами (0-нет, 1-есть)

invalid BIT NOT NULL DEFAULT 0, --инвалид

lost\_parent BIT NOT NULL DEFAULT 0,-потерял последнего из родителей во время учёбы incomplete\_family BIT NOT NULL DEFAULT 0, --неполная семья

large\_family BIT NOT NULL DEFAULT 0, --многодетная семья

invalid\_parent BIT NOT NULL DEFAULT 0, --родитель инвалид

victims\_of\_the\_Chernobyl\_accident BIT NOT NULL DEFAULT 0 --пострадавший от аварии на ЧАЭС

refugee\_family BIT NOT NULL DEFAULT 0, --семья беженцев

on\_internal\_control BIT NOT NULL DEFAULT 0, --состоящие на внутреннем underachieving\_students BIT NOT NULL DEFAULT 0, --неуспевающий студент

create\_family BIT NOT NULL DEFAULT 0, --студент, создавший семью

have\_children BIT NOT NULL DEFAULT 0, --имеет детей

serious\_chronic\_disease BIT NOT NULL DEFAULT 0, -тяжёлое хроническое заболевание activist BIT NOT NULL DEFAULT 0, --активист молодёжных общественных организаций CONSTRAINT PK\_AdditionalInfoId PRIMARY KEY (id),

CONSTRAINT FK\_AdditionalInfo\_To\_Student FOREIGN KEY (record\_book) REFERENCES Student (record\_book)

);

--31 Отработки неаттестаций

CREATE TABLE ListNotAttestation(

id\_attestation INT, --номер аттестации

record\_book INT NOT NULL, --номер зачётной книжки студента

discipline INT NOT NULL, --код дисциплины

teacher INT NOT NULL, --код преподавателя

mark INT , --код оценки

date\_end DATE NOT NULL, --дата сдачи листа

CONSTRAINT PK\_ListNotAttestationId PRIMARY KEY (id\_attestation,record\_book,discipline),

CONSTRAINT FK\_ListNotAttestation\_To\_Attestation FOREIGN KEY (id\_attestation) REFERENCES Attestation (id\_attestation),

CONSTRAINT FK\_ListNotAttestation\_To\_Discipline FOREIGN KEY (discipline) REFERENCES Discipline (id\_disc),

CONSTRAINT FK\_ListNotAttestation\_To\_Teacher FOREIGN KEY (teacher) REFERENCES Teacher (id\_teach),

CONSTRAINT FK\_ListNotAttestation\_To\_Mark FOREIGN KEY (mark) REFERENCES Mark (id\_mark),

);

--32 Описания форм учебного процесса

CREATE TABLE FormProcess(

id\_form NCHAR(2) NOT NULL, -- 'ОС','ЗЭ','ЗК','ВС','ЛЭ','ПР','ЛК'

name NVARCHAR(30) UNIQUE NOT NULL, -- название

CONSTRAINT PK\_FormProcessId PRIMARY KEY (id\_form)

);

--33 График учебного процесса

CREATE TABLE EducationalProcess(

id INT IDENTITY(1, 1), --id

academic\_year CHAR(9) NOT NULL, --номер учебного года

form NCHAR(2) NOT NULL, --описание

course INT CHECK(course between 1 and 4), --курс

date\_start DATE NOT NULL, --дата начала

date\_end DATE NOT NULL, --дата конца

CONSTRAINT PK\_EducationalProcessId PRIMARY KEY (id),

CONSTRAINT FK\_EducationalProcess\_To\_Discipline FOREIGN KEY (form) REFERENCES FormProcess(id\_form)

);

--34 Пользователи

CREATE TABLE Users(

user\_login NVARCHAR(16) NOT NULL, --логин

user\_password NVARCHAR(20) NOT NULL, --пароль

user\_admin BIT NOT NULL DEFAULT 0, --админ?(0-студент, 1-админ)

CONSTRAINT PK\_UsersId PRIMARY KEY (user\_login)

)

# Приложение В

USE master;

--обычные студенты

CREATE LOGIN studentdb

WITH PASSWORD=N'student',

DEFAULT\_DATABASE=Deanery,

DEFAULT\_LANGUAGE=[русский];

ALTER LOGIN studentdb ENABLE

--старосты

CREATE LOGIN leaderdb

WITH PASSWORD=N'leader',

DEFAULT\_DATABASE=Deanery,

DEFAULT\_LANGUAGE=[русский];

ALTER LOGIN leaderdb ENABLE;

--админы

CREATE LOGIN admindb

WITH PASSWORD=N'admin',

DEFAULT\_DATABASE=Deanery,

DEFAULT\_LANGUAGE=[русский];

use Deanery;

CREATE USER student FOR LOGIN studentdb;

CREATE USER leader FOR LOGIN leaderdb;

CREATE USER [admin] FOR LOGIN admindb;

-----------------------------------Admin\_Role-----------------------------------------

CREATE ROLE Admin\_Role;

GRANT EXECUTE ON [dbo].[ListEnrollmentInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ResultExamUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ListEnrollmentDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ListExpelledInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ListExpelledDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ParentsByStudent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AbsentsInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ParentsInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[StudentByRecordBook] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[CertificateDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ResultExamByNumber] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[CertificateInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ResultExamByStudent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[MessengerByStudent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ExamCreditInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AttributeInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[MessengerByCourseGroup] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AttributeUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[MessengerGroupByStudent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AttributeDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[DistrictSelect] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[NotAttestationById] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[DistrictInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[CurentNotAttestationByIdGroup] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[DistrictUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[CurentNotAttestation] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[DistrictDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[NotFixAttestationById] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[GetGroupIdByCourseNum] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[TypeAddressSelect] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ListExpelledByStudent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[TypeAddressInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ListExpelledByNumber] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AttestationByIdGroup] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[TypeAddressUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ListEnrollmentByNumber] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[CurentAttestation] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[TypeAddressDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ListEnrollmentByStudent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AttestationInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[SubjectCTSelect] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[SubjectCTInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ExamCreditByNumber] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[SubjectCTUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[CertificateByStudent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AttestationDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[SubjectCTDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AdditionalInfoByStudent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[LenguageSelect] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AdditionalInfoByGroup] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[LenguageInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AdditionalInfoByIdGroup] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[LenguageUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[StudentUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[LenguageDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ParentsDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[FormProcessSelect] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[SearchStudent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[StudentInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ResultAttestationByStydent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[FormProcessInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[FormProcessUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[FormProcessDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[MessengerInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[MessengerDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[EducationalProcessInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ExpelledUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ExamCreditDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[EducationalProcessDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ResultAttestationByNumber] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[EnrollmentSelect] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ExpelledSelect] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[UsersSelect] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[EnrollmentInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[MarkSelect] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[UsersInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ProfessionSelect] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[EnrollmentUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[UsersUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[PulpitSelect] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[NotAttestationByIdGroup] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[UsersDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[EnrollmentDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[TeacherSelect] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AbsentsUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AbsentsDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AlterAttestationStatus] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[DisciplineDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AddressStudentInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ExpelledDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[CurenNotAttestationByGroupNum] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ExpelledInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AddressStudentDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AttestationByGroupNum] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[MarkInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AttestationUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[MarkUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AdditionalInfoUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ResultExamInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[MarkDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AttributeSelect] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[CertificateUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[EducationalProcessUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ResultExamDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ParentsUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ProfessionInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[TeacherUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ProfessionUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ProfessionDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ResultCTByStudent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[PulpitInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[PulpitUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[PlanDisciplineBySpeciality] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[PulpitDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[PlanDisciplineByDiscipline] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[PlanDisciplineBySemester] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ResultAttestationInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ExtractFromRecordBook] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ResultAttestationUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ResultAttestationDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[StudentByRecordBookForCertificate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[NotAttestationUpdateMark] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[GetGroupIdByStudent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[NotAttestationDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[EducationalProcessSelectByYear] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[EducationalProcessSelectByCourse] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[DisciplineSelect] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[DisciplineInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[DisciplineUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AbsentsByStudent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[UserIsAdmin] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[PlanDisciplineInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[StudentIsLeader] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[PlanDisciplineUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AbsentsRespectfulByIdStudent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[PlanDisciplineDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AbsentsNotRespectfulByIdStudent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[Authorisation] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[GetGroupById] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AttestationByStudent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[GetNotDisbandGroup] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AddressByStudent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[GetDisbandGroup] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[GroupInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AddressByIdGroup] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[TeacherDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[GroupDisbandUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[GroupCourseUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[GroupDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[GetGroupByCourseNum] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[TeacherInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AbsentsByIdGroup] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[GeneralAbsentsByStudent] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AdditionalInfoInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ResultCTInsert] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ResultCTUpdate] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[NotFixAbsentsByIdGroup] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[ResultCTDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[GeneralAbsentsByIdGroup] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AdditionalInfoDelete] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AbsentsNotRespectfulByIdGroup] TO [Admin\_Role];

GRANT EXECUTE ON [dbo].[AbsentsRespectfulByIdGroup] TO [Admin\_Role];

go

sp\_addrolemember 'Admin\_Role', 'admin';

------------------------------------------Leader\_Role-------------------------------------------

CREATE ROLE Leader\_Role;

GRANT EXECUTE ON [dbo].[ResultExamUpdate] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ParentsByStudent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AbsentsInsert] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ParentsInsert] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[StudentByRecordBook] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ResultExamByNumber] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ResultExamByStudent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[MessengerByStudent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[MessengerByCourseGroup] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[MessengerGroupByStudent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[DistrictSelect] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[NotAttestationById] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[CurentNotAttestationByIdGroup] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[CurentNotAttestation] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[NotFixAttestationById] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[GetGroupIdByCourseNum] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[TypeAddressSelect] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ListExpelledByStudent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ListExpelledByNumber] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AttestationByIdGroup] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ListEnrollmentByNumber] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[CurentAttestation] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ListEnrollmentByStudent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AttestationInsert] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[SubjectCTSelect] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ExamCreditByNumber] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[CertificateByStudent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AdditionalInfoByStudent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[LenguageSelect] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AdditionalInfoByGroup] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AdditionalInfoByIdGroup] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ParentsDelete] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[FormProcessSelect] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[SearchStudent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ResultAttestationByStydent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[MessengerInsert] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[MessengerDelete] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ResultAttestationByNumber] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[EnrollmentSelect] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ExpelledSelect] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[MarkSelect] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ProfessionSelect] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[PulpitSelect] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[NotAttestationByIdGroup] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[TeacherSelect] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AbsentsUpdate] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AbsentsDelete] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AddressStudentInsert] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[CurenNotAttestationByGroupNum] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AddressStudentDelete] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AttestationByGroupNum] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AdditionalInfoUpdate] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ResultExamInsert] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AttributeSelect] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ResultExamDelete] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ParentsUpdate] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ResultCTByStudent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[PlanDisciplineBySpeciality] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[PlanDisciplineByDiscipline] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[PlanDisciplineBySemester] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ResultAttestationInsert] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ExtractFromRecordBook] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ResultAttestationUpdate] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[ResultAttestationDelete] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[StudentByRecordBookForCertificate] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[NotAttestationUpdateMark] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[GetGroupIdByStudent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[EducationalProcessSelectByYear] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[EducationalProcessSelectByCourse] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[DisciplineSelect] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AbsentsByStudent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[StudentIsLeader] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AbsentsRespectfulByIdStudent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AbsentsNotRespectfulByIdStudent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[GetGroupById] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AttestationByStudent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[GetNotDisbandGroup] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AddressByStudent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AddressByIdGroup] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[GetGroupByCourseNum] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AbsentsByIdGroup] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[GeneralAbsentsByStudent] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AdditionalInfoInsert] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[NotFixAbsentsByIdGroup] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[GeneralAbsentsByIdGroup] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AdditionalInfoDelete] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AbsentsNotRespectfulByIdGroup] TO [Leader\_Role];

GRANT EXECUTE ON [dbo].[AbsentsRespectfulByIdGroup] TO [Leader\_Role];

go

sp\_addrolemember 'Leader\_Role', 'leader';

------------------------------------------Student\_Role------------------------------------------

CREATE ROLE Student\_Role;

GRANT EXECUTE ON [dbo].[ParentsByStudent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[StudentByRecordBook] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[ResultExamByStudent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[MessengerByStudent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[MessengerGroupByStudent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[DistrictSelect] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[NotAttestationById] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[CurentNotAttestationByIdGroup] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[CurentNotAttestation] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[NotFixAttestationById] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[GetGroupIdByCourseNum] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[TypeAddressSelect] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[ListExpelledByStudent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[ListExpelledByNumber] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[AttestationByIdGroup] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[ListEnrollmentByNumber] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[CurentAttestation] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[ListEnrollmentByStudent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[SubjectCTSelect] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[ExamCreditByNumber] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[CertificateByStudent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[AdditionalInfoByStudent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[LenguageSelect] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[AdditionalInfoByGroup] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[AdditionalInfoByIdGroup] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[FormProcessSelect] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[SearchStudent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[ResultAttestationByStydent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[MessengerInsert] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[ResultAttestationByNumber] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[MarkSelect] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[ProfessionSelect] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[PulpitSelect] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[NotAttestationByIdGroup] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[TeacherSelect] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[CurenNotAttestationByGroupNum] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[AttestationByGroupNum] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[AttributeSelect] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[ResultCTByStudent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[PlanDisciplineBySpeciality] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[PlanDisciplineByDiscipline] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[PlanDisciplineBySemester] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[ExtractFromRecordBook] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[StudentByRecordBookForCertificate] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[NotAttestationUpdateMark] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[GetGroupIdByStudent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[EducationalProcessSelectByYear] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[EducationalProcessSelectByCourse] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[DisciplineSelect] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[AbsentsByStudent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[StudentIsLeader] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[AbsentsRespectfulByIdStudent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[AbsentsNotRespectfulByIdStudent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[GetGroupById] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[AttestationByStudent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[GetNotDisbandGroup] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[AddressByStudent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[AddressByIdGroup] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[GetGroupByCourseNum] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[AbsentsByIdGroup] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[GeneralAbsentsByStudent] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[NotFixAbsentsByIdGroup] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[GeneralAbsentsByIdGroup] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[AbsentsNotRespectfulByIdGroup] TO [Student\_Role];

GRANT EXECUTE ON [dbo].[AbsentsRespectfulByIdGroup] TO [Student\_Role];

go

sp\_addrolemember 'Student\_Role', 'student';

# Приложение Г

CREATE PROC [dbo].[AbsentsByIdGroup]

@id\_group NVARCHAR(18),

@date\_start DATE,

@date\_end DATE

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT s.surname + ' '+ s.first\_name as name, d.short\_name as discipline ,

a.form, a.date\_abs, a.count\_hours, a.reason, a.work\_out

FROM dbo.Absents a inner join dbo.Student s on a.record\_book = s.record\_book

inner join dbo.Discipline d on a.discipline = d.id\_disc

inner join dbo.GroupStudent g on s.id\_group = g.id\_group

WHERE (a.date\_abs BETWEEN @date\_start AND @date\_end ) AND( LOWER(g.id\_group) like LOWER(@id\_group));

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AbsentsByStudent] Script Date: 22.12.2018 9:56:15 CREATE PROC [dbo].[AbsentsByStudent]

@record\_book INT,

@date\_start DATE,

@date\_end DATE

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT d.short\_name as discipline , a.form, a.date\_abs, a.count\_hours, a.reason, a.work\_out

FROM dbo.Absents a inner join dbo.Student s on a.record\_book = s.record\_book

inner join dbo.Discipline d on a.discipline = d.id\_disc

inner join dbo.GroupStudent g on s.id\_group = g.id\_group

WHERE a.record\_book=@record\_book AND ( a.date\_abs between @date\_start and @date\_end);

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AbsentsDelete] Script Date: 22.12.2018 9:56:15

CREATE PROC [dbo].[AbsentsDelete]

@rc INT OUTPUT,

@record\_book INT,

@discipline NVARCHAR(20),

@form NCHAR(2),

@date\_abs DATE

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @id\_discipline INT = 0;

SELECT @id\_discipline = id\_disc FROM dbo.Discipline WHERE LOWER(short\_name) like LOWER(@discipline);

IF(EXISTS(SELECT record\_book FROM dbo.Absents WHERE record\_book=@record\_book AND discipline=@id\_discipline AND

LOWER(form)LIKE LOWER(@form) AND date\_abs= @date\_abs))

BEGIN

BEGIN TRAN;

DELETE

FROM dbo.Absents

WHERE record\_book=@record\_book AND discipline=@id\_discipline AND

LOWER(form)LIKE LOWER(@form) AND date\_abs= @date\_abs;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AbsentsInsert] Script Date: 22.12.2018 9:56:15

