Оглавление

[Введение 4](#_Toc531216125)

[1. Постановка задачи 5](#_Toc531216126)

[2. Разработка модели базы данных 6](#_Toc531216127)

[3. Разработка необходимых объектов 8](#_Toc531216128)

[3.1. Таблицы 8](#_Toc531216129)

[3.2. Пользователи 8](#_Toc531216130)

[3.3. Процедуры и функции 8](#_Toc531216131)

[4. Описание процедур импорта и экспорта данных 10](#_Toc531216132)

[4.1. Описание процедуры импорта данных 10](#_Toc531216133)

[4.2. Описание процедуры экспорта данных 10](#_Toc531216134)

[5. Тестирование производительности 11](#_Toc531216135)

[5.1. Тестирование производительности базы данных 11](#_Toc531216136)

[6. Описание технологии 14](#_Toc531216137)

[6.1. Описание применения технологии 14](#_Toc531216138)

[7. Руководство пользователя 18](#_Toc531216139)

[Заключение 24](#_Toc531216140)

[Список используемых источников 25](#_Toc531216141)

[Приложение А 26](#_Toc531216142)

[Приложение Б 27](#_Toc531216143)

[Приложение В 28](#_Toc531216144)

[Приложение Г 32](#_Toc531216145)

# Введение

Базы данных — это совокупность структур, предназначенных для хранения больших объемов информации и программных модулей, осуществляющих управление данными, их выборку, сортировку и другие подобные действия.

Информация базы данных хранится в одной или нескольких таблицах. Любая таблица с данными состоит из набора однотипных записей, расположенных друг за другом. Они представляют собой строки таблицы, которые можно добавлять, удалять или изменять.

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.

Сегодня большое значение имеет работа с данными. Для хранения данных используются различные системы управления базами данных: MS SQL Server, Oracle, MySQL и так далее. И большинство крупных приложений, так или иначе используют для хранения данных эти системы управления базами данных.

# Постановка задачи

Цифровые технологии сильно изменили жизнь человека. Компьютер предоставляет большие возможности для работы с разного вида информацией.

В настоящее время существует большое количество программ, которые позволяют создавать и проходить тесты. Компьютер как средство для тестирования имеет свои достоинства и недостатки. Однако в объективности оценки ему отказать трудно. Видимо, именно этот фактор способствует расширению сферы применения компьютерного тестирования — оно успешно используется в школах, вузах, при поступлении на работу, при сдаче экзаменов в ГИБДД и т.д.

Сегодня разработчики предлагают разнообразные компьютерные инструменты для подготовки тестовых заданий, начиная от различных редакторов и программ для разработки презентаций и до использования языков программирования и возможностей сети Интернет. Но разработка качественного тестового инструментария – длительный, трудоемкий и дорогостоящий процесс.

В соответствии с заданием курсового проекта следует создать базу данных (БД) для хранения тестов с использованием программного продукта Microsoft SQL Server 2017. Так же надо создать клиентское приложение, представляющее собой интерфейс для взаимодействия с базой данных.

# Разработка модели базы данных

Для создания, хранения и управления данными, была использованная реляционная СУБД Microsoft SQL Server. Ниже на рисунке 2 .1 представлена модель базы данных (таблицы, отмеченные на них первичных и вторичных ключей) курсового проекта. База данных была подвергнута процедуре нормализации данных.