CREATE PROC [dbo].[AbsentsInsert]

@rc INT OUTPUT,

@record\_book INT,

@discipline NVARCHAR(20),

@form NCHAR(2),

@date\_abs DATE,

@hours INT =2,

@reason BIT = 0

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET DATEFIRST 1;

SET @rc = 0;

DECLARE @id\_discipline INT = 0;

SELECT @id\_discipline = id\_disc FROM dbo.Discipline WHERE LOWER(short\_name) like LOWER(@discipline);

DECLARE @course INT = 0;

SELECT @course=g.course FROM dbo.Student s inner join dbo.GroupStudent g on s.id\_group = g.id\_group WHERE record\_book=@record\_book;

IF( (NOT EXISTS(SELECT record\_book FROM dbo.Absents WHERE record\_book=@record\_book AND discipline=@id\_discipline AND

LOWER(form)LIKE LOWER(@form) AND date\_abs= @date\_abs)) AND @course!=0 AND @id\_discipline!=0 AND

(@form IN('ЛК','ЛЗ','ПЗ','КП')) AND @date\_abs<=GETDATE() AND (@hours IN(1,2)) AND

DATEPART(weekday, @date\_abs) != 7 AND

(SELECT form FROM dbo.EducationalProcess e WHERE (@date\_abs between date\_start and date\_end) AND course = @course) in ('ОС','ВС')

)

BEGIN

DECLARE @work\_out BIT = 0;

IF (@form IN('ЛК','ПЗ','КП'))

SET @work\_out = 1;

BEGIN TRAN;

INSERT INTO dbo.Absents(record\_book, discipline, form, date\_abs, count\_hours, reason, work\_out)

VALUES(@record\_book, @id\_discipline, LOWER(@form), @date\_abs, @hours, @reason, @work\_out);

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AbsentsNotRespectfulByIdGroup

CREATE PROC [dbo].[AbsentsNotRespectfulByIdGroup]

@id\_group NVARCHAR(18),

@date\_start DATE,

@date\_end DATE

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT s.surname + ' '+ s.first\_name as name, SUM(count\_hours) as count\_hours

FROM dbo.Absents a inner join dbo.Student s on a.record\_book = s.record\_book

inner join dbo.GroupStudent g on s.id\_group = g.id\_group

WHERE (a.date\_abs BETWEEN @date\_start AND @date\_end ) AND g.id\_group=@id\_group

GROUP BY a.record\_book, s.first\_name, s.surname, a.reason

HAVING a.reason=0;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AbsentsNotRespectfulByIdStudent

CREATE PROC [dbo].[AbsentsNotRespectfulByIdStudent]

@record\_book INT,

@date\_start DATE,

@date\_end DATE

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT SUM(count\_hours)

FROM dbo.Absents

WHERE record\_book=@record\_book AND (date\_abs between @date\_start and @date\_end) AND reason=0;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AbsentsRespectfulByIdGroup]

CREATE PROC [dbo].[AbsentsRespectfulByIdGroup]

@id\_group NVARCHAR(18),

@date\_start DATE,

@date\_end DATE

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT s.surname + ' '+ s.first\_name as name, SUM(count\_hours) as count\_hours

FROM dbo.Absents a inner join dbo.Student s on a.record\_book = s.record\_book

inner join dbo.GroupStudent g on s.id\_group = g.id\_group

WHERE (a.date\_abs BETWEEN @date\_start AND @date\_end ) AND g.id\_group=@id\_group

GROUP BY a.record\_book, s.first\_name, s.surname, a.reason

HAVING a.reason=1;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AbsentsRespectfulByIdStudent

CREATE PROC [dbo].[AbsentsRespectfulByIdStudent]

@record\_book INT,

@date\_start DATE,

@date\_end DATE

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT SUM(count\_hours)

FROM dbo.Absents

WHERE record\_book=@record\_book AND (date\_abs between @date\_start and @date\_end) AND reason=1;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AbsentsUpdate]

CREATE PROC [dbo].[AbsentsUpdate]

@rc INT OUTPUT,

@record\_book INT,

@discipline NVARCHAR(20),

@form NCHAR(2),

@date\_abs DATE,

@hours INT =NULL,

@reason BIT = NULL,

@work\_out BIT = NULL

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @id\_discipline INT = 0;

SELECT @id\_discipline = id\_disc FROM dbo.Discipline WHERE LOWER(short\_name) like LOWER(@discipline);

IF(EXISTS(SELECT record\_book FROM dbo.Absents WHERE record\_book=@record\_book AND discipline=@id\_discipline AND

LOWER(form)LIKE LOWER(@form) AND @date\_abs= @date\_abs))

BEGIN

IF (@hours IS NOT NULL AND @hours IN(1,2))

BEGIN

BEGIN TRAN;

UPDATE dbo.Absents

SET count\_hours = @hours

WHERE record\_book=@record\_book AND discipline=@id\_discipline AND( LOWER(form)LIKE LOWER(@form)) AND date\_abs= @date\_abs;

COMMIT;

SET @rc = 1;

END

IF (@reason IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.Absents

SET reason = @reason

WHERE record\_book=@record\_book AND discipline=@id\_discipline AND LOWER(form)LIKE LOWER(@form) AND date\_abs= @date\_abs;

COMMIT;

SET @rc = 1;

END

IF (@work\_out IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.Absents

SET work\_out = @work\_out

WHERE record\_book=@record\_book AND discipline=@id\_discipline AND LOWER(form)LIKE LOWER(@form) AND date\_abs= @date\_abs;

COMMIT;

SET @rc = 1;

END

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AdditionalInfoByGroup

CREATE PROC [dbo].[AdditionalInfoByGroup]

@course INT,

@group INT,

@academic\_year CHAR(9)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT s.first\_name +' '+ s.surname +' '+ ISNULL(s.patronymic,'') as name, hostel, orphan\_state\_support, orphan\_with\_guardians,

invalid, lost\_parent, incomplete\_family, large\_family, invalid\_parent, victims\_of\_the\_Chernobyl\_accident, refugee\_family,

on\_internal\_control, underachieving\_students, create\_family, have\_children, serious\_chronic\_disease, activist

FROM dbo.AdditionalInfo a inner join dbo.Student s on a.record\_book = s.record\_book

inner join dbo.GroupStudent g on s.id\_group = g.id\_group

WHERE g.course = @course AND g.group\_number = @group AND g.disband = 0 AND a.academic\_year = @academic\_year;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AdditionalInfoByIdGroup]

CREATE PROC [dbo].[AdditionalInfoByIdGroup]

@id\_group NVARCHAR(15),

@academic\_year CHAR(9)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT s.first\_name +' '+ s.surname +' '+ ISNULL(s.patronymic,'') as name, hostel, orphan\_state\_support, orphan\_with\_guardians,

invalid, lost\_parent, incomplete\_family, large\_family, invalid\_parent, victims\_of\_the\_Chernobyl\_accident, refugee\_family,

on\_internal\_control, underachieving\_students, create\_family, have\_children, serious\_chronic\_disease, activist

FROM dbo.AdditionalInfo a inner join dbo.Student s on a.record\_book = s.record\_book

inner join dbo.GroupStudent g on s.id\_group = g.id\_group

WHERE LOWER(g.id\_group) like ('%'+LOWER(@id\_group)+'%') AND a.academic\_year = @academic\_year ;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AdditionalInfoByStudent

CREATE PROC [dbo].[AdditionalInfoByStudent]

@record\_book INT,

@academic\_year CHAR(9) = NULL

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT a.academic\_year,s.first\_name +' '+ s.surname +' '+ ISNULL(s.patronymic,'') as name, hostel, orphan\_state\_support, orphan\_with\_guardians,

invalid, lost\_parent, incomplete\_family, large\_family, invalid\_parent, victims\_of\_the\_Chernobyl\_accident, refugee\_family,

on\_internal\_control, underachieving\_students, create\_family, have\_children, serious\_chronic\_disease, activist

FROM dbo.AdditionalInfo a inner join dbo.Student s on a.record\_book = s.record\_book

WHERE ( a.record\_book = @record\_book AND a.academic\_year = @academic\_year) OR (@academic\_year IS NULL);

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AdditionalInfoDelete

CREATE PROC [dbo].[AdditionalInfoDelete]

@rc INT OUTPUT,

@record\_book INT,

@academic\_year CHAR(9)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT record\_book FROM dbo.AdditionalInfo WHERE record\_book = @record\_book AND academic\_year = @academic\_year ))

BEGIN

BEGIN TRAN;

DELETE

dbo.AdditionalInfo WHERE record\_book = @record\_book AND academic\_year = @academic\_year ;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AdditionalInfoInsert

CREATE PROC [dbo].[AdditionalInfoInsert]

@rc INT OUTPUT,

@record\_book INT,

@academic\_year CHAR(9),

@hostel BIT = 0,

@orphan\_state\_support BIT = 0,

@orphan\_with\_guardians BIT = 0,

@invalid BIT = 0,

@lost\_parent BIT = 0,

@incomplete\_family BIT = 0,

@large\_family BIT = 0,

@invalid\_parent BIT = 0,

@victims\_of\_the\_Chernobyl\_accident BIT = 0,

@refugee\_family BIT = 0,

@on\_internal\_control BIT = 0,

@underachieving\_students BIT = 0,

@create\_family BIT = 0,

@have\_children BIT = 0,

@serious\_chronic\_disease BIT = 0,

@activist BIT = 0

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT record\_book FROM dbo.Student WHERE record\_book = @record\_book) AND NOT

EXISTS(SELECT record\_book FROM dbo.AdditionalInfo WHERE record\_book = @record\_book AND academic\_year = @academic\_year) )

BEGIN

BEGIN TRAN;

INSERT INTO dbo.AdditionalInfo(academic\_year,record\_book, hostel, orphan\_state\_support ,orphan\_with\_guardians, invalid,

lost\_parent,incomplete\_family,large\_family,invalid\_parent,victims\_of\_the\_Chernobyl\_accident,refugee\_family,

on\_internal\_control,underachieving\_students,create\_family,have\_children,serious\_chronic\_disease,activist)

VALUES (@academic\_year,@record\_book, @hostel, @orphan\_state\_support,@orphan\_with\_guardians, @invalid,

@lost\_parent, @incomplete\_family,@large\_family,@invalid\_parent,@victims\_of\_the\_Chernobyl\_accident,@refugee\_family,

@on\_internal\_control,@underachieving\_students,@create\_family,@have\_children,@serious\_chronic\_disease,@activist);

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AdditionalInfoUpdate

CREATE PROC [dbo].[AdditionalInfoUpdate]

@rc INT OUTPUT,

@record\_book INT,

@academic\_year CHAR(9),

@hostel BIT = NULL,

@orphan\_state\_support BIT = NULL,

@orphan\_with\_guardians BIT = NULL,

@invalid BIT = NULL,

@lost\_parent BIT = NULL,

@incomplete\_family BIT = NULL,

@large\_family BIT = NULL,

@invalid\_parent BIT = NULL,

@victims\_of\_the\_Chernobyl\_accident BIT = NULL,

@refugee\_family BIT = NULL,

@on\_internal\_control BIT = NULL,

@underachieving\_students BIT = NULL,

@create\_family BIT = NULL,

@have\_children BIT = NULL,

@serious\_chronic\_disease BIT = NULL,

@activist BIT = NULL

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT record\_book FROM dbo.AdditionalInfo WHERE record\_book = @record\_book AND academic\_year = @academic\_year ))

BEGIN

IF(@hostel IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.AdditionalInfo SET hostel = @hostel WHERE record\_book = @record\_book AND academic\_year = @academic\_year;

COMMIT;

SET @rc = 1;

END

IF(@orphan\_state\_support IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.AdditionalInfo SET orphan\_state\_support = @orphan\_state\_support

WHERE record\_book = @record\_book AND academic\_year = @academic\_year;

COMMIT;

SET @rc = 1;

END

IF(@orphan\_with\_guardians IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.AdditionalInfo SET orphan\_with\_guardians = @orphan\_with\_guardians

WHERE record\_book = @record\_book AND academic\_year = @academic\_year;

COMMIT;

SET @rc = 1;

END

IF(@invalid IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.AdditionalInfo SET invalid = @invalid WHERE record\_book = @record\_book AND academic\_year = @academic\_year;

COMMIT;

SET @rc = 1;

END

IF(@lost\_parent IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.AdditionalInfo SET lost\_parent = @lost\_parent WHERE record\_book = @record\_book AND academic\_year = @academic\_year;

COMMIT;

SET @rc = 1;

END

IF(@incomplete\_family IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.AdditionalInfo SET incomplete\_family = @incomplete\_family

WHERE record\_book = @record\_book AND academic\_year = @academic\_year;

COMMIT;

SET @rc = 1;

END

IF(@large\_family IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.AdditionalInfo SET large\_family = @large\_family WHERE record\_book = @record\_book AND academic\_year = @academic\_year;

COMMIT;

SET @rc = 1;

END

IF(@invalid\_parent IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.AdditionalInfo SET invalid\_parent = @invalid\_parent WHERE record\_book = @record\_book AND academic\_year = @academic\_year;

COMMIT;

SET @rc = 1;

END

IF(@victims\_of\_the\_Chernobyl\_accident IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.AdditionalInfo SET victims\_of\_the\_Chernobyl\_accident = @victims\_of\_the\_Chernobyl\_accident

WHERE record\_book = @record\_book AND academic\_year = @academic\_year;

COMMIT;

SET @rc = 1;

END

IF((@refugee\_family IS NULL))

BEGIN

BEGIN TRAN;

UPDATE dbo.AdditionalInfo SET refugee\_family = @refugee\_family WHERE record\_book = @record\_book AND academic\_year = @academic\_year;

COMMIT;

SET @rc = 1;

END

IF(@on\_internal\_control IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.AdditionalInfo SET on\_internal\_control = @on\_internal\_control

WHERE record\_book = @record\_book AND academic\_year = @academic\_year;

COMMIT;

SET @rc = 1;

END

IF(@underachieving\_students IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.AdditionalInfo SET underachieving\_students = @underachieving\_students

WHERE record\_book = @record\_book AND academic\_year = @academic\_year;

COMMIT;

SET @rc = 1;

END

IF(@create\_family IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.AdditionalInfo SET create\_family = @create\_family WHERE record\_book = @record\_book AND academic\_year = @academic\_year;

COMMIT;

SET @rc = 1;

END

IF(NOT (@have\_children IS NULL))

BEGIN

BEGIN TRAN;

UPDATE dbo.AdditionalInfo SET have\_children = @have\_children WHERE record\_book = @record\_book AND academic\_year = @academic\_year;

COMMIT;

SET @rc = 1;

END

IF(@serious\_chronic\_disease IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.AdditionalInfo SET serious\_chronic\_disease = @serious\_chronic\_disease

WHERE record\_book = @record\_book AND academic\_year = @academic\_year;

COMMIT;

SET @rc = 1;

END

IF(@activist IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.AdditionalInfo SET activist = @activist WHERE record\_book = @record\_book AND academic\_year = @academic\_year;

COMMIT;

SET @rc = 1;

END

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AddressByIdGroup

CREATE PROC [dbo].[AddressByIdGroup]

@id\_group NVARCHAR(18)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT (s.surname + ' ' + s.first\_name +' '+ ISNULL(s.patronymic,'')) AS name,

r.name as region, d.name as district, a.[address], t.name\_type

FROM dbo.AddressStudent a inner join dbo.Region r on a.region = r.id\_reg

inner join dbo.District d on r.district = d.id\_distr

inner join dbo.TypeAddress t on a.type\_addr = t.id\_type

inner join dbo.Student s on a.record\_book = s.record\_book

WHERE s.id\_group = @id\_group;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AddressByStudent]

CREATE PROC [dbo].[AddressByStudent]

@record\_book INT

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT r.name as region, d.name as district, a.[address], t.name\_type

FROM dbo.AddressStudent a inner join dbo.Region r on a.region = r.id\_reg

inner join dbo.District d on r.district = d.id\_distr

inner join dbo.TypeAddress t on a.type\_addr = t.id\_type

WHERE record\_book = @record\_book;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AddressStudentDelete]

CREATE PROC [dbo].[AddressStudentDelete]

@rc INT OUTPUT,

@record\_book INT,

@type\_addr INT

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT record\_book FROM dbo.AddressStudent WHERE record\_book = @record\_book AND type\_addr=@type\_addr))

BEGIN

BEGIN TRAN;

DELETE

FROM dbo.AddressStudent

WHERE record\_book = @record\_book AND type\_addr=@type\_addr;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AddressStudentInsert

CREATE PROC [dbo].[AddressStudentInsert]

@rc INT OUTPUT,

@record\_book INT,

@region\_name NVARCHAR(20) = '',

@address NVARCHAR(150),

@type\_addr INT

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @id\_region INT = 0;

SELECT @id\_region = id\_reg FROM dbo.Region WHERE LOWER(name) like LOWER(@region\_name);

IF(@region\_name IS NULL)

SELECT @id\_region = id\_reg FROM dbo.Region WHERE name='';

IF( EXISTS(SELECT record\_book FROM dbo.Student WHERE record\_book = @record\_book) AND

EXISTS(SELECT id\_type FROM dbo.TypeAddress WHERE id\_type = @type\_addr) )

BEGIN

BEGIN TRAN;

INSERT INTO dbo.AddressStudent (record\_book, region,[address], type\_addr)

VALUES (@record\_book, @id\_region, @address, @type\_addr);

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AlterAttestationStatus

CREATE PROC [dbo].[AlterAttestationStatus]

@rc INT OUTPUT,

@id\_attestation INT,

@date\_end\_fix DATE

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF(EXISTS(SELECT id\_attestation FROM dbo.Attestation WHERE id\_attestation = @id\_attestation) )

BEGIN

BEGIN TRAN;

UPDATE dbo.Attestation

SET stat = 1

WHERE id\_attestation = @id\_attestation;

COMMIT;

DECLARE @record\_book INT,@discipline INT, @teacher INT,@mark INT;

DECLARE curs CURSOR LOCAL DYNAMIC

FOR

SELECT record\_book, discipline, teacher, mark

FROM dbo.ResultAttestation

WHERE id\_attestation = @id\_attestation;

OPEN curs;

FETCH NEXT FROM curs

INTO @record\_book, @discipline, @teacher, @mark;

WHILE @@fetch\_status = 0

BEGIN

IF((SELECT type\_mark FROM dbo.Mark WHERE id\_mark = @mark) = 0)

BEGIN

BEGIN TRAN;

INSERT INTO dbo.ListNotAttestation (id\_attestation, record\_book, discipline, teacher, mark, date\_end)

VALUES(@id\_attestation, @record\_book, @discipline, @teacher, NULL, @date\_end\_fix);

COMMIT;

END

FETCH NEXT FROM curs

INTO @record\_book, @discipline, @teacher, @mark;

END;

CLOSE curs;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AttestationByGroupNum

CREATE PROC [dbo].[AttestationByGroupNum]

@course INT,

@group INT

AS

SET XACT\_ABORT, NOCOUNT ON;

DECLARE @id NVARCHAR(15) = NULL;

EXEC dbo.GetGroupIdByCourseNum @course, @group, @id OUT;

IF(@id IS NOT NULL)

EXEC dbo.AttestationByIdGroup @id;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AttestationByIdGroup

CREATE PROC [dbo].[AttestationByIdGroup]

@id\_group NVARCHAR(18)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT a.id\_attestation, date\_begin, date\_end, stat

FROM dbo.Attestation a inner join dbo.GroupStudent g on a.id\_group = g.id\_group

WHERE LOWER(g.id\_group) like ('%'+LOWER(@id\_group)+'%');

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AttestationByStudent]

CREATE PROC [dbo].[AttestationByStudent]

@record\_book INT

AS

SET XACT\_ABORT, NOCOUNT ON;

DECLARE @id NVARCHAR(18) = NULL;

EXEC dbo.GetGroupIdByStudent @record\_book, @id OUT;

IF(@id IS NOT NULL)

EXEC dbo.AttestationByIdGroup @id;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AttestationDelete]

CREATE PROC [dbo].[AttestationDelete]

@rc INT OUTPUT,

@id\_attestation INT

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF(EXISTS(SELECT id\_attestation FROM dbo.Attestation WHERE id\_attestation = @id\_attestation))

BEGIN

BEGIN TRAN;

DELETE

FROM dbo.Attestation

WHERE id\_attestation = @id\_attestation;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AttestationInsert]

CREATE PROC [dbo].[AttestationInsert]

@rc INT OUTPUT,

@id\_attestation INT,

@id\_group NVARCHAR(18),

@date\_begin DATE,

@date\_end DATE,

@stat BIT = 0

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF(EXISTS(SELECT id\_group FROM dbo.GroupStudent WHERE id\_group = @id\_group) AND

DATEDIFF(day,@date\_begin, @date\_end)>0)

BEGIN;

BEGIN TRAN;

INSERT INTO dbo.Attestation(id\_attestation,id\_group,date\_begin, date\_end, stat)

VALUES(@id\_attestation,@id\_group,@date\_begin,@date\_end,@stat);

COMMIT;

SET @rc = 1;

END;

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AttestationUpdate]

CREATE PROC [dbo].[AttestationUpdate]

@rc INT OUTPUT,

@id\_attestation INT,

@date\_begin DATE = NULL,

@date\_end DATE = NULL

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @d\_begin DATE = NULL;

DECLARE @d\_end DATE = NULL;

SELECT @d\_begin= date\_begin, @d\_end = date\_end FROM dbo.Attestation

WHERE id\_attestation = @id\_attestation;

IF( (@d\_begin IS NOT NULL) AND (@d\_end IS NOT NULL) )

BEGIN

IF((@date\_begin IS NOT NULL) AND DATEDIFF(day,@date\_begin, @d\_end)>0)

BEGIN

BEGIN TRAN;

UPDATE dbo.Attestation

SET date\_begin = @date\_begin

WHERE id\_attestation = @id\_attestation;

COMMIT;

SET @rc = 1;

END

IF((@date\_end IS NOT NULL) AND DATEDIFF(day,@d\_begin, @date\_end)>0)

BEGIN

BEGIN TRAN;

UPDATE dbo.Attestation

SET date\_end = @date\_end

WHERE id\_attestation = @id\_attestation;

COMMIT;

SET @rc = 1;

END

IF((@date\_end IS NOT NULL) AND DATEDIFF(day,@d\_begin, @date\_end)>0)

BEGIN

BEGIN TRAN;

UPDATE dbo.Attestation

SET date\_end = @date\_end

WHERE id\_attestation = @id\_attestation;

COMMIT;

SET @rc = 1;

END

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AttributeDelete]

CREATE PROC [dbo].[AttributeDelete]

@id INT

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

DELETE

FROM dbo.Attribute

WHERE id\_atr = @id

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AttributeInsert]

CREATE PROC [dbo].[AttributeInsert]

@name NVARCHAR(15)

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

INSERT INTO dbo.Attribute ( name )

VALUES( @name )

SELECT id\_atr, name

FROM dbo.Attribute

WHERE id\_atr = SCOPE\_IDENTITY()

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AttributeSelect]

CREATE PROC [dbo].[AttributeSelect]

@id INT = NULL

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

SELECT id\_atr, name

FROM dbo.Attribute

WHERE (id\_atr = @id OR @id IS NULL)

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[AttributeUpdate]

CREATE PROC [dbo].[AttributeUpdate]

@id INT,

@name NVARCHAR(15)

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

UPDATE dbo.Attribute

SET name=@name

WHERE id\_atr = @id

SELECT id\_atr, name

FROM dbo.Attribute

WHERE id\_atr = @id

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[Authorisation]

CREATE PROC [dbo].[Authorisation]

@login NVARCHAR(16),

@password NVARCHAR(20),

@rc INT OUTPUT

AS BEGIN

SET @rc = -1;

DECLARE @admin INT;

DECLARE @leader INT;

IF EXISTS(SELECT user\_login FROM dbo.Users WHERE user\_login like @login and user\_password like @password)

BEGIN

exec dbo.UserIsAdmin @login, @password, @admin out;

IF (@admin = 1)

SET @rc = 2;

ELSE

BEGIN

exec dbo.StudentIsLeader @login, @leader out;

IF (@leader=1)

SET @rc = 1;

ELSE

SET @rc = 0;

END

END

END

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[CertificateByStudent]

CREATE PROC [dbo].[CertificateByStudent]

@record\_book INT

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT c.number, s.first\_name +' '+ s.surname +' '+ ISNULL(s.patronymic,'') as name, c.school,

c.belarusian\_language, c.belarusian\_literature, c.russian\_language, c.russian\_literature, c.foreign\_language, l.full\_name as [language],

c.mathematics, c.informatics, c.history\_of\_belarus, c.world\_history, c.social\_science, c.geography\_subj, c.biology, c.physics, c.astronomy,

c.chemistry, c.physical\_culture, c.preliminary\_medical,c.gpa, c.gold\_medal, c.silver\_medal

FROM dbo.[Certificate] c inner join dbo.Student s on c.record\_book = s.record\_book

inner join dbo.Lenguage l on c.f\_language = l.short\_name

WHERE c.record\_book = @record\_book;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[CertificateDelete]

CREATE PROC [dbo].[CertificateDelete]

@rc INT OUTPUT,

@record\_book INT

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF(EXISTS(SELECT record\_book FROM dbo.[Certificate] WHERE record\_book = @record\_book))

BEGIN

BEGIN TRAN;

DELETE

dbo.[Certificate] WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[CertificateInsert]

CREATE PROC [dbo].[CertificateInsert]

@rc INT OUTPUT,

@number INT,

@record\_book INT,

@school NVARCHAR(50),

@gpa FLOAT,

@belarusian\_language INT,

@belarusian\_literature INT,

@russian\_language INT,

@russian\_literature INT,

@foreign\_language INT,

@language NVARCHAR(20), --справочник языков

@mathematics INT,

@informatics INT,

@history\_of\_belarus INT,

@world\_history INT,

@social\_science INT,

@geography\_subj INT,

@biology INT,

@physics INT,

@astronomy INT,

@chemistry INT,

@physical\_culture INT,

@preliminary\_medical INT,

@gold\_medal BIT = 0,

@silver\_medal BIT = 0

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @short\_leng NVARCHAR(20) = '';

SELECT @short\_leng = short\_name FROM dbo.Lenguage WHERE LOWER(full\_name) like LOWER(@language);

DECLARE @check\_gpa FLOAT = -1;

SET @check\_gpa = ROUND((@belarusian\_language + @belarusian\_literature + @russian\_language + @russian\_literature + @foreign\_language +

@mathematics + @informatics + @history\_of\_belarus + @world\_history + @social\_science + @geography\_subj + @biology +

@physics + @astronomy + @chemistry + @physical\_culture + @preliminary\_medical)/17.,1);

IF( EXISTS(SELECT record\_book FROM dbo.Student WHERE record\_book = @record\_book) AND

@short\_leng!='' AND @check\_gpa = @gpa AND

@gpa>=0.0 AND @gpa<=10.0 AND

@belarusian\_language >=0.0 AND @belarusian\_language <=10.0 AND

@belarusian\_literature>=0.0 AND @belarusian\_literature<=10.0 AND

@russian\_language >=0.0 AND @russian\_language <=10.0 AND

@russian\_literature >=0.0 AND @russian\_literature <=10.0 AND

@foreign\_language >=0.0 AND @foreign\_language <=10.0 AND

@mathematics >=0.0 AND @mathematics <=10.0 AND

@informatics >=0.0 AND @informatics <=10.0 AND

@history\_of\_belarus >=0.0 AND @history\_of\_belarus <=10.0 AND

@world\_history >=0.0 AND @world\_history <=10.0 AND

@social\_science >=0.0 AND @social\_science <=10.0 AND

@geography\_subj >=0.0 AND @geography\_subj <=10.0 AND

@biology >=0.0 AND @biology <=10.0 AND

@physics >=0.0 AND @physics <=10.0 AND

@astronomy >=0.0 AND @astronomy <=10.0 AND

@chemistry >=0.0 AND @chemistry <=10.0 AND

@physical\_culture >=0.0 AND @physical\_culture <=10.0 AND

@preliminary\_medical>=0.0 AND @preliminary\_medical <=10.0 )

BEGIN

BEGIN TRAN;

INSERT INTO dbo.[Certificate](number,record\_book,school,gpa,belarusian\_language,belarusian\_literature,russian\_language,

russian\_literature,foreign\_language,f\_language,mathematics,informatics,history\_of\_belarus,world\_history,

social\_science,geography\_subj,biology,physics,astronomy,chemistry,physical\_culture,preliminary\_medical,

gold\_medal,silver\_medal)

VALUES (@number,@record\_book,@school,@gpa,@belarusian\_language,@belarusian\_literature,@russian\_language,

@russian\_literature,@foreign\_language,@short\_leng,@mathematics,@informatics,@history\_of\_belarus,@world\_history,

@social\_science,@geography\_subj,@biology,@physics,@astronomy,@chemistry,@physical\_culture,@preliminary\_medical,

@gold\_medal,@silver\_medal);

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[CertificateUpdate]

CREATE PROC [dbo].[CertificateUpdate]

@rc INT OUTPUT,

@record\_book INT,

@school NVARCHAR(50) = '',

@belarusian\_language INT = -1,

@belarusian\_literature INT = -1,

@russian\_language INT = -1,

@russian\_literature INT = -1,

@foreign\_language INT = -1,

@language NVARCHAR(20) = '', --справочник языков

@mathematics INT = -1,

@informatics INT = -1,

@history\_of\_belarus INT = -1,

@world\_history INT = -1,

@social\_science INT = -1,

@geography\_subj INT = -1,

@biology INT = -1,

@physics INT= -1,

@astronomy INT = -1,

@chemistry INT = -1,

@physical\_culture INT = -1,

@preliminary\_medical INT = -1,

@gold\_medal BIT = NULL,

@silver\_medal BIT = NULL

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT record\_book FROM dbo.[Certificate] WHERE record\_book = @record\_book))

BEGIN

IF(@school !='')

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET school = @school WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@belarusian\_language >=0 AND @belarusian\_language<=10)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET belarusian\_language = @belarusian\_language WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@belarusian\_literature >=0 AND @belarusian\_literature<=10)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET belarusian\_literature = @belarusian\_literature WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@russian\_language >=0 AND @russian\_language<=10)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET russian\_language = @russian\_language WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@russian\_literature >=0 AND @russian\_literature<=10)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET russian\_literature = @russian\_literature WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@foreign\_language >=0 AND @foreign\_language <=10)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET foreign\_language = @foreign\_language WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

DECLARE @short\_leng NVARCHAR(20) = '';

SELECT @short\_leng = short\_name FROM dbo.Lenguage WHERE LOWER(full\_name) like LOWER(@language);

IF(@short\_leng !='')

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET f\_language = @short\_leng WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@mathematics >=0 AND @mathematics <=10)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET mathematics = @mathematics WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@informatics >=0 AND @informatics <=10)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET informatics = @informatics WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@history\_of\_belarus >=0 AND @history\_of\_belarus <=10)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET history\_of\_belarus = @history\_of\_belarus WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@world\_history>=0 AND @world\_history <=10)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET world\_history = @world\_history WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@social\_science>=0 AND @social\_science <=10)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET social\_science = @social\_science WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@geography\_subj>=0 AND @geography\_subj <=10)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET geography\_subj = @geography\_subj WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@biology>=0 AND @biology <=10)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET biology = @biology WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@physics>=0 AND @physics <=10)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET physics = @physics WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@astronomy>=0 AND @astronomy<=10)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET astronomy = @astronomy WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@chemistry>=0 AND @chemistry<=10)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET chemistry = @chemistry WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@physical\_culture>=0 AND @physical\_culture<=10)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET physical\_culture = @physical\_culture WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@preliminary\_medical >=0 AND @preliminary\_medical<=10)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET preliminary\_medical = @preliminary\_medical WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@gold\_medal IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET gold\_medal = @gold\_medal WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@silver\_medal IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.[Certificate] SET silver\_medal = @silver\_medal WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

DECLARE @gpa FLOAT;

SELECT @gpa = ROUND((belarusian\_language + belarusian\_literature + russian\_language + russian\_literature + foreign\_language +

mathematics + informatics + history\_of\_belarus + world\_history + social\_science + geography\_subj + biology +

physics + astronomy + chemistry + physical\_culture + preliminary\_medical)/17.,1)

FROM dbo.[Certificate] WHERE record\_book = @record\_book;

BEGIN TRAN;

UPDATE dbo.[Certificate] SET gpa = @gpa WHERE record\_book = @record\_book;

COMMIT;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[CurenNotAttestationByGroupNum]

CREATE PROC [dbo].[CurenNotAttestationByGroupNum]

@course INT,

@group INT

AS

SET XACT\_ABORT, NOCOUNT ON;