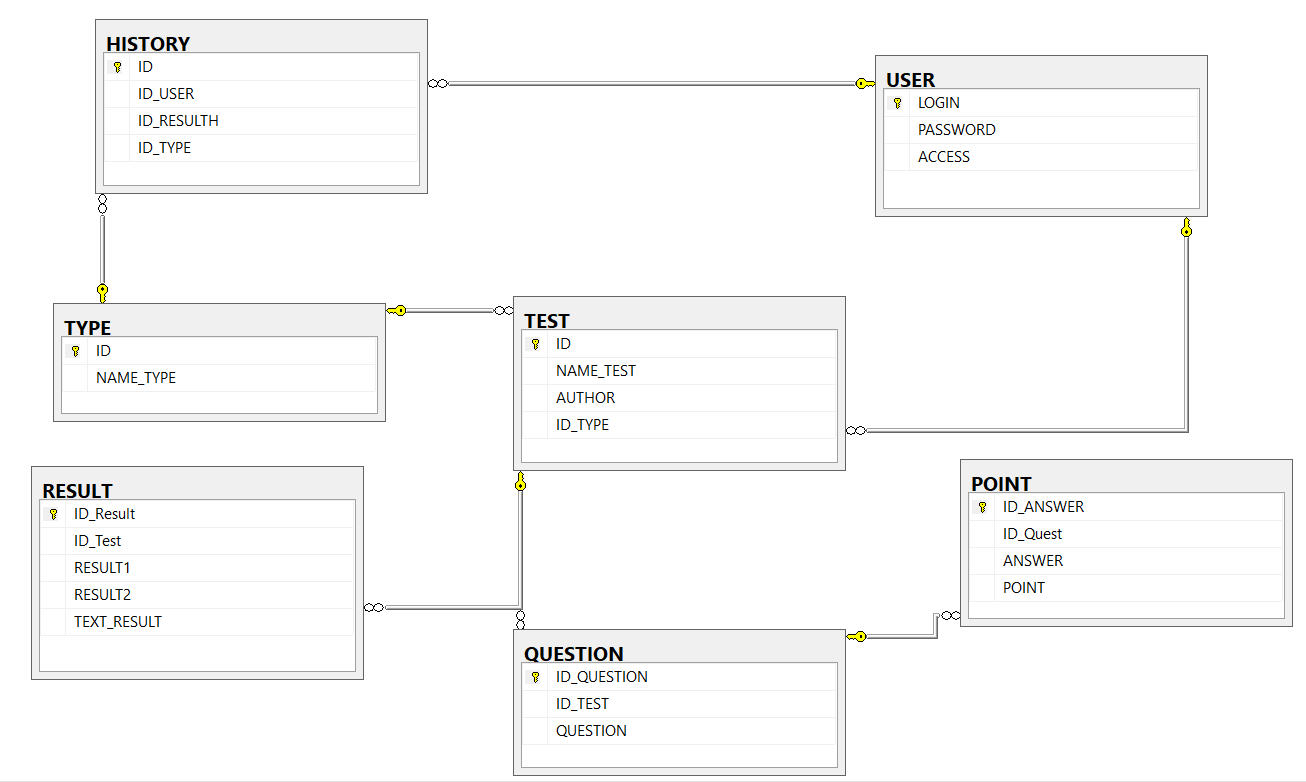


Рисунок 2.1 – Диаграмма модели базы данных

База данных на выше представленной схеме в полной мере описывает необходимые таблицы для решения поставленной задачи данного курсового проекта.

Таблица HISTORY содержит информацию об истории пользователя:

* ID – первичный ключ, идентификатор истории;
* ID\_USER – идентификатор пользователя;
* ID\_RESULTH – идентификатор результата;
* ID\_TYPE – идентификатор типа теста;

Таблица TYPE содержит информацию о типах тестов:

* ID – первичный ключ, идентификатор типа;
* NAME\_TYPE – наименование типа.

Таблица USER содержит информацию о пользователях:

* LOGIN – первичный ключ, идентификатор пользователя;
* PASSWORD – пароль пользователя;
* ACCESS – идентификатор доступа;

Таблица TEST содержит данные о тестах:

* ID –первичный ключ, идентификатор теста;
* NAME\_TEST – название теста;
* AUTHOR – идентификатор создателя;
* ID\_TYPE– идентификатор типа теста;

Таблица QUESTION содержит данные о вопросах:

* ID\_QUESTION – первичный ключ, идентификатор вопроса;
* ID\_TEST – идентификатор теста, к которому относится вопрос;
* QUESTION – текст вопроса;

Таблица POINT содержит данные об ответах на вопросы и баллы за них:

* ID\_ANSWER – первичный ключ, идентификатор ответа;
* ID\_Quest – идентификатор вопроса;
* ANSWER – текст ответа;
* POINT – балл за ответ;

Таблица RESULT содержит данные о результатах тестов:

* ID\_Result – первичный ключ, идентификатор результата;
* ID\_Test – идентификатор теста;
* RESULT1 – нижняя граница числового результата;
* RESULT2 – верхняя граница числового результата;
* TEXT\_RESULT- текст результата.