DECLARE @id NVARCHAR(18) = NULL;

EXEC dbo.GetGroupIdByCourseNum @course, @group, @id OUT;

IF(@id IS NOT NULL)

EXEC dbo.CurentNotAttestationByIdGroup @id;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[CurentAttestation]

CREATE PROC [dbo].[CurentAttestation]

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT a.id\_attestation, g.course, g.group\_number, date\_begin, date\_end

FROM dbo.Attestation a inner join dbo.GroupStudent g on a.id\_group= g.id\_group

WHERE ( a.stat = 0);

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[CurentNotAttestation]

CREATE PROC [dbo].[CurentNotAttestation]

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT g.course, g.group\_number,(s.surname + ',' + SPACE(1) + SUBSTRING(s.first\_name, 1, 1) + '.' + SPACE(1) + SUBSTRING( ISNULL(s.patronymic,''), 1, 1) + '.') AS name,

d.short\_name as discipline, ISNULL(m.name,'') as mark, t.short\_fio

FROM dbo.ListNotAttestation l inner join dbo.Student s on l.record\_book = s.record\_book

inner join dbo.Discipline d on l.discipline = d.id\_disc

inner join dbo.Teacher t on l.teacher = t.id\_teach

inner join dbo.GroupStudent g on s.id\_group = g.id\_group

left outer join dbo.Mark m on l.mark = m.id\_mark

WHERE l.date\_end<=GETDATE();

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[CurentNotAttestationByIdGroup]

CREATE PROC [dbo].[CurentNotAttestationByIdGroup]

@id\_group NVARCHAR(18)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT (s.surname + ',' + SPACE(1) + SUBSTRING(s.first\_name, 1, 1) + '.' + SPACE(1) + SUBSTRING( ISNULL(s.patronymic,''), 1, 1) + '.') AS name, d.short\_name as discipline,

ISNULL(m.name,'') as mark, t.short\_fio

FROM dbo.ListNotAttestation l inner join dbo.Student s on l.record\_book = s.record\_book

inner join dbo.Discipline d on l.discipline = d.id\_disc

inner join dbo.Teacher t on l.teacher = t.id\_teach

inner join dbo.GroupStudent g on s.id\_group = g.id\_group

left outer join dbo.Mark m on l.mark = m.id\_mark

WHERE LOWER(s.id\_group) like ('%'+LOWER(@id\_group)+'%') AND l.date\_end<=GETDATE();

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[DisciplineDelete]

CREATE PROC [dbo].[DisciplineDelete]

@id INT

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

DELETE

FROM dbo.Discipline

WHERE id\_disc = @id

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[DisciplineInsert]

CREATE PROC [dbo].[DisciplineInsert]

@short\_name NVARCHAR(20),

@full\_name NVARCHAR(50),

@link NVARCHAR(500)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

INSERT INTO dbo.Discipline ( short\_name, full\_name, link)

VALUES( @short\_name, @full\_name, @link );

COMMIT;

SELECT id\_disc, short\_name, full\_name, link

FROM dbo.Discipline

WHERE id\_disc= SCOPE\_IDENTITY();

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[DisciplineSelect]

CREATE PROC [dbo].[DisciplineSelect]

@short\_name NVARCHAR(20)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT id\_disc, short\_name, full\_name, link

FROM dbo.Discipline

WHERE (LOWER(short\_name) like('%'+ LOWER(@short\_name) + '%') OR @short\_name IS NULL);

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[DisciplineUpdate]

CREATE PROC [dbo].[DisciplineUpdate]

@id INT,

@short\_name NVARCHAR(20),

@full\_name NVARCHAR(50),

@link NVARCHAR(500)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

UPDATE dbo.Discipline

SET short\_name = @short\_name, full\_name = @full\_name, link = @link

WHERE id\_disc = @id;

COMMIT;

SELECT id\_disc, short\_name, full\_name, link

FROM dbo.Discipline

WHERE id\_disc = @id;

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[DistrictDelete]

CREATE PROC [dbo].[DistrictDelete]

@id INT

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

DELETE

FROM dbo.District

WHERE id\_distr = @id

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[DistrictInsert]

CREATE PROC [dbo].[DistrictInsert]

@name NVARCHAR(12)

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

INSERT INTO dbo.District ( name )

VALUES( @name )

SELECT id\_distr, name

FROM dbo.District

WHERE id\_distr = SCOPE\_IDENTITY()

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[DistrictSelect]

CREATE PROC [dbo].[DistrictSelect]

@id INT

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

SELECT id\_distr, name

FROM dbo.District

WHERE (id\_distr = @id OR @id IS NULL)

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[DistrictUpdate]

CREATE PROC [dbo].[DistrictUpdate]

@id INT,

@name NVARCHAR(12)

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

UPDATE dbo.District

SET name=@name

WHERE id\_distr = @id

SELECT id\_distr, name

FROM dbo.District

WHERE id\_distr = @id

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[EducationalProcessDelete]

CREATE PROC [dbo].[EducationalProcessDelete]

@rc INT OUTPUT,

@academic\_year CHAR(9),

@form NCHAR(2),

@course INT

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF(EXISTS(SELECT id FROM dbo.EducationalProcess WHERE academic\_year = @academic\_year AND course=@course AND form = @form))

BEGIN

BEGIN TRAN;

DELETE

FROM dbo.EducationalProcess

WHERE academic\_year = @academic\_year AND course=@course AND form = @form;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[EducationalProcessInsert]

CREATE PROC [dbo].[EducationalProcessInsert]

@rc INT OUTPUT,

@academic\_year CHAR(9),

@form NCHAR(2),

@course INT,

@date\_start DATE,

@date\_end DATE

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF(EXISTS(SELECT id\_form FROM dbo.FormProcess WHERE id\_form = @form) AND

DATEDIFF(day,@date\_start, @date\_end)>0 AND @course>0 AND @course<5)

BEGIN;

BEGIN TRAN;

INSERT INTO dbo.EducationalProcess (academic\_year, form, course, date\_start, date\_end)

VALUES( @academic\_year, @form, @course, @date\_start, @date\_end);

COMMIT;

SET @rc = 1;

END;

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[EducationalProcessSelectByCourse]

CREATE PROC [dbo].[EducationalProcessSelectByCourse]

@academic\_year CHAR(9),

@course INT

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT f.name, e.course, e.date\_start, e.date\_end

FROM dbo.EducationalProcess e inner join dbo.FormProcess f on e.form = f.id\_form

WHERE academic\_year = @academic\_year AND course=@course;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[EducationalProcessSelectByYear]

CREATE PROC [dbo].[EducationalProcessSelectByYear]

@academic\_year CHAR(9)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT DISTINCT f.name,

stuff(( SELECT ', ' + CAST(e2.course as CHAR(1)) FROM dbo.EducationalProcess e2 WHERE e1.form = e2.form AND e1.date\_start= e2.date\_start AND e1.date\_end=e2.date\_end FOR XML PATH('')),1,1,'') course,

e1.date\_start, e1.date\_end

FROM dbo.EducationalProcess e1 inner join dbo.FormProcess f on e1.form = f.id\_form WHERE academic\_year = @academic\_year

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[EducationalProcessUpdate]

CREATE PROC [dbo].[EducationalProcessUpdate]

@rc INT OUTPUT,

@academic\_year CHAR(9),

@form NCHAR(2),

@course INT,

@date\_start DATE = NULL,

@date\_end DATE = NULL

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @d\_start DATE = NULL;

DECLARE @d\_end DATE = NULL;

SELECT @d\_start= date\_start, @d\_end = date\_end FROM dbo.EducationalProcess

WHERE academic\_year = @academic\_year AND course=@course AND form = @form;

IF((@d\_start IS NOT NULL) AND (@d\_end IS NOT NULL) )

BEGIN

IF((@date\_start IS NOT NULL) AND DATEDIFF(day,@date\_start, @d\_end)>0)

BEGIN

BEGIN TRAN;

UPDATE dbo.EducationalProcess

SET date\_start = @date\_start

WHERE academic\_year = @academic\_year AND course=@course AND form = @form;

COMMIT;

SET @rc = 1;

END

IF((@date\_end IS NOT NULL) AND DATEDIFF(day,@d\_start, @date\_end)>0)

BEGIN

BEGIN TRAN;

UPDATE dbo.EducationalProcess

SET date\_end = @date\_end

WHERE academic\_year = @academic\_year AND course=@course AND form = @form;

COMMIT;

SET @rc = 1;

END

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[EnrollmentDelete]

CREATE PROC [dbo].[EnrollmentDelete]

@number VARCHAR(15)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

DELETE

FROM dbo.Enrollment

WHERE number = @number;

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[EnrollmentInsert]

CREATE PROC [dbo].[EnrollmentInsert]

@number VARCHAR(15),

@date\_sig DATE,

@signed NVARCHAR(15)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

INSERT INTO dbo.Enrollment (number, date\_sig, signed)

VALUES( @number, @date\_sig, @signed);

COMMIT;

SELECT number, date\_sig, signed

FROM dbo.Enrollment

WHERE number = @number;

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[EnrollmentSelect]

CREATE PROC [dbo].[EnrollmentSelect]

@number VARCHAR(15)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT number, date\_sig, signed

FROM dbo.Enrollment

WHERE (number = @number OR @number IS NULL);

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[EnrollmentUpdate]

CREATE PROC [dbo].[EnrollmentUpdate]

@number VARCHAR(15),

@numbernew VARCHAR(15),

@date\_sig DATE,

@signed NVARCHAR(15)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

UPDATE dbo.Enrollment

SET number = @numbernew, date\_sig = @date\_sig, signed = @signed

WHERE number = @number;

COMMIT;

SELECT number, date\_sig, signed

FROM dbo.Enrollment

WHERE number = @numbernew;

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ExamCreditByNumber]

CREATE PROC [dbo].[ExamCreditByNumber]

@number VARCHAR(10)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT e.number\_statement,a.name as[type], e.date\_sig, d.full\_name as discipline, g.course, g.group\_number,

sp.short\_name as speciality, t1.full\_fio as teacher\_1 , ISNULL(t2.full\_fio,'-') as teacher\_2, ISNULL(t3.full\_fio,'-') as teacher\_3

FROM dbo.ExamCredit e inner join dbo.Attribute a on e.attribute = a.id\_atr

inner join dbo.Discipline d on e.discipline = d.id\_disc

inner join dbo.GroupStudent g on e.id\_group = g.id\_group

inner join dbo.Speciality sp on g.kod\_special = sp.kod\_special

inner join dbo.Teacher t1 on e.teacher\_1 = t1.id\_teach

left outer join dbo.Teacher t2 on e.teacher\_2 = t2.id\_teach

left outer join dbo.Teacher t3 on e.teacher\_3 = t3.id\_teach

WHERE e.number\_statement like ('%'+ @number +'%');

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ExamCreditDelete]

CREATE PROC [dbo].[ExamCreditDelete]

@rc INT OUTPUT,

@number\_statement VARCHAR(10)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT number\_statement FROM dbo.ExamCredit WHERE number\_statement = @number\_statement))

BEGIN

BEGIN TRAN;

DELETE

FROM dbo.ExamCredit

WHERE number\_statement = @number\_statement;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ExamCreditInsert]

CREATE PROC [dbo].[ExamCreditInsert]

@rc INT OUTPUT,

@number\_statement VARCHAR(10),

@attribute\_name NVARCHAR(15),

@date\_sig DATE,

@discipline NVARCHAR(20),

@id\_group NVARCHAR(18),

@teacher\_1 INT,

@teacher\_2 INT = NULL,

@teacher\_3 INT = NULL

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @id\_atr INT = 0;

SELECT @id\_atr = id\_atr FROM dbo.Attribute WHERE LOWER(name) like LOWER(@attribute\_name)

DECLARE @id\_discipline INT = 0;

SELECT @id\_discipline = id\_disc FROM dbo.Discipline WHERE LOWER(short\_name) like LOWER(@discipline);

DECLARE @id\_g NVARCHAR(18) = '';

SELECT @id\_g = id\_group FROM dbo.GroupStudent WHERE LOWER(id\_group) like LOWER(@id\_group);

IF( EXISTS(SELECT id\_teach FROM dbo.Teacher WHERE id\_teach=@teacher\_1) AND

@id\_g!='' AND @id\_atr!=0 AND @id\_discipline!=0 AND

((@teacher\_2 IS NULL) OR EXISTS(SELECT id\_teach FROM dbo.Teacher WHERE id\_teach=@teacher\_2)) AND

((@teacher\_3 IS NULL) OR EXISTS(SELECT id\_teach FROM dbo.Teacher WHERE id\_teach=@teacher\_3)))

BEGIN

BEGIN TRAN;

INSERT INTO dbo.ExamCredit(number\_statement, attribute, date\_sig, discipline, id\_group, teacher\_1, teacher\_2, teacher\_3)

VALUES( @number\_statement, @id\_atr, @date\_sig, @id\_discipline, @id\_g, @teacher\_1, @teacher\_2, @teacher\_3);

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ExpelledDelete

CREATE PROC [dbo].[ExpelledDelete]

@number VARCHAR(15)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

DELETE

FROM dbo.Expelled

WHERE number = @number;

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ExpelledInsert]

CREATE PROC [dbo].[ExpelledInsert]

@number VARCHAR(15),

@date\_sig DATE,

@signed NVARCHAR(15)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

INSERT INTO dbo.Expelled (number, date\_sig, signed)

VALUES( @number, @date\_sig, @signed);

COMMIT;

SELECT number, date\_sig, signed

FROM dbo.Expelled

WHERE number = @number;

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ExpelledSelect]

CREATE PROC [dbo].[ExpelledSelect]

@number VARCHAR(15)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT number, date\_sig, signed

FROM dbo.Expelled

WHERE (number = @number OR @number IS NULL);

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ExpelledUpdate]

CREATE PROC [dbo].[ExpelledUpdate]

@number VARCHAR(15),

@numbernew VARCHAR(15),

@date\_sig DATE,

@signed NVARCHAR(15)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

UPDATE dbo.Expelled

SET number = @numbernew, date\_sig = @date\_sig, signed = @signed

WHERE number = @number;

COMMIT;

SELECT number, date\_sig, signed

FROM dbo.Expelled

WHERE number = @numbernew;

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ExtractFromRecordBook]

CREATE PROC [dbo].[ExtractFromRecordBook]

@record\_book INT

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT r.number\_statement, d.full\_name as discipline, a.name as [type], m.name as mark, e.date\_sig

FROM dbo.ResultExam r inner join dbo.ExamCredit e on r.number\_statement = e.number\_statement

inner join dbo.Student s on r.record\_book = s.record\_book

inner join dbo.Mark m on r.mark = m.id\_mark

inner join dbo.Discipline d on d.id\_disc = e.discipline

inner join dbo.Attribute a on e.attribute = a.id\_atr

WHERE r.record\_book=@record\_book AND m.type\_mark = 1 ORDER BY date\_sig;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[FormProcessDelete]

CREATE PROC [dbo].[FormProcessDelete]

@id NVARCHAR(2)

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

DELETE

FROM dbo.FormProcess

WHERE id\_form = @id

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[FormProcessInsert]

CREATE PROC [dbo].[FormProcessInsert]

@id NVARCHAR(2),

@name NVARCHAR(30)

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

INSERT INTO dbo.FormProcess ( id\_form, name)

VALUES(@id, @name )

SELECT id\_form, name

FROM dbo.FormProcess

WHERE id\_form= @id

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[FormProcessSelect]

CREATE PROC [dbo].[FormProcessSelect]

@id NVARCHAR(2)

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

SELECT id\_form, name

FROM dbo.FormProcess

WHERE (id\_form = @id OR @id IS NULL)

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[FormProcessUpdate]

CREATE PROC [dbo].[FormProcessUpdate]

@id NVARCHAR(2),

@id\_new NVARCHAR(2),

@name NVARCHAR(30)

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

UPDATE dbo.FormProcess

SET id\_form = @id\_new, name = @name

WHERE id\_form = @id

SELECT id\_form, name

FROM dbo.FormProcess

WHERE id\_form = @id\_new

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[GeneralAbsentsByGroup]

CREATE PROC [dbo].[GeneralAbsentsByGroup]

@course INT,

@group INT,

@date\_start DATE,

@date\_end DATE

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT s.surname + ' '+ s.first\_name as name,

Respectful = SUM(CASE WHEN reason= 1 THEN A.count\_hours ELSE 0 END),

NotRespectful = SUM(CASE WHEN reason= 0 THEN A.count\_hours ELSE 0 END)