# Разработка необходимых объектов

База данных данного курсового проекта содержит следующие объекты: таблицы, хранимые процедуры, пользователи, функции. Их подробное описание представлено ниже.

## Таблицы

База данных тестов содержит 6 таблиц, которые описаны в главе 2. SQL-скрипты для создания таблиц находятся в приложении А.

## Пользователи

Пользователь базы данных – это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации. При проектировании базы данных понадобилось 2 пользователя(admin и user). У каждого пользователя есть свои права, предназначенные для выполнения соответствующих ему требований.

Пользователь admin наделен правами, имитирующими администратора, который может проходить и удалять тесты.

Пользователь user наделен правами, имитирующего пользователя, который может создавать, проходить и искать тесты, а также смотреть свою историю.

Скрипты назначения привилегий пользователям приведены в приложении Б.

## Процедуры и функции

Хранимая процедура – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Их использование в курсовом проекте предназначено для ограничения доступа пользователей к таблицам базы данных, оставив пользователям только разрешения на выполнение некоторых процедур, обеспечивающих косвенный и строго регламентированный доступ к данным.

Процедуры, разработанные на данном курсовом проекте можно разделить на два типа: процедуры CRUD (Create, Read, Update, Delete) для каждой таблицы (используя DML - операторы) и процедуры выборки по таблицам.

Администратору доступны следующие процедуры и функции:

* AutoValid – проверка авторизации;
* Take\_access – получение уровня доступа;
* IfEmpty – проверка уникальности логина;
* InserUser – добавление пользователя;
* DeleteUser – удаление пользователя;
* UsersSelect – список пользователей;
* Take\_id\_Test– получение id теста;
* DeleteTest – удаление теста;
* DeleteQuestionByTest – удаление вопроса;
* IdQuestionByIdTest – получение id вопроса;
* DeletePointByIdQuest– удаление ответа;
* DeleteResultBiIdTest – удаление результата;
* IdResultIdTest – получение id результата;
* DeleteHistoryByIdResult – удаление истории;
* ListTestsSelect – список тестов;
* TestSelectListByAuthor – список тестов;
* GetIdResultByLogin – список id результатов;
* GetTestIdByResultId – список id тестов;
* GetNameTest– список тестов;
* QuestionidByText – список id вопросов;
* GetQuestionById – список вопросов;
* QuestionSelectListByIdTest – список вопросов;
* AnswerSelectListByIdQuest – список ответов;
* ShowResult – получение id результата;
* GetTextRes – получение результата;
* GetTypeById – получение типа теста.

Пользователю доступны следующие процедуры и функции:

* Take\_id\_Test – получение id теста;
* IdQuestionByIdTest – получение id вопроса;
* IdResultIdTest – получение id результата;
* ListTestsSelect – список тестов;
* QuestionidByText – список id вопросов;
* GetQuestionById – список вопросов;
* AnswerSelectListByIdQuest – список ответов;
* ShowResult – получение id результата;
* GetTextRes – получение результата;
* GetTypeById – получение типа теста;
* SearchTestByName – поиск теста;
* TestCreate – создание теста;
* QuestionByTestEmpty – проверка заполнения вопроса;
* AddNewQuestion – создание вопроса;
* NumberQuest – получение количества вопросов;
* AddNewPoint – создание ответа;
* Ball – получение баллов за ответ;
* AddNewResult – создание результата;
* AddNewHistory – создание истории пользователя.

Листинг SQL-кода всех процедур находится в приложении В.