FROM dbo.Absents a inner join dbo.Student s on a.record\_book = s.record\_book

inner join dbo.GroupStudent g on s.id\_group = g.id\_group

WHERE (a.date\_abs BETWEEN @date\_start AND @date\_end ) AND g.course = @course AND g.group\_number = @group

GROUP BY s.first\_name, s.surname;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[GeneralAbsentsByStudent]

CREATE PROC [dbo].[GeneralAbsentsByStudent]

@record\_book INT,

@date\_start DATE,

@date\_end DATE

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT s.surname + ' '+ s.first\_name as name,

Respectful = SUM(CASE WHEN reason= 1 THEN A.count\_hours ELSE 0 END),

NotRespectful = SUM(CASE WHEN reason= 0 THEN A.count\_hours ELSE 0 END)

FROM dbo.Absents a inner join dbo.Student s on a.record\_book = s.record\_book

WHERE (a.date\_abs BETWEEN @date\_start AND @date\_end ) AND s.record\_book=@record\_book

GROUP BY s.first\_name, s.surname;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[GetDisbandGroup]

CREATE PROC [dbo].[GetDisbandGroup]

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT id\_group, course, group\_number,s.short\_name, begin\_learning, end\_learning, disband

FROM dbo.GroupStudent g inner join dbo.Speciality s on g.kod\_special=s.kod\_special

WHERE g.disband=1;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[GetGroupByCourseNum]

CREATE PROC [dbo].[GetGroupByCourseNum]

@course INT,

@number INT

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT id\_group, course, group\_number,s.short\_name, begin\_learning, end\_learning, disband

FROM dbo.GroupStudent g inner join dbo.Speciality s on g.kod\_special=s.kod\_special

WHERE g.course =@course AND g.group\_number = @number;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[GetGroupById]

CREATE PROC [dbo].[GetGroupById]

@id NVARCHAR(18)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT id\_group, course, group\_number,s.short\_name, begin\_learning, end\_learning, disband

FROM dbo.GroupStudent g inner join dbo.Speciality s on g.kod\_special=s.kod\_special

WHERE LOWER(g.id\_group) like('%'+ LOWER(@id) + '%') ;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[GetGroupIdByCourseNum]

CREATE PROC [dbo].[GetGroupIdByCourseNum]

@course INT,

@number INT,

@id NVARCHAR(18) OUTPUT

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT @id = id\_group

FROM dbo.GroupStudent

WHERE course = @course AND group\_number = @number;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[GetGroupIdByStudent]

CREATE PROC [dbo].[GetGroupIdByStudent]

@record\_book INT,

@id NVARCHAR(18) OUTPUT

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT @id = id\_group

FROM dbo.Student

WHERE record\_book = @record\_book;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[GetNotDisbandGroup]

CREATE PROC [dbo].[GetNotDisbandGroup]

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT id\_group, course, group\_number,s.short\_name, begin\_learning, end\_learning, disband

FROM dbo.GroupStudent g inner join dbo.Speciality s on g.kod\_special=s.kod\_special

WHERE g.disband=0;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[GroupCourseUpdate]

CREATE PROC [dbo].[GroupCourseUpdate]

@id NVARCHAR(18),

@course INT

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

IF(@course>0 AND @course<5)

BEGIN

BEGIN TRAN;

UPDATE dbo.GroupStudent SET course = @course WHERE id\_group= @id;

COMMIT;

END

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[GroupDelete]

CREATE PROC [dbo].[GroupDelete]

@id NVARCHAR(18)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

DELETE

FROM dbo.GroupStudent

WHERE id\_group= @id;

COMMIT;

END TRY

BEGIN CATCH

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[GroupDisbandUpdate]

CREATE PROC [dbo].[GroupDisbandUpdate]

@rc INT OUTPUT,

@id NVARCHAR(18),

@disband BIT

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

UPDATE dbo.GroupStudent SET disband = @disband WHERE id\_group= @id;

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[GroupInsert]

CREATE PROC [dbo].[GroupInsert]

@rc INT OUTPUT,

@id NVARCHAR(18),

@course INT = 1,

@group\_number INT,

@special NVARCHAR(10),

@begin\_learning DATE ,

@end\_learning DATE,

@disband BIT = 0

AS

BEGIN TRY

SET @rc = 0;

SET XACT\_ABORT, NOCOUNT ON;

IF EXISTS(SELECT kod\_special FROM dbo.Speciality WHERE LOWER(short\_name) like LOWER(@special))

BEGIN

DECLARE @kod NVARCHAR(13);

SELECT @kod=kod\_special FROM dbo.Speciality WHERE LOWER(short\_name) like LOWER(@special);

BEGIN TRAN;

INSERT INTO dbo.GroupStudent(id\_group,course,group\_number, kod\_special, begin\_learning, end\_learning, disband)

VALUES(@id,@course,@group\_number, @kod, @begin\_learning, @end\_learning, @disband);

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[InsertValuesAddresses]

CREATE procedure [dbo].[InsertValuesAddresses]

as begin

declare @i int = 0

while @i < 100000

begin

declare @surname nvarchar(5) exec RandString @surname out

declare @first\_name nvarchar(5) exec RandString @first\_name out

declare @patronymic nvarchar(5) exec RandString @patronymic out

INSERT INTO Deanery.dbo.Student(record\_book,surname, first\_name,patronymic, sex, date\_birth, id\_group, form)

VALUES (@i, @surname, @first\_name, @patronymic, 'м', '1998-12-15', '2016/2020ПОИТ5','п');

set @i = @i +1

end

end

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[LenguageDelete]

CREATE PROC [dbo].[LenguageDelete]

@short\_name NVARCHAR(4)

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

DELETE

FROM dbo.Lenguage

WHERE short\_name = @short\_name

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[LenguageInsert]

CREATE PROC [dbo].[LenguageInsert]

@short\_name NVARCHAR(4),

@full\_name NVARCHAR(20)

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

INSERT INTO dbo.Lenguage ( short\_name, full\_name)

VALUES(@short\_name, @full\_name )

SELECT short\_name, full\_name

FROM dbo.Lenguage

WHERE short\_name = @short\_name

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[LenguageSelect]

CREATE PROC [dbo].[LenguageSelect]

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

SELECT short\_name, full\_name

FROM dbo.Lenguage

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[LenguageUpdate]

CREATE PROC [dbo].[LenguageUpdate]

@short\_name NVARCHAR(4),

@short\_name\_new NVARCHAR(4),

@full\_name NVARCHAR(20)

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

UPDATE dbo.Lenguage

SET short\_name = @short\_name\_new, full\_name = @full\_name

WHERE short\_name = @short\_name

SELECT short\_name, full\_name

FROM dbo.Lenguage

WHERE short\_name = @short\_name\_new

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ListEnrollmentByNumber]

CREATE PROC [dbo].[ListEnrollmentByNumber]

@number VARCHAR(15)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT s.first\_name +' '+ s.surname +' '+ ISNULL(s.patronymic,'') as name, g.course, g.group\_number, sp.short\_name

FROM dbo.ListEnrollment l inner join dbo.Student s on l.record\_book=s.record\_book

inner join dbo.GroupStudent g on l.id\_group = g.id\_group

inner join dbo.Speciality sp on g.kod\_special = sp.kod\_special

WHERE l.number\_enrol like @number;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ListEnrollmentByStudent]

CREATE PROC [dbo].[ListEnrollmentByStudent]

@record\_book INT

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT s.first\_name +' '+ s.surname +' '+ ISNULL(s.patronymic,'') as name, g.course, g.group\_number, sp.short\_name as speciality

FROM dbo.ListEnrollment l inner join dbo.Student s on l.record\_book=s.record\_book

inner join dbo.GroupStudent g on l.id\_group = g.id\_group

inner join dbo.Speciality sp on g.kod\_special = sp.kod\_special

WHERE l.record\_book = @record\_book;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ListEnrollmentDelete]

CREATE PROC [dbo].[ListEnrollmentDelete]

@rc INT OUTPUT,

@number VARCHAR(15),

@record\_book INT,

@id\_group NVARCHAR(18)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT number\_enrol FROM dbo.ListEnrollment WHERE number\_enrol = @number AND record\_book = @record\_book AND id\_group = @id\_group))

BEGIN

BEGIN TRAN;

DELETE

FROM dbo.ListEnrollment

WHERE number\_enrol = @number AND record\_book = @record\_book AND id\_group = @id\_group;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ListEnrollmentInsert]

CREATE PROC [dbo].[ListEnrollmentInsert]

@rc INT OUTPUT,

@number VARCHAR(15),

@record\_book INT,

@id\_group NVARCHAR(18)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT record\_book FROM dbo.Student WHERE record\_book = @record\_book) AND

EXISTS(SELECT id\_group FROM dbo.GroupStudent WHERE id\_group = @id\_group) AND

EXISTS(SELECT number FROM dbo.Enrollment WHERE number = @number))

BEGIN

BEGIN TRAN;

INSERT INTO dbo.ListEnrollment (number\_enrol, record\_book, id\_group)

VALUES( @number, @record\_book, @id\_group);

SET @rc = 1;

UPDATE dbo.Student SET id\_group = @id\_group WHERE record\_book = @record\_book;

IF( (SELECT dismissed FROM dbo.Student WHERE record\_book = @record\_book) = 1)

UPDATE dbo.Student SET restored = 1, dismissed = 0 WHERE record\_book = @record\_book;

COMMIT;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ListExpelledByNumber]

CREATE PROC [dbo].[ListExpelledByNumber]

@number VARCHAR(15)

AS

SELECT s.first\_name +' '+ s.surname +' '+ ISNULL(s.patronymic,'') as name, g.course, g.group\_number, sp.short\_name

FROM dbo.ListExpelled l inner join dbo.Student s on l.record\_book=s.record\_book

inner join dbo.GroupStudent g on l.id\_group = g.id\_group

inner join dbo.Speciality sp on g.kod\_special = sp.kod\_special

WHERE l.number\_exp like @number;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ListExpelledByStudent]

CREATE PROC [dbo].[ListExpelledByStudent]

@record\_book INT

AS

SELECT s.first\_name +' '+ s.surname +' '+ ISNULL(s.patronymic,'') as name, g.course, g.group\_number, sp.short\_name as speciality

FROM dbo.ListExpelled l inner join dbo.Student s on l.record\_book=s.record\_book

inner join dbo.GroupStudent g on l.id\_group = g.id\_group

inner join dbo.Speciality sp on g.kod\_special = sp.kod\_special

WHERE l.record\_book = @record\_book;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ListExpelledDelete]

CREATE PROC [dbo].[ListExpelledDelete]

@rc INT OUTPUT,

@number VARCHAR(15),

@record\_book INT,

@id\_group NVARCHAR(18)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT number\_exp FROM dbo.ListExpelled WHERE number\_exp = @number AND record\_book = @record\_book AND id\_group = @id\_group))

BEGIN

BEGIN TRAN;

DELETE

FROM dbo.ListExpelled

WHERE number\_exp = @number AND record\_book = @record\_book AND id\_group = @id\_group;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ListExpelledInsert]

CREATE PROC [dbo].[ListExpelledInsert]

@rc INT OUTPUT,

@number VARCHAR(15),

@record\_book INT,

@id\_group NVARCHAR(18)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT record\_book FROM dbo.Student WHERE record\_book = @record\_book) AND

EXISTS(SELECT id\_group FROM dbo.GroupStudent WHERE id\_group = @id\_group) AND

EXISTS(SELECT number FROM dbo.Expelled WHERE number = @number))

BEGIN

BEGIN TRAN;

INSERT INTO dbo.ListExpelled (number\_exp, record\_book, id\_group)

VALUES( @number, @record\_book, @id\_group);

SET @rc = 1;

IF( (SELECT dismissed FROM dbo.Student WHERE record\_book = @record\_book) = 0)

UPDATE dbo.Student SET dismissed = 1 WHERE record\_book = @record\_book;

COMMIT;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[MarkDelete]

CREATE PROC [dbo].[MarkDelete]

@id INT

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

DELETE

FROM dbo.Mark

WHERE id\_mark = @id;

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[MarkInsert]

CREATE PROC [dbo].[MarkInsert]

@name NVARCHAR(15),

@type\_mark BIT

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

INSERT INTO dbo.Mark ( name, type\_mark)

VALUES( @name, @type\_mark)

COMMIT;

SELECT id\_mark, name, type\_mark

FROM dbo.Mark

WHERE id\_mark = SCOPE\_IDENTITY();

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[MarkSelect

CREATE PROC [dbo].[MarkSelect]

@id INT

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT id\_mark, name, type\_mark

FROM dbo.Mark

WHERE (id\_mark = @id OR @id IS NULL);

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[MarkUpdate]

CREATE PROC [dbo].[MarkUpdate]

@id INT,

@name NVARCHAR(15),

@type\_mark BIT

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

UPDATE dbo.Mark

SET name=@name, type\_mark=@type\_mark

WHERE id\_mark = @id;

COMMIT;

SELECT id\_mark, name, type\_mark

FROM dbo.Mark

WHERE id\_mark = @id;

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[MessengerByCourseGroup]

CREATE PROC [dbo].[MessengerByCourseGroup]

@course INT,

@group INT

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT surname +' '+ first\_name+' '+ ISNULL(patronymic,'') as name , m.mess

FROM dbo.Messenger m inner join dbo.Student s on m.record\_book = s.record\_book

inner join dbo.GroupStudent g on s.id\_group = g.id\_group

WHERE g.course = @course AND g.group\_number=@group;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[MessengerByStudent]

CREATE PROC [dbo].[MessengerByStudent]

@record\_book INT

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT surname +' '+ first\_name+' '+ ISNULL(patronymic,'') as name , m.mess

FROM dbo.Messenger m inner join dbo.Student s on m.record\_book = s.record\_book

WHERE m.record\_book = @record\_book;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[MessengerDelete]

CREATE PROC [dbo].[MessengerDelete]

@rc INT OUTPUT,

@record\_book INT,

@mess NVARCHAR(100) = NULL

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT record\_book FROM dbo.Messenger WHERE record\_book = @record\_book))

BEGIN

BEGIN TRAN;

DELETE

FROM dbo.Messenger

WHERE (record\_book = @record\_book AND mess=@mess) OR(record\_book = @record\_book AND @mess IS NULL) ;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[MessengerGroupByStudent]

CREATE PROC [dbo].[MessengerGroupByStudent]

@record\_book INT

AS

SET XACT\_ABORT, NOCOUNT ON;

DECLARE @id\_group NVARCHAR(13);

exec dbo.GetGroupIdByStudent @record\_book, @id\_group OUT;

SELECT s.surname +' '+ s.first\_name+' '+ ISNULL(s.patronymic,'') as fio,

stuff(( SELECT ', ' + mess FROM Messenger m WHERE m.record\_book=s.record\_book FOR XML PATH('')),1,1,'') messenger

FROM Student s WHERE s.id\_group = @id\_group ORDER BY fio

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[MessengerInsert]

CREATE PROC [dbo].[MessengerInsert]

@rc INT OUTPUT,

@record\_book INT,

@mess NVARCHAR(100)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT record\_book FROM dbo.Student WHERE record\_book = @record\_book))

BEGIN

BEGIN TRAN;

INSERT INTO dbo.Messenger(record\_book, mess)

VALUES( @record\_book, @mess);

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[NotAttestationById]

CREATE PROC [dbo].[NotAttestationById]

@id\_attestation INT

AS

SELECT (s.surname + ',' + SPACE(1) + SUBSTRING(s.first\_name, 1, 1) + '.' + SPACE(1) + SUBSTRING( ISNULL(s.patronymic,''), 1, 1) + '.') AS name,

d.short\_name as discipline, ISNULL(m.name,'') as mark, t.short\_fio

FROM dbo.ListNotAttestation l inner join dbo.Student s on l.record\_book = s.record\_book

inner join dbo.Discipline d on l.discipline = d.id\_disc

inner join dbo.Teacher t on l.teacher = t.id\_teach

left outer join dbo.Mark m on l.mark = m.id\_mark

WHERE l.id\_attestation = @id\_attestation;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[NotAttestationByIdGroup]

CREATE PROC [dbo].[NotAttestationByIdGroup]

@id\_group NVARCHAR(15)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT s.first\_name, s.surname, ISNULL(s.patronymic,'') as patronymic, d.short\_name as discipline,

ISNULL(m.name,'') as mark, t.short\_fio

FROM dbo.ListNotAttestation l inner join dbo.Student s on l.record\_book = s.record\_book

inner join dbo.Discipline d on l.discipline = d.id\_disc

inner join dbo.Teacher t on l.teacher = t.id\_teach

inner join dbo.GroupStudent g on s.id\_group = g.id\_group

left outer join dbo.Mark m on l.mark = m.id\_mark

WHERE LOWER(s.id\_group) like ('%'+LOWER(@id\_group)+'%');

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[NotAttestationDelete]

CREATE PROC [dbo].[NotAttestationDelete]

@rc INT OUTPUT,

@id\_attestation INT,

@record\_book INT,

@discipline NVARCHAR(20)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @id\_disc INT = -1;

SELECT @id\_disc = id\_disc FROM dbo.Discipline WHERE LOWER(short\_name) like LOWER(@discipline);

IF(@id\_disc !=-1 AND EXISTS(SELECT id\_attestation FROM dbo.ListNotAttestation WHERE id\_attestation = @id\_attestation AND record\_book = @record\_book AND discipline = @id\_disc))

BEGIN

BEGIN TRAN;

DELETE

FROM dbo.ListNotAttestation

WHERE id\_attestation = @id\_attestation AND record\_book = @record\_book AND discipline = @id\_disc;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[NotAttestationUpdateMark

CREATE PROC [dbo].[NotAttestationUpdateMark]

@rc INT OUTPUT,

@id\_attestation INT,

@record\_book INT,

@discipline NVARCHAR(20),

@mark NVARCHAR(15)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @id\_disc INT = -1;

SELECT @id\_disc = id\_disc FROM dbo.Discipline WHERE LOWER(short\_name) like LOWER(@discipline);

DECLARE @id\_mark INT = -1;

SELECT @id\_mark = id\_mark FROM dbo.Mark WHERE LOWER(name) like LOWER(@mark);

IF(@id\_mark !=-1 AND @id\_disc !=-1 AND EXISTS(SELECT id\_attestation FROM dbo.ListNotAttestation WHERE id\_attestation = @id\_attestation AND record\_book = @record\_book AND discipline = @id\_disc))

BEGIN

BEGIN TRAN;

UPDATE dbo.ListNotAttestation

SET mark = @id\_mark

WHERE id\_attestation = @id\_attestation AND record\_book = @record\_book AND discipline = @id\_disc;

COMMIT;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[NotFixAbsentsByIdGroup]

CREATE PROC [dbo].[NotFixAbsentsByIdGroup]

@id\_group NVARCHAR(18),

@date\_start DATE,

@date\_end DATE

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT s.surname + ' '+ s.first\_name as name, d.short\_name as discipline , a.date\_abs, count\_hours, reason

FROM dbo.Absents a inner join dbo.Student s on a.record\_book = s.record\_book

inner join dbo.Discipline d on a.discipline = d.id\_disc

inner join dbo.GroupStudent g on s.id\_group = g.id\_group

WHERE (a.date\_abs BETWEEN @date\_start AND @date\_end ) AND g.id\_group=@id\_group

GROUP BY a.record\_book, s.first\_name, s.surname, a.work\_out, count\_hours, reason, d.short\_name , a.form, a.date\_abs

HAVING a.work\_out=0;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[NotFixAttestationById]

CREATE PROC [dbo].[NotFixAttestationById]

@id\_attestation INT

AS

SELECT (s.surname + ',' + SPACE(1) + SUBSTRING(s.first\_name, 1, 1) + '.' + SPACE(1) + SUBSTRING( ISNULL(s.patronymic,''), 1, 1) + '.') AS name,

d.short\_name as discipline, ISNULL(m.name,'') as mark, t.short\_fio

FROM dbo.ListNotAttestation l inner join dbo.Student s on l.record\_book = s.record\_book

inner join dbo.Discipline d on l.discipline = d.id\_disc

inner join dbo.Teacher t on l.teacher = t.id\_teach

left outer join dbo.Mark m on l.mark = m.id\_mark

WHERE l.id\_attestation = @id\_attestation AND l.mark IS NULL;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ParentsByStudent]

CREATE PROC [dbo].[ParentsByStudent]

@record\_book INT

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT fio, sex, date\_birth, address\_perent, messenger, name as profession, workplace, post

FROM dbo.Parents p inner join dbo.Profession pr on p.profession = pr.id\_prof

WHERE p.record\_book = @record\_book;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ParentsDelete]

CREATE PROC [dbo].[ParentsDelete]

@rc INT OUTPUT,

@record\_book INT,

@sex NCHAR(1)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT record\_book FROM dbo.Parents WHERE record\_book = @record\_book AND LOWER(sex) like LOWER(@sex)))

BEGIN

BEGIN TRAN;

DELETE

dbo.Parents WHERE record\_book = @record\_book AND LOWER(sex) like LOWER(@sex) ;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ParentsInsert]

CREATE PROC [dbo].[ParentsInsert]

@rc INT OUTPUT,

@record\_book INT,

@fio NVARCHAR(50),

@sex NCHAR(1) = 'Ж',

@date\_birth DATE,

@address\_perent NVARCHAR(150),

@messenger NVARCHAR(50),

@profession INT,

@workplace NVARCHAR(150),

@post NVARCHAR(30)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT record\_book FROM dbo.Student WHERE record\_book = @record\_book) AND

EXISTS(SELECT id\_prof FROM dbo.Profession WHERE id\_prof = @profession) AND

(@sex in( 'М' ,'м','Ж','ж')))

BEGIN

BEGIN TRAN;

INSERT INTO dbo.Parents(record\_book, fio, sex, date\_birth, address\_perent, messenger,profession,workplace,post)

VALUES (@record\_book, @fio, LOWER(@sex), @date\_birth, @address\_perent, @messenger, @profession, @workplace, @post);

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ParentsUpdate]

CREATE PROC [dbo].[ParentsUpdate]

@rc INT OUTPUT,

@record\_book INT,

@sex NCHAR(1),

@fio NVARCHAR(20) = NULL,

@date\_birth DATE = NULL,

@address\_perent NVARCHAR(150) = NULL,

@messenger NVARCHAR(50) = NULL,

@profession INT = NULL,

@workplace NVARCHAR(150) = NULL,

@post NVARCHAR(30) = NULL

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT record\_book FROM dbo.Parents WHERE record\_book = @record\_book AND LOWER(sex) like LOWER(@sex)))

BEGIN

IF(@fio IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.Parents SET fio = @fio WHERE record\_book = @record\_book AND LOWER(sex) like LOWER(@sex);

COMMIT;

SET @rc = 1;

END

IF(@date\_birth IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.Parents SET date\_birth = @date\_birth WHERE record\_book = @record\_book AND LOWER(sex) like LOWER(@sex);

COMMIT;

SET @rc = 1;

END

IF(@address\_perent IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.Parents SET address\_perent = @address\_perent WHERE record\_book = @record\_book AND LOWER(sex) like LOWER(@sex);

COMMIT;

SET @rc = 1;

END

IF(@messenger IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.Parents SET messenger = @messenger WHERE record\_book = @record\_book AND LOWER(sex) like LOWER(@sex);

COMMIT;

SET @rc = 1;

END

IF(@profession IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.Parents SET profession = @profession WHERE record\_book = @record\_book AND LOWER(sex) like LOWER(@sex);

COMMIT;

SET @rc = 1;

END

IF(@workplace IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.Parents SET workplace = @workplace WHERE record\_book = @record\_book AND LOWER(sex) like LOWER(@sex);

COMMIT;

SET @rc = 1;

END

IF(@post IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.Parents SET post = @post WHERE record\_book = @record\_book AND LOWER(sex) like LOWER(@sex);

COMMIT;

SET @rc = 1;

END

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[PlanDisciplineByDiscipline]

CREATE PROC [dbo].[PlanDisciplineByDiscipline]

@discipline NVARCHAR(20)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT s.short\_name as speciality, p.semester, p.total, p.lecture, p.lab, p.practice, p.seminar, a.name as form

FROM dbo.PlanDiscipline p inner join dbo.Discipline d on p.discipline = d.id\_disc

inner join dbo.Speciality s on p.speciality = s.kod\_special

inner join dbo.Attribute a on p.form = a.id\_atr

WHERE LOWER(d.short\_name) like LOWER(@discipline);

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[PlanDisciplineBySemester]

CREATE PROC [dbo].[PlanDisciplineBySemester]

@semester INT,

@speciality NVARCHAR(10) = NULL

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT s.short\_name as speciality,d.short\_name as discipline, p.semester, p.total, p.lecture, p.lab, p.practice, p.seminar, a.name as form

FROM dbo.PlanDiscipline p inner join dbo.Discipline d on p.discipline = d.id\_disc

inner join dbo.Speciality s on p.speciality = s.kod\_special

inner join dbo.Attribute a on p.form = a.id\_atr

WHERE p.semester = @semester AND (LOWER(s.short\_name) like LOWER(@speciality) OR @speciality IS NULL);

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[PlanDisciplineBySpeciality]

CREATE PROC [dbo].[PlanDisciplineBySpeciality]

@speciality NVARCHAR(10)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT d.short\_name as discipline, p.semester, p.total, p.lecture, p.lab, p.practice, p.seminar, a.name as form

FROM dbo.PlanDiscipline p inner join dbo.Discipline d on p.discipline = d.id\_disc

inner join dbo.Speciality s on p.speciality = s.kod\_special

inner join dbo.Attribute a on p.form = a.id\_atr

WHERE LOWER(s.short\_name) like LOWER(@speciality);

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[PlanDisciplineDelete]

CREATE PROC [dbo].[PlanDisciplineDelete]

@rc INT OUTPUT,

@discipline NVARCHAR(20),

@speciality NVARCHAR(10),

@semester INT

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @id\_disc INT = -1;

SELECT @id\_disc = id\_disc FROM dbo.Discipline WHERE LOWER(short\_name) like LOWER(@discipline);

DECLARE @id\_spec INT = -1;

SELECT @id\_spec = kod\_special FROM dbo.Speciality WHERE LOWER(short\_name) like LOWER(@speciality);

IF( @id\_disc!=-1 AND @id\_spec!=-1 AND

EXISTS(SELECT discipline FROM dbo.PlanDiscipline WHERE discipline=@id\_disc AND speciality= @id\_spec AND semester=@semester))

BEGIN

BEGIN TRAN;

DELETE

FROM dbo.PlanDiscipline

WHERE discipline=@id\_disc AND speciality= @id\_spec AND semester=@semester;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[PlanDisciplineInsert]

CREATE PROC [dbo].[PlanDisciplineInsert]

@rc INT OUTPUT,

@discipline NVARCHAR(20),

@speciality NVARCHAR(10),

@semester INT ,

@total INT ,

@lecture INT = 0,

@lab INT = 0,

@practice INT = 0,

@seminar INT =0,

@form INT

AS

BEGIN TRY

SET @rc = 0;

SET XACT\_ABORT, NOCOUNT ON;

DECLARE @id\_disc INT = -1;

SELECT @id\_disc = id\_disc FROM dbo.Discipline WHERE LOWER(short\_name) like LOWER(@discipline);

DECLARE @id\_spec INT = -1;

SELECT @id\_spec = kod\_special FROM dbo.Speciality WHERE LOWER(short\_name) like LOWER(@speciality);

IF (@id\_disc!=-1 AND @id\_spec!=-1 AND @semester>0 AND @semester<11 AND

@lecture>=0 AND @lab>=0 AND @practice>=0 AND @seminar>=0 AND @total>0 AND

@total = @lecture+@lab+@practice+@seminar AND

EXISTS(SELECT id\_atr FROM dbo.Attribute WHERE id\_atr = @form))

BEGIN

BEGIN TRAN;

INSERT INTO dbo.PlanDiscipline(discipline, speciality, semester, total, lecture, lab, practice, seminar, form)

VALUES(@id\_disc, @id\_spec, @semester, @total, @lecture, @lab, @practice, @seminar, @form);

SET @rc = 1;

COMMIT;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[PlanDisciplineUpdate]

CREATE PROC [dbo].[PlanDisciplineUpdate]

@rc BIT OUTPUT,

@discipline NVARCHAR(20),

@speciality NVARCHAR(10),

@semester INT ,

@new\_semester INT =NULL,

@lecture INT =NULL,

@lab INT =NULL,

@practice INT =NULL,

@seminar INT =NULL,

@form INT =NULL

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @id\_disc INT = -1;

SELECT @id\_disc = id\_disc FROM dbo.Discipline WHERE LOWER(short\_name) like LOWER(@discipline);

DECLARE @id\_spec INT = -1;

SELECT @id\_spec = kod\_special FROM dbo.Speciality WHERE LOWER(short\_name) like LOWER(@speciality);

IF( @id\_disc!=-1 AND @id\_spec!=-1 AND

EXISTS(SELECT discipline FROM dbo.PlanDiscipline WHERE discipline=@id\_disc AND speciality= @id\_spec AND semester=@semester))

BEGIN

IF(@lecture IS NOT NULL AND @lecture>=0)

BEGIN

BEGIN TRAN;

UPDATE dbo.PlanDiscipline SET lecture=@lecture WHERE discipline=@id\_disc AND speciality= @id\_spec AND semester=@semester ;

COMMIT;

SET @rc = 1;

END

IF(@lab IS NOT NULL AND @lab>=0)

BEGIN

BEGIN TRAN;

UPDATE dbo.PlanDiscipline SET lab=@lab WHERE discipline=@id\_disc AND speciality= @id\_spec AND semester=@semester ;

COMMIT;

SET @rc = 1;

END

IF(@practice IS NOT NULL AND @practice>=0)

BEGIN

BEGIN TRAN;

UPDATE dbo.PlanDiscipline SET practice=@practice WHERE discipline=@id\_disc AND speciality= @id\_spec AND semester=@semester ;

COMMIT;

SET @rc = 1;

END

IF(@seminar IS NOT NULL AND @seminar>=0)

BEGIN

BEGIN TRAN;

UPDATE dbo.PlanDiscipline SET seminar=@seminar WHERE discipline=@id\_disc AND speciality= @id\_spec AND semester=@semester ;

COMMIT;

SET @rc = 1;

END

IF @rc = 1

BEGIN

DECLARE @total INT =-1;

SELECT @lecture=lecture, @lab=lab, @practice=practice, @seminar=seminar FROM dbo.PlanDiscipline WHERE discipline=@id\_disc AND speciality= @id\_spec AND semester=@semester

SET @total = @lecture+@lab+@practice+@seminar;

BEGIN TRAN;

UPDATE dbo.PlanDiscipline SET seminar=@seminar WHERE discipline=@id\_disc AND speciality= @id\_spec AND semester=@semester ;

COMMIT;

END

IF(@semester IS NOT NULL AND @semester>0 AND @semester<11 )

BEGIN

BEGIN TRAN;

UPDATE dbo.PlanDiscipline SET semester=@new\_semester WHERE discipline=@id\_disc AND speciality= @id\_spec AND semester=@semester ;

COMMIT;

SET @rc = 1;

END

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ProfessionDelete]

CREATE PROC [dbo].[ProfessionDelete]

@id INT

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

DELETE

FROM dbo.Profession

WHERE id\_prof = @id;

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ProfessionInsert]

CREATE PROC [dbo].[ProfessionInsert]

@name NVARCHAR(30)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

INSERT INTO dbo.Profession ( name )

VALUES( @name );

COMMIT;

SELECT id\_prof, name

FROM dbo.Profession

WHERE id\_prof= SCOPE\_IDENTITY();

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ProfessionSelect

CREATE PROC [dbo].[ProfessionSelect]

@name NVARCHAR(30)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT id\_prof, name

FROM dbo.Profession

WHERE (LOWER(name) like('%'+ LOWER(@name) + '%') OR @name IS NULL);

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ProfessionUpdate]

CREATE PROC [dbo].[ProfessionUpdate]

@id INT,

@name NVARCHAR(30)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

UPDATE dbo.Profession

SET name=@name

WHERE id\_prof = @id;

COMMIT;

SELECT id\_prof, name

FROM dbo.Profession

WHERE id\_prof = @id;

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[PulpitDelete]

CREATE PROC [dbo].[PulpitDelete]

@short\_name NVARCHAR(10)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

DELETE

FROM dbo.Pulpit

WHERE short\_name = @short\_name;

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[PulpitInsert]

CREATE PROC [dbo].[PulpitInsert]

@short\_name NVARCHAR(10),

@full\_name NVARCHAR(50),

@link NVARCHAR(500)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

INSERT INTO dbo.Pulpit ( short\_name, full\_name, link)

VALUES(@short\_name, @full\_name, @link );

COMMIT;

SELECT short\_name, full\_name, link

FROM dbo.Pulpit

WHERE short\_name = @short\_name;

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[PulpitSelect] Script Date: 22.12.2018 9:56:15 \*\*\*\*\*\*/

CREATE PROC [dbo].[PulpitSelect]

@short\_name NVARCHAR(10)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT short\_name, full\_name, link