# Описание процедур импорта и экспорта данных

В данной курсовой работе реализованы процедуры экспорта и импорта данных из xml файла в базу данных таблицы USER и наоборот. При данных операциях работает с файловой системой приложение, а разбором и генерацией xml занимается sql server.

## Описание процедуры импорта данных

Существует несколько вариантов процедуры импорта данных xml в таблицы MS SQL Server.

Были выполнены две цели, для выполнения преобразования данных из xml:

* импорт данных xml в MS SQL Server с помощью функции OPENROWSET;
* преобразование формата xml с помощью BULK.

Код создания процедуры импорта представлен на рисунке 4.1.

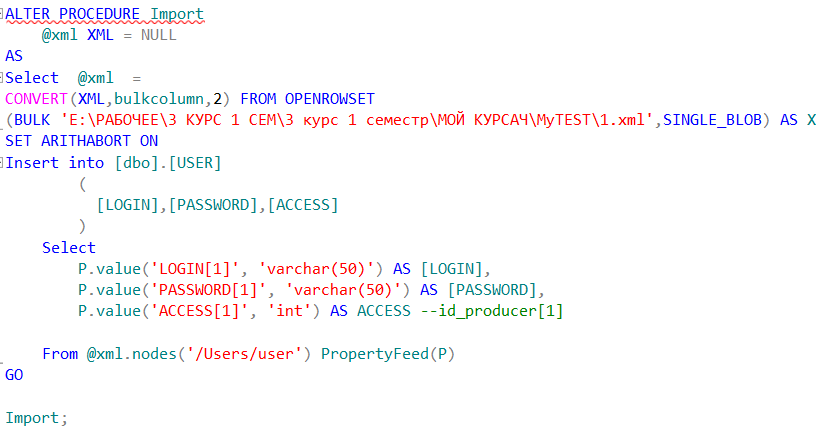


Рисунок 4.1 – процедура импорта из xml в таблицу USER

## Описание процедуры экспорта данных

Код создания процедуры экспорта данных таблицы в xml формат представлен на рисунке 4.2.

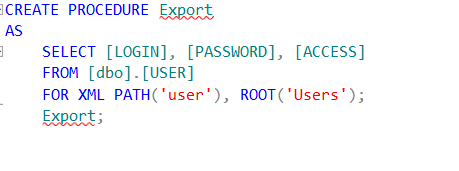


Рисунок 4.2 – процедура экспорта данных

# Тестирование производительности

Оптимизация запросов — процесс изменения запроса и/или структуры БД с целью уменьшения использования вычислительных ресурсов при выполнении запроса. Один и тот же результат может быть получен СУБД различными способами (планами выполнения запросов), которые могут существенно отличаться как по затратам ресурсов, так и по времени выполнения.

В MS SQL Server оптимизация запросом в основном заключается в построение индексов над таблицами, и изменением плана запроса. Индекс – это объект базы данных, предназначенный для ускорения запросов к данным в таблице базы данных. MSS поддерживает два типа индексов: кластеризованные и некластеризованные индексы.

При создании кластеризованного индекса данные индексируемой таблицы располагаются в физическом порядке, соответствующем индексу, и становятся частью кластеризованного индекса. Поэтому кластеризованный индекс для таблицы может быть создан только один.

Некластеризованный индекс – это отдельный объект, имеющий указатели на строки таблицы. Максимальное количество некластерированных индексов для одной таблицы не должно превышать 1000.

## Тестирование производительности базы данных

Для тестирования базы данных была выбрана таблица USER. В базе данных данного курсового проекта в каждой таблице находится поле с уникальным значением поля — id типа int, соответственно каждая таблица содержит кластеризованный индекс. База данных содержит множество процедур с выборкой содержащий оператор WHERE сравнивающий строки. Наиболее частое сравнение — это логин и пароль для проверки на корректность запроса от пользователя. Даже при незначительном заполнении базы данных (около 100000 записей в таблице) время выборки начинает занимать существенное время. Ниже на рисунке 5.1 представлена карта запроса при выборке в таблице пользователей.

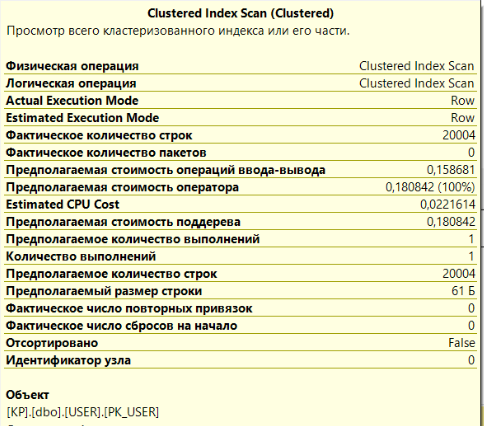


Рисунок 5.1 – карта запроса без некластеризованного индекса

Как можно увидеть, стоимость запроса достаточно велика по сравнению с таблицей, уже содержащей некластеризованный индекс. Теперь для сравнения добавим некластеризованный индекс к этой таблице для поля PASSWORD. При создании индекса указывается один или несколько столбцов таблицы, по значениям которых будет построен индекс. Результат представлен на рисунке 5.2.

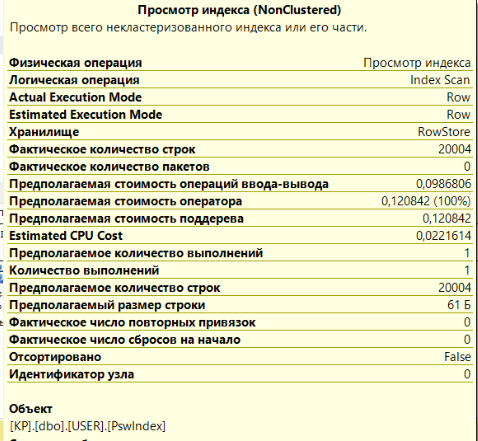


Рисунок 5.2 – карта запроса с некластеризованным индексом

Так же был создан индекс для поля ACCESS. Карта запроса с этим кластеризованным индексом представлена на рисунке 5.3.

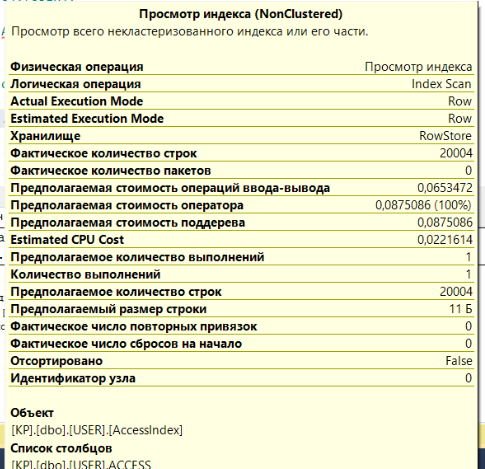


Рисунок 5.3 – карта запроса с некластеризованным индексом

По результатам стоимости запроса можно сказать, что построение индексов в данной таблице максимально необходимо, так как во втором случае общая стоимость запроса уменьшилась более чем в 2 раза. Примеры создания индексов находятся в приложении Г.

# Описание технологии

В данном курсовом проекте была выбрана технология полнотекстового поиска. Полнотекстовый поиск — это поиск слов или фраз в текстовых данных. Обычно такой вид поиска используется для поиска текста в большом объёме данных, так как он значительно быстрей обычного поиска, который можно осуществить, используя конструкцию LIKE.

## Описание применения технологии

Полнотекстовый поиск подразумевает создание специального индекса текстовых данных, который представляет собой некий словарь слов, которые встречаются в этих данных. Так как помимо своей быстрой работы он обладает ещё и возможностью ранжировать найденные строки, т.е. выставлять ранг каждой найденной строке. Можно найти самые релевантные записи, т.е. самые подходящие под запрос.