FROM dbo.Pulpit

WHERE (LOWER(short\_name) like('%'+ LOWER(@short\_name) + '%') OR @short\_name IS NULL);

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[PulpitUpdate] Script Date: 22.12.2018 9:56:15 \*\*\*\*\*\*/

CREATE PROC [dbo].[PulpitUpdate]

@short\_name NVARCHAR(10),

@short\_name\_new NVARCHAR(10),

@full\_name NVARCHAR(50),

@link NVARCHAR(500)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

UPDATE dbo.Pulpit

SET short\_name = @short\_name\_new, full\_name = @full\_name, link = @link

WHERE short\_name = @short\_name;

COMMIT;

SELECT short\_name, full\_name, link

FROM dbo.Pulpit

WHERE short\_name = @short\_name\_new;

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[RandString] Script Date: 22.12.2018 9:56:15 \*\*\*\*\*\*/

CREATE procedure [dbo].[RandString] (@r nvarchar(5) output)

as

begin

declare @s nvarchar(5)

SET @s = (

SELECT

c1 AS [text()]

FROM

(

SELECT TOP (5) c1

FROM

(

VALUES

('A'), ('B'), ('C'), ('D'), ('E'), ('F'), ('G'), ('H'), ('I'), ('J'),

('K'), ('L'), ('M'), ('N'), ('O'), ('P'), ('Q'), ('R'), ('S'), ('T'),

('U'), ('V'), ('W'), ('X'), ('Y'), ('Z'), ('0'), ('1'), ('2'), ('3'),

('4'), ('5'), ('6'), ('7'), ('8'), ('9')

) AS T1(c1)

ORDER BY ABS(CHECKSUM(NEWID()))

) AS T2

FOR XML PATH('')

);

set @r = @s

end

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ResultAttestationByNumber]

CREATE PROC [dbo].[ResultAttestationByNumber]

@id\_attestation INT

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT (s.surname + ',' + SPACE(1) + SUBSTRING(s.first\_name, 1, 1) + '.' + SPACE(1) + SUBSTRING( ISNULL(s.patronymic,''), 1, 1) + '.') AS name,

d.short\_name as discipline, r.form, m.name as mark, r.hours\_absent, t.short\_fio as teacher

FROM dbo.ResultAttestation r inner join dbo.Student s on r.record\_book = s.record\_book

inner join dbo.Mark m on r.mark = m.id\_mark

inner join dbo.Discipline d on r.discipline = d.id\_disc

inner join dbo.Teacher t on r.teacher = t.id\_teach

WHERE r.id\_attestation = @id\_attestation;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ResultAttestationByStydent]

CREATE PROC [dbo].[ResultAttestationByStydent]

@record\_book INT,

@id\_attestation INT = NULL

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT a.date\_begin, a.date\_end, d.short\_name, r.form, m.name as mark, r.hours\_absent, t.short\_fio

FROM dbo.ResultAttestation r inner join dbo.Attestation a on r.id\_attestation = a.id\_attestation

inner join dbo.Student s on r.record\_book = s.record\_book

inner join dbo.Mark m on r.mark = m.id\_mark

inner join dbo.Discipline d on r.discipline = d.id\_disc

inner join dbo.Teacher t on r.teacher = t.id\_teach

WHERE r.record\_book=@record\_book AND ( r.id\_attestation = @id\_attestation OR (@id\_attestation IS NULL))

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ResultAttestationDelete

CREATE PROC [dbo].[ResultAttestationDelete]

@rc INT OUTPUT,

@id\_attestation INT ,

@record\_book INT ,

@discipline NVARCHAR(20)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @id\_disc INT = -1;

SELECT @id\_disc = id\_disc FROM dbo.Discipline WHERE LOWER(short\_name) like LOWER(@discipline);

IF(@id\_disc!=-1 AND EXISTS(SELECT id\_attestation FROM dbo.ResultAttestation WHERE id\_attestation = @id\_attestation AND record\_book = @record\_book AND discipline = @id\_disc))

BEGIN

BEGIN TRAN;

DELETE

FROM dbo.ResultAttestation

WHERE id\_attestation = @id\_attestation AND record\_book = @record\_book AND discipline = @id\_disc;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ResultAttestationInsert]

CREATE PROC [dbo].[ResultAttestationInsert]

@rc INT OUTPUT,

@id\_attestation INT ,

@record\_book INT ,

@discipline NVARCHAR(20) ,

@teacher INT,

@mark NVARCHAR(15),

@hours\_absent INT ,

@form NCHAR(2)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @id\_mark INT = -1;

SELECT @id\_mark = id\_mark FROM dbo.Mark WHERE LOWER(name) like LOWER(@mark);

DECLARE @id\_disc INT = -1;

SELECT @id\_disc = id\_disc FROM dbo.Discipline WHERE LOWER(short\_name) like LOWER(@discipline);

IF( EXISTS(SELECT id\_attestation FROM dbo.ResultAttestation WHERE id\_attestation = @id\_attestation) AND

EXISTS(SELECT record\_book FROM dbo.Student WHERE record\_book = @record\_book) AND

EXISTS(SELECT id\_teach FROM dbo.Teacher WHERE id\_teach = @teacher) AND

@id\_mark != -1 AND @id\_disc != -1 AND @hours\_absent >=0 AND @form IN('ЛЗ','ПЗ','КП','лз','пз','кп') AND

(SELECT id\_group FROM dbo.Attestation WHERE id\_attestation = @id\_attestation)=(SELECT id\_group FROM dbo.Student WHERE record\_book = @record\_book))

BEGIN

BEGIN TRAN;

INSERT INTO dbo.ResultAttestation (id\_attestation, record\_book, discipline, teacher, mark, hours\_absent, form)

VALUES( @id\_attestation, @record\_book, @id\_disc, @teacher, @id\_mark, @hours\_absent, @form);

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ResultAttestationUpdate]

CREATE PROC [dbo].[ResultAttestationUpdate]

@rc INT OUTPUT,

@id\_attestation INT ,

@record\_book INT ,

@discipline NVARCHAR(20) ,

@teacher INT = NULL,

@mark NVARCHAR(15) = NULL,

@hours\_absent INT = NULL

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @id\_disc INT = -1;

SELECT @id\_disc = id\_disc FROM dbo.Discipline WHERE LOWER(short\_name) like LOWER(@discipline);

IF(@id\_disc!=-1 AND EXISTS(SELECT id\_attestation FROM dbo.ResultAttestation WHERE id\_attestation = @id\_attestation AND record\_book = @record\_book AND discipline = @id\_disc))

BEGIN

IF((@teacher IS NOT NULL) AND EXISTS(SELECT id\_teach FROM dbo.Teacher WHERE id\_teach = @teacher))

BEGIN

BEGIN TRAN;

UPDATE dbo.ResultAttestation

SET teacher = @teacher

WHERE id\_attestation = @id\_attestation AND record\_book = @record\_book AND discipline = @id\_disc;

COMMIT;

SET @rc = 1;

END

DECLARE @id\_mark INT = -1;

SELECT @id\_mark = id\_mark FROM dbo.Mark WHERE LOWER(name) like LOWER(@mark);

IF(@id\_mark !=-1)

BEGIN

BEGIN TRAN;

UPDATE dbo.ResultAttestation

SET mark = @id\_mark

WHERE id\_attestation = @id\_attestation AND record\_book = @record\_book AND discipline = @id\_disc;

COMMIT;

SET @rc = 1;

END

IF((@hours\_absent IS NOT NULL) AND @hours\_absent >0)

BEGIN

BEGIN TRAN;

UPDATE dbo.ResultAttestation

SET hours\_absent = @hours\_absent

WHERE id\_attestation = @id\_attestation AND record\_book = @record\_book AND discipline = @id\_disc;

COMMIT;

SET @rc = 1;

END

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ResultCTByStudent]

CREATE PROC [dbo].[ResultCTByStudent]

@record\_book INT

AS

SELECT s.name, r.score, r.year\_ct

FROM dbo.ResultCT r inner join dbo.SubjectCT s on r.id\_subj = s.id\_subj

WHERE record\_book = @record\_book;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ResultCTDelete]

CREATE PROC [dbo].[ResultCTDelete]

@rc INT OUTPUT,

@record\_book INT,

@name NVARCHAR(20)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @id INT = 0;

SELECT @id=id\_subj FROM dbo.SubjectCT WHERE LOWER(name) like LOWER(@name);

IF( EXISTS(SELECT record\_book FROM dbo.ResultCT WHERE record\_book = @record\_book AND id\_subj = @id))

BEGIN

BEGIN TRAN;

DELETE

FROM dbo.ResultCT

WHERE record\_book = @record\_book AND id\_subj = @id;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ResultCTInsert]

CREATE PROC [dbo].[ResultCTInsert]

@rc INT OUTPUT,

@record\_book INT,

@name1 NVARCHAR(20),

@score1 INT,

@year1 INT,

@name2 NVARCHAR(20),

@score2 INT,

@year2 INT,

@name3 NVARCHAR(20),

@score3 INT,

@year3 INT

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT record\_book FROM dbo.Student WHERE record\_book = @record\_book))

BEGIN

DECLARE @id1 INT = 0;

DECLARE @id2 INT = 0;

DECLARE @id3 INT = 0;

SELECT @id1=id\_subj FROM dbo.SubjectCT WHERE LOWER(name) like LOWER(@name1);

SELECT @id2=id\_subj FROM dbo.SubjectCT WHERE LOWER(name) like LOWER(@name2);

SELECT @id3=id\_subj FROM dbo.SubjectCT WHERE LOWER(name) like LOWER(@name3);

IF( @id1 != 0 AND @id2 != 0 AND @id3 != 0 AND

@score1>0 AND @score1<101 AND @score2>0 AND @score2<101 AND @score3>0 AND @score3<101 AND

@year1>2003 AND @year1 <= YEAR(GETDATE()) AND @year2>2003 AND @year2 <= YEAR(GETDATE()) AND @year3>2003 AND @year3 <= YEAR(GETDATE()) )

BEGIN

BEGIN TRAN;

INSERT INTO dbo.ResultCT (record\_book, id\_subj, score, year\_ct)

VALUES (@record\_book, @id1, @score1, @year1),

(@record\_book, @id2, @score2, @year2),

(@record\_book, @id3, @score3, @year3);

COMMIT;

SET @rc = 1;

END

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ResultCTUpdate]

CREATE PROC [dbo].[ResultCTUpdate]

@rc INT OUTPUT,

@record\_book INT,

@name NVARCHAR(20),

@score INT,

@year INT

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @id INT = 0;

SELECT @id=id\_subj FROM dbo.SubjectCT WHERE LOWER(name) like LOWER(@name);

IF( EXISTS(SELECT record\_book FROM dbo.ResultCT WHERE record\_book = @record\_book AND id\_subj = @id)

AND @score>0 AND @score<101 AND @year>2003 AND @year <= YEAR(GETDATE()))

BEGIN

BEGIN TRAN;

UPDATE dbo.ResultCT SET score = @score, year\_ct=@year WHERE record\_book = @record\_book AND id\_subj = @id;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ResultExamByNumber]

CREATE PROC [dbo].[ResultExamByNumber]

@number\_statement VARCHAR(10)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT r.number\_statement,surname +' '+ first\_name+' '+ ISNULL(patronymic,'') as name , r.record\_book, m.name as mark, m.type\_mark

FROM dbo.ResultExam r inner join dbo.Student s on r.record\_book = s.record\_book

inner join dbo.Mark m on r.mark = m.id\_mark

WHERE r.number\_statement like ('%'+ @number\_statement +'%') ;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ResultExamByStudent]

CREATE PROC [dbo].[ResultExamByStudent]

@record\_book INT

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT r.number\_statement, d.full\_name as discipline, a.name as [type], m.name as mark, e.date\_sig

FROM dbo.ResultExam r inner join dbo.ExamCredit e on r.number\_statement = e.number\_statement

inner join dbo.Mark m on r.mark = m.id\_mark

inner join dbo.Discipline d on d.id\_disc = e.discipline

inner join dbo.Attribute a on e.attribute = a.id\_atr

WHERE r.record\_book=@record\_book ;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ResultExamDelete]

CREATE PROC [dbo].[ResultExamDelete]

@rc INT OUTPUT,

@number\_statement VARCHAR(10),

@record\_book INT

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT number\_statement FROM dbo.ExamCredit WHERE number\_statement = @number\_statement) AND

EXISTS(SELECT record\_book FROM dbo.Student WHERE record\_book = @record\_book) AND

EXISTS(SELECT number\_statement FROM dbo.ResultExam WHERE number\_statement = @number\_statement AND record\_book = @record\_book))

BEGIN

BEGIN TRAN;

DELETE

FROM dbo.ResultExam

WHERE number\_statement = @number\_statement AND record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ResultExamInsert]

CREATE PROC [dbo].[ResultExamInsert]

@rc INT OUTPUT,

@number\_statement VARCHAR(10),

@record\_book INT,

@mark NVARCHAR(15)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @id\_mark INT = -1;

SELECT @id\_mark = id\_mark FROM dbo.Mark WHERE LOWER(name) like LOWER(@mark)

IF( EXISTS(SELECT number\_statement FROM dbo.ExamCredit WHERE number\_statement = @number\_statement) AND

EXISTS(SELECT record\_book FROM dbo.Student WHERE record\_book = @record\_book) AND @id\_mark != -1)

BEGIN

BEGIN TRAN;

INSERT INTO dbo.ResultExam(number\_statement, record\_book, mark)

VALUES( @number\_statement, @record\_book, @id\_mark);

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ResultExamUpdate]

CREATE PROC [dbo].[ResultExamUpdate]

@rc INT OUTPUT,

@number\_statement VARCHAR(10),

@record\_book INT,

@mark NVARCHAR(15)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @id\_mark INT = -1;

SELECT @id\_mark = id\_mark FROM dbo.Mark WHERE LOWER(name) like LOWER(@mark)

IF( EXISTS(SELECT number\_statement FROM dbo.ExamCredit WHERE number\_statement = @number\_statement) AND

EXISTS(SELECT record\_book FROM dbo.Student WHERE record\_book = @record\_book) AND @id\_mark != -1 AND

EXISTS(SELECT number\_statement FROM dbo.ResultExam WHERE number\_statement = @number\_statement AND record\_book = @record\_book))

BEGIN

BEGIN TRAN;

UPDATE dbo.ResultExam

SET mark = @id\_mark

WHERE number\_statement = @number\_statement AND record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[SearchStudent]

CREATE PROC [dbo].[SearchStudent]

@param NVARCHAR(100)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT surname +' '+ first\_name+' '+ ISNULL(patronymic,'') as FIO ,course, group\_number,short\_name as speciality, sex, date\_birth, form, leader, disband, restored

FROM dbo.Student s inner join dbo.GroupStudent g on s.id\_group = g.id\_group

inner join dbo.Speciality sp on g.kod\_special = sp.kod\_special

WHERE LOWER(surname +' '+ first\_name+' '+ ISNULL(patronymic,'')) LIKE '%' + LOWER(@param) + '%';

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[StudentByRecordBook]

CREATE PROC [dbo].[StudentByRecordBook]

@record\_book INT

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT surname +' '+ first\_name+' '+ ISNULL(patronymic,'') as name ,course, group\_number,short\_name as speciality, sex, date\_birth, form, leader, disband, restored

FROM dbo.Student s inner join dbo.GroupStudent g on s.id\_group = g.id\_group

inner join dbo.Speciality sp on g.kod\_special = sp.kod\_special

WHERE s.record\_book = @record\_book;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[StudentByRecordBookForCertificate]

CREATE PROC [dbo].[StudentByRecordBookForCertificate]

@record\_book INT

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT TOP(1) surname + ' '+ first\_name + ' '+ ISNULL(patronymic,'') as fio,course, group\_number, sp.short\_name as speciality,

CASE form WHEN 'п' THEN 'платно' ELSE 'за счёт средств бюджета' END as form, g.begin\_learning, g.end\_learning

FROM dbo.Student s inner join dbo.GroupStudent g on s.id\_group = g.id\_group

inner join dbo.Speciality sp on g.kod\_special = sp.kod\_special

WHERE s.record\_book = @record\_book;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[StudentDelete]

CREATE PROC [dbo].[StudentDelete]

@rc INT OUTPUT,

@record\_book INT,

@sex NCHAR(1)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT record\_book FROM dbo.Student WHERE record\_book=@record\_book))

BEGIN

BEGIN TRAN;

DELETE

dbo.Student WHERE record\_book=@record\_book;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[StudentInsert]

CREATE PROC [dbo].[StudentInsert]