Возможности полнотекстового поиска в MS SQL Server :

* в полнотекстовом поиске SQL сервера можно осуществлять поиск не только по отдельным словам или фразам, но и по префиксным выражениям, например, задать текст начала слова или фразы;
* можно искать слова по словоформам, например, различные формы глаголов или существительные в единственном и во множественном числе, т.е. по производным выражениям;
* можно построить запрос так, чтобы найти слова или фразы, находящиеся рядом с другими словами или фразами, т.е. выражения с учетом расположения;
* есть возможность искать синонимические формы конкретного слова.

Перед тем как приступать к созданию полнотекстового поиска, необходимо знать несколько важных моментов:

* для реализации полнотекстового поиска компонент Full-Text Search (Полнотекстовый поиск) должен быть установлен;
* у таблицы может быть только один полнотекстовый индекс;
* чтобы создать полнотекстовый индекс, таблица должна содержать один уникальный индекс, который включает один столбец и не допускает значений NULL. Рекомендовано использовать уникальный кластеризованный индекс (или просто первичный ключ), первый столбец которого должен иметь целочисленный тип данных;
* полнотекстовый индекс можно создавать на столбцах с типом данных: char, varchar, nchar, nvarchar, text, ntext, image, xml, varbinary или varbinary(max).

Для того чтобы создать полнотекстовый индекс сначала необходимо создать полнотекстовый каталог. Начиная с SQL Server 2008 полнотекстовый каталог это логическое понятие, обозначающее группу полнотекстовых индексов.

Полнотекстовый каталог можно создать и в графическом интерфейсе Management Studio. Для этого открываем базу данных, переходим в папку Хранилище ->Полнотекстовые каталоги и выбираем «Создать полнотекстовый каталог». Откроется окно создания, представленное на рисунке 6.1.

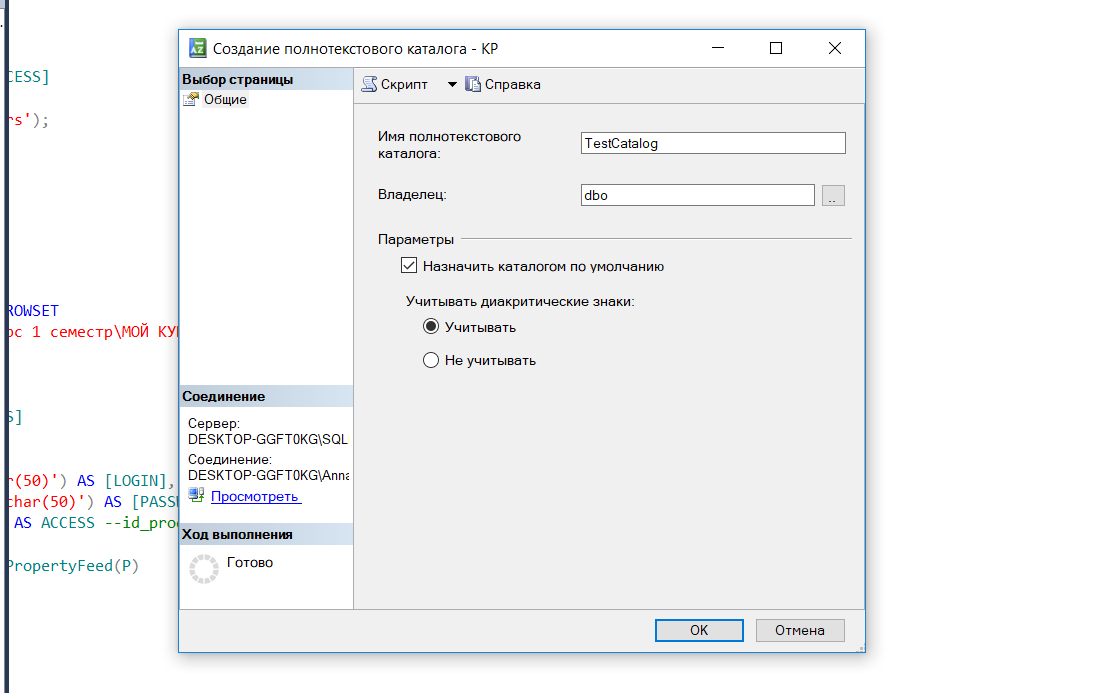


Рисунок 6.1 – окно создания полнотекстового каталога

После создания полнотекстового каталога можно начинать создавать в нем полнотекстовые индекса. В нашем случае мы хотим создать полнотекстовый индекс, в котором участвует поле NAME\_TEST таблицы TEST.

Полнотекстовый индекс можно создать, используя и графические инструменты, для этого открываем свойства полнотекстового каталога и переходим в пункт «Таблицы или представления», выбираем нужную таблицу, поле, уникальный индекс и способ отслеживания изменений.

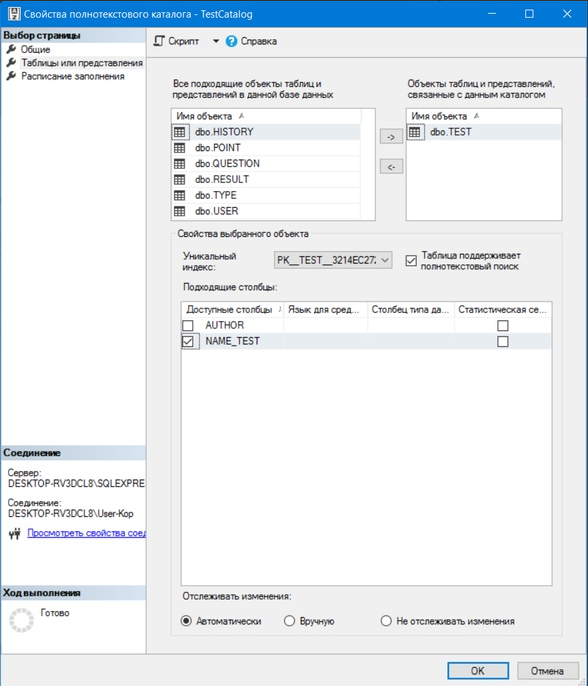


Рисунок 6.2 – создание полнотекстового индекса

Таким образом, мы выполнили создание полнотекстового каталога и индекса.

В качестве примера и подтверждения того, что наш полнотекстовый поиск работает, на рисунке 6.3 представлен пример выполнения полнотекстового запроса.

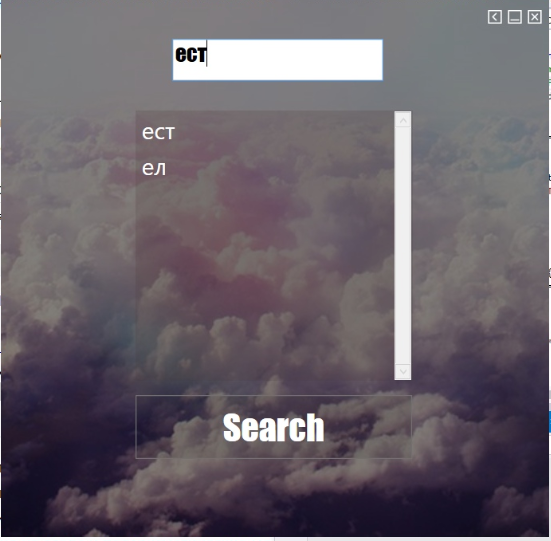


Рисунок 6.3 – пример полнотекстового запроса

Для осуществления запросов могут использоваться различные предикаты и функции. CONTAINS/CONTAINSTABLE и FREETEXT/FREETEXTTABLE полезны для различных типов сопоставления, поиска точных и неточных соответствий отдельных слов и фраз. Полнотекстовые предикаты CONTAINS и FREETEXT используются в предложении WHERE или HAVING инструкции SELECT. С помощью функций CONTAINS, CONTAINSTABLE, FREETEXT или FREETEXTTABLE можно найти все грамматические формы глаголов и существительных. Код запроса представлен на рисунке 6.4.