@rc INT OUTPUT,

@record\_book INT,

@surname NVARCHAR(15),

@first\_name NVARCHAR(20),

@patronymic NVARCHAR(15) = NULL,

@sex NCHAR(1),

@date\_birth DATE,

@course INT,

@group\_number INT,

@form NCHAR(1),

@leader BIT =0

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

DECLARE @id\_group NVARCHAR(18) = NULL;

SELECT @id\_group = id\_group FROM dbo.GroupStudent WHERE course = @course AND group\_number = @group\_number;

IF( @id\_group IS NOT NULL AND (@sex in('М','Ж')) AND @date\_birth<GETDATE() AND (@form in('П','Б')) AND

NOT EXISTS(SELECT record\_book FROM dbo.Student WHERE record\_book=@record\_book) AND

((@leader = 1 AND NOT EXISTS(SELECT record\_book FROM dbo.Student WHERE id\_group = @id\_group AND leader = 1) OR @leader = 0)))

BEGIN

BEGIN TRAN;

INSERT INTO dbo.Student(record\_book,surname, first\_name,patronymic, sex, date\_birth, id\_group, form, leader)

VALUES (@record\_book, @surname, @first\_name, @patronymic, LOWER(@sex), @date\_birth, @id\_group, LOWER(@form), @leader);

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[StudentIsLeader]

CREATE PROC [dbo].[StudentIsLeader]

@login NVARCHAR(16),

@rc INT OUTPUT

AS

BEGIN

DECLARE @leader BIT = 0;

SELECT @leader = leader FROM dbo.Student WHERE record\_book = CAST( @login as INT);

IF(@leader=1)

SET @rc = 1;

ELSE

SET @rc = 0;

END

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[StudentsByRecordBookLeader]

CREATE PROC [dbo].[StudentsByRecordBookLeader]

@record\_book INT

AS

DECLARE @id NVARCHAR(18) EXEC dbo.GetGroupIdByStudent @record\_book, @id out;

SET XACT\_ABORT, NOCOUNT ON;

SELECT s.record\_book, surname +' '+ first\_name+' '+ ISNULL(patronymic,'') as name ,sex, date\_birth, form, leader

FROM dbo.Student s inner join dbo.GroupStudent g on s.id\_group = g.id\_group

inner join dbo.Speciality sp on g.kod\_special = sp.kod\_special

WHERE g.id\_group = @id order by surname;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[StudentUpdate]

CREATE PROC [dbo].[StudentUpdate]

@rc INT OUTPUT,

@record\_book INT,

@surname NVARCHAR(15) = NULL,

@first\_name NVARCHAR(20) = NULL,

@patronymic NVARCHAR(15) = NULL,

@sex NCHAR(1) = NULL,

@date\_birth DATE = NULL,

@form NCHAR(1) = NULL,

@leader BIT = NULL

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT record\_book FROM dbo.Student WHERE record\_book=@record\_book))

BEGIN

IF(@surname IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.Student SET surname= @surname WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@first\_name IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.Student SET first\_name= @first\_name WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@patronymic IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.Student SET patronymic= @patronymic WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@sex IS NOT NULL AND (@sex in('М','Ж')))

BEGIN

BEGIN TRAN;

UPDATE dbo.Student SET sex= @sex WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@date\_birth IS NOT NULL AND @date\_birth<GETDATE())

BEGIN

BEGIN TRAN;

UPDATE dbo.Student SET date\_birth= @date\_birth WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@form IS NOT NULL AND (@form in('П','Б')))

BEGIN

BEGIN TRAN;

UPDATE dbo.Student SET form= @form WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

IF(@leader IS NOT NULL )

BEGIN

BEGIN TRAN;

UPDATE dbo.Student SET leader= @leader WHERE record\_book = @record\_book;

COMMIT;

SET @rc = 1;

END

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[SubjectCTDelete]

CREATE PROC [dbo].[SubjectCTDelete]

@id INT

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

DELETE

FROM dbo.SubjectCT

WHERE id\_subj = @id

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[SubjectCTInsert]

CREATE PROC [dbo].[SubjectCTInsert]

@name NVARCHAR(20)

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

INSERT INTO dbo.SubjectCT( name )

VALUES( @name )

SELECT id\_subj, name

FROM dbo.SubjectCT

WHERE id\_subj = SCOPE\_IDENTITY()

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[SubjectCTSelect]

@id INT

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

SELECT id\_subj, name

FROM dbo.SubjectCT

WHERE (id\_subj = @id OR @id IS NULL)

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[SubjectCTUpdate]

CREATE PROC [dbo].[SubjectCTUpdate]

@id INT,

@name NVARCHAR(20)

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

UPDATE dbo.SubjectCT

SET name=@name

WHERE id\_subj = @id

SELECT id\_subj, name

FROM dbo.SubjectCT

WHERE id\_subj = @id

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[TeacherDelete]

CREATE PROC [dbo].[TeacherDelete]

@id INT

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

BEGIN TRAN;

DELETE

FROM dbo.Teacher

WHERE id\_teach = @id;

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[TeacherInsert]

CREATE PROC [dbo].[TeacherInsert]

@rc INT OUTPUT,

@short\_fio NVARCHAR(10),

@full\_fio NVARCHAR(50),

@pulpit NVARCHAR(10),

@work BIT = 1

AS

BEGIN TRY

SET @rc = 0;

SET XACT\_ABORT, NOCOUNT ON

IF EXISTS(SELECT short\_name FROM dbo.Pulpit WHERE short\_name like @pulpit)

BEGIN

BEGIN TRAN;

INSERT INTO dbo.Teacher (short\_fio, full\_fio, pulpit, work)

VALUES(@short\_fio, @full\_fio, @pulpit, @work);

SET @rc = 1;

COMMIT;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[TeacherSelect]

CREATE PROC [dbo].[TeacherSelect]

@name NVARCHAR(50)

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT id\_teach, short\_fio, full\_fio, pulpit, work

FROM dbo.Teacher

WHERE (LOWER(full\_fio) like('%'+ LOWER(@name) + '%') OR @name IS NULL);

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[TeacherUpdate]

CREATE PROC [dbo].[TeacherUpdate]

@rc BIT OUTPUT,

@id INT,

@short\_fio NVARCHAR(10) = NULL,

@full\_fio NVARCHAR(50) = NULL,

@pulpit NVARCHAR(10) = NULL,

@work BIT = NULL

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF(@short\_fio IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.Teacher SET short\_fio = @short\_fio WHERE id\_teach = @id;

COMMIT;

SET @rc = 1;

END

IF(@full\_fio IS NOT NULL)

BEGIN

BEGIN TRAN;

UPDATE dbo.Teacher SET full\_fio = @full\_fio WHERE id\_teach = @id;

COMMIT;

SET @rc = 1;

END

IF(@pulpit IS NOT NULL)

BEGIN

IF EXISTS(SELECT short\_name FROM dbo.Pulpit WHERE short\_name like @pulpit)

BEGIN

BEGIN TRAN;

UPDATE dbo.Teacher SET pulpit = @pulpit WHERE id\_teach = @id;

COMMIT;

SET @rc = 1;

END

ELSE SET @rc = -1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[TypeAddressDelete]

CREATE PROC [dbo].[TypeAddressDelete]

@id INT

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

DELETE

FROM dbo.TypeAddress

WHERE id\_type = @id

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[TypeAddressInsert]

CREATE PROC [dbo].[TypeAddressInsert]

@name NVARCHAR(30)

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

INSERT INTO dbo.TypeAddress ( name\_type )

VALUES( @name )

SELECT id\_type, name\_type

FROM dbo.TypeAddress

WHERE id\_type = SCOPE\_IDENTITY()

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[TypeAddressSelect]

CREATE PROC [dbo].[TypeAddressSelect]

@id INT

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

SELECT id\_type, name\_type

FROM dbo.TypeAddress

WHERE (id\_type = @id OR @id IS NULL)

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[TypeAddressUpdate]

CREATE PROC [dbo].[TypeAddressUpdate]

@id INT,

@name NVARCHAR(30)

AS

SET XACT\_ABORT, NOCOUNT ON

BEGIN TRAN

UPDATE dbo.TypeAddress

SET name\_type=@name

WHERE id\_type = @id

SELECT id\_type, name\_type

FROM dbo.TypeAddress

WHERE id\_type = @id

COMMIT

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[UserIsAdmin]

CREATE PROC [dbo].[UserIsAdmin]

@login NVARCHAR(16),

@password NVARCHAR(20),

@rc INT = -1 OUTPUT

AS

BEGIN

DECLARE @admin BIT = 0;

SELECT @admin = user\_admin FROM dbo.Users WHERE user\_login like @login and user\_password like @password;

IF(@admin=1)

SET @rc = 1;

ELSE

SET @rc = 0;

END

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[UsersDelete]

CREATE PROC [dbo].[UsersDelete]

@rc INT OUTPUT,

@login NVARCHAR(16),

@password NVARCHAR(20)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT user\_login FROM dbo.Users WHERE user\_login = @login AND user\_password = @password))

BEGIN

BEGIN TRAN;

DELETE

FROM dbo.Users WHERE user\_login = @login AND user\_password = @password;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[UsersInsert]

CREATE PROC [dbo].[UsersInsert]

@rc INT OUTPUT,

@login NVARCHAR(16),

@password NVARCHAR(20),

@admin BIT

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF(NOT EXISTS(SELECT user\_login FROM dbo.Users WHERE user\_login = @login) AND LEN(@password)<4)

BEGIN

IF((@admin = 0 AND EXISTS(SELECT record\_book FROM dbo.Student WHERE record\_book = CAST( @login as INT))) OR @admin = 1 )

BEGIN

BEGIN TRAN;

INSERT INTO dbo.Users( user\_login, user\_password, user\_admin)

VALUES (@login, @password, @admin);

COMMIT;

SET @rc = 1;

END

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[UsersSelect]

CREATE PROC [dbo].[UsersSelect]

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT user\_login, user\_password, user\_admin

FROM dbo.Users

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[UsersUpdate]

CREATE PROC [dbo].[UsersUpdate]

@rc INT OUTPUT,

@login NVARCHAR(16),

@password NVARCHAR(20),

@password\_new NVARCHAR(20)

AS

BEGIN TRY

SET XACT\_ABORT, NOCOUNT ON;

SET @rc = 0;

IF( EXISTS(SELECT user\_login FROM dbo.Users WHERE user\_login = @login AND user\_password = @password) AND LEN(@password\_new)>3)

BEGIN

BEGIN TRAN;

UPDATE dbo.Users

SET user\_password = @password\_new

WHERE user\_login = @login AND user\_password = @password;

COMMIT;

SET @rc = 1;

END

END TRY

BEGIN CATCH

SET @rc = -1;

ROLLBACK;

SELECT ERROR\_NUMBER() AS ErrorNumber,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

# 

# Приложение Д

CREATE PROC [dbo].[ExportToExcel]

AS

SET XACT\_ABORT, NOCOUNT ON;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Absents$]') select \* from Deanery.dbo.Absents;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [AdditionalInfo$]') select \* from Deanery.dbo.AdditionalInfo;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [AddressStudent$]') select \* from Deanery.dbo.AddressStudent;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Attestation$]') select \* from Deanery.dbo.Attestation;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Attribute$]') select \* from Deanery.dbo.Attribute;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Certificate$]') select \* from Deanery.dbo.[Certificate];

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Discipline$]') select \* from Deanery.dbo.Discipline;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [District$]') select \* from Deanery.dbo.District;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [EducationalProcess$]') select \* from Deanery.dbo.EducationalProcess;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Enrollment$]') select \* from Deanery.dbo.Enrollment;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [ExamCredit$]') select \* from Deanery.dbo.ExamCredit;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Expelled$]') select \* from Deanery.dbo.Expelled;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [FormProcess$]') select \* from Deanery.dbo.FormProcess;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [GroupStudent$]') select \* from Deanery.dbo.GroupStudent;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Lenguage$]') select \* from Deanery.dbo.Lenguage;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [ListEnrollment$]') select \* from Deanery.dbo.ListEnrollment;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [ListExpelled$]') select \* from Deanery.dbo.ListExpelled;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [ListNotAttestation$]') select \* from Deanery.dbo.ListNotAttestation;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Mark$]') select \* from Deanery.dbo.Mark;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Messenger$]') select \* from Deanery.dbo.Messenger;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Parents$]') select \* from Deanery.dbo.Parents;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [PlanDiscipline$]') select \* from Deanery.dbo.PlanDiscipline;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Profession$]') select \* from Deanery.dbo.Profession;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Pulpit$]') select \* from Deanery.dbo.Pulpit;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Region$]') select \* from Deanery.dbo.Region;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [ResultAttestation$]') select \* from Deanery.dbo.ResultAttestation;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [ResultCT$]') select \* from Deanery.dbo.ResultCT;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [ResultExam$]') select \* from Deanery.dbo.ResultExam;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Speciality$]') select \* from Deanery.dbo.Speciality;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Student$]') select \* from Deanery.dbo.Student;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [SubjectCT$]') select \* from Deanery.dbo.SubjectCT;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Teacher$]') select \* from Deanery.dbo.Teacher;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [TypeAddress$]') select \* from Deanery.dbo.TypeAddress;

insert into OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\DB.xlsx;',

'SELECT \* FROM [Users$]') select \* from Deanery.dbo.Users;

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[UsersToXML] Script Date: 22.12.2018 9:56:15 \*\*\*\*\*\*/

GO

CREATE PROC [dbo].[UsersToXML]

AS

SELECT \* FROM Deanery.dbo.Users for XML PATH('User'), Root('Users')

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[ImportToExcel] Script Date: 22.12.2018 9:56:15 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE PROC [dbo].[ImportToExcel]

AS

SET XACT\_ABORT, NOCOUNT ON;

SELECT \* INTO #temp FROM OPENROWSET('Microsoft.ACE.OLEDB.12.0',

'Excel 12.0;Database=D:\Важно\КП\CW\SQL\FIT2.xlsm',

'SELECT [№ зачетки], [фамилия], [имя], [отчество], [Пол], [Дата], [Курс], [Группа], [Вид дого-вора],[староста], [Адрес], [Телефоны] FROM [IT$]');

DECLARE @record\_book INT,@surname NVARCHAR(15), @first\_name NVARCHAR(20), @patronymic NVARCHAR(15),

@sex NCHAR(1), @date\_birth DATE, @course INT, @number INT, @form NVARCHAR(15), @leader BIT,

@address NVARCHAR(100), @messenger NVARCHAR(100);

DECLARE curs CURSOR LOCAL DYNAMIC

FOR

SELECT \* FROM #temp;

OPEN curs;

FETCH NEXT FROM curs

INTO @record\_book, @surname, @first\_name, @patronymic, @sex, @date\_birth, @course, @number, @form, @leader, @address, @messenger;

WHILE @@fetch\_status = 0

BEGIN

DECLARE @f NCHAR(1) = 'п';

IF(SUBSTRING(@form, 1, 1) = '9')

SET @f = 'б';

DECLARE @rc INT, @rb INT, @l BIT = ISNULL(@leader,0);

exec Deanery.dbo.StudentInsert @rc out, @record\_book, @surname, @first\_name,@patronymic, @sex, @date\_birth, @course, @number, @f, @l;

IF(@address is not null)

exec Deanery.dbo.AddressStudentInsert @rc, @record\_book,'',@address, 1;

IF(@messenger is not null)

exec Deanery.dbo.MessengerInsert @rc, @record\_book, @messenger;

FETCH NEXT FROM curs

INTO @record\_book, @surname, @first\_name, @patronymic, @sex, @date\_birth, @course, @number, @form, @leader, @address, @messenger;

END;

CLOSE curs;

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[UsersFromXML] Script Date: 22.12.2018 9:56:15 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE PROC [dbo].[UsersFromXML]

AS

DECLARE @x xml

SELECT @x = P

FROM OPENROWSET(bulk 'D:\Важно\КП\CW\SQL\UsersXML.xml',single\_blob) as Users(P)

DECLARE @h int = 0;

EXEC sp\_xml\_preparedocument @h output, @x;

INSERT INTO Deanery.dbo.Users

SELECT \* FROM OPENXML(@h, '/Users/User', 2)

WITH(

user\_login NVARCHAR(16),

user\_password NVARCHAR(20),

user\_admin BIT)

GO

# Приложение Е

<User>

<user\_login>71600135</user\_login>

<user\_password>123456</user\_password>

<user\_admin>0</user\_admin>

</User>

<User>

<user\_login>admin</user\_login>

<user\_password>admin</user\_password>

<user\_admin>1</user\_admin>

</User>

</Users>