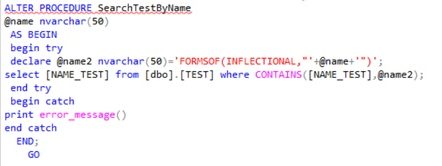


Рисунок 6.4 – пример кода полнотекстового запроса

# Руководство пользователя

Пользовательское приложение предоставляет собой десктоп приложение для взаимодействия с базой данных для пользователей администратор и пользователь. Оно было реализовано с помощью языка C#.

Начальной страницей приложения является окно авторизации, представленное на рисунке 7.1.

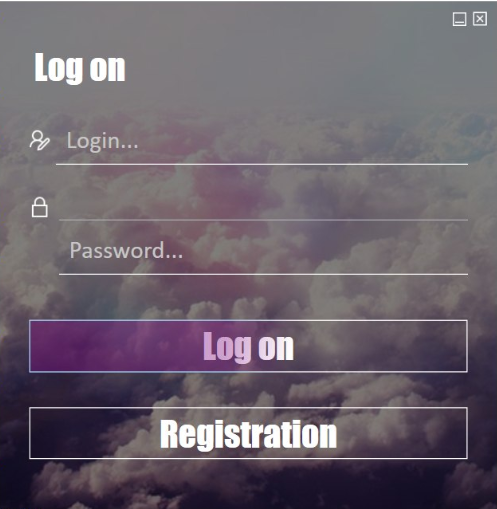


Рисунок 7.1 – Начальная страница

Если пользователь ещё не авторизован, то следует нажать кнопку «Registration» и откроется окно, представленное на рисунке 7.2. После заполнения всех полей и нажатия кнопки «Create» система перекинет пользователя на окно регистрации, представленное на рисунке 7.3.

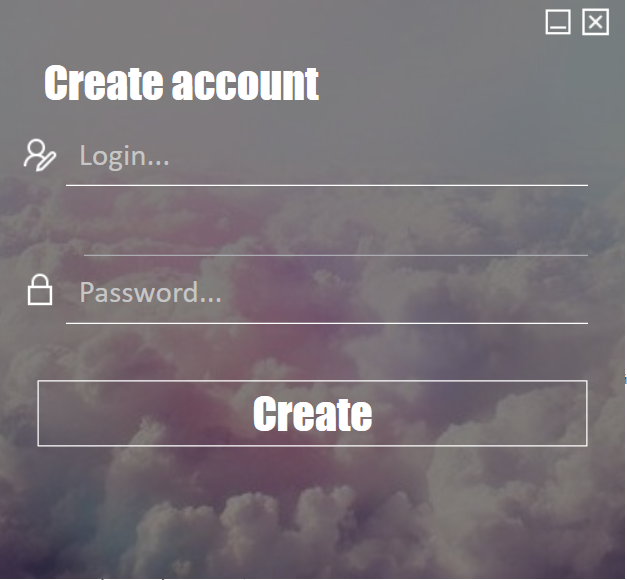


Рисунок 7.2 – Создание аккаунта

Далее, в зависимости от того, кто входит в приложение (пользователь или администратор), открывается соответствующее меню (рисунки 7.3– 7.4).

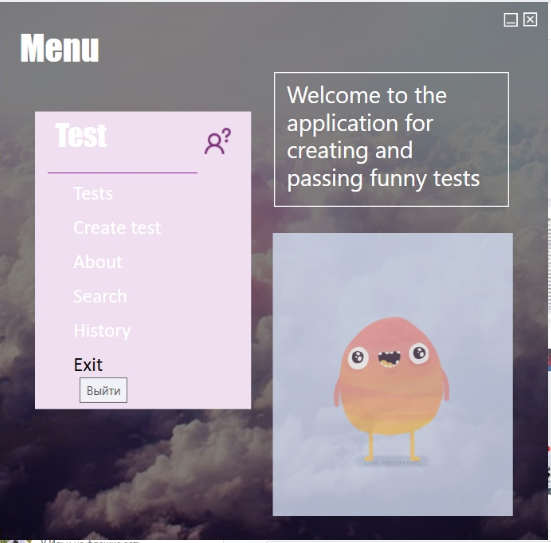


Рисунок 7.3 – Меню пользователя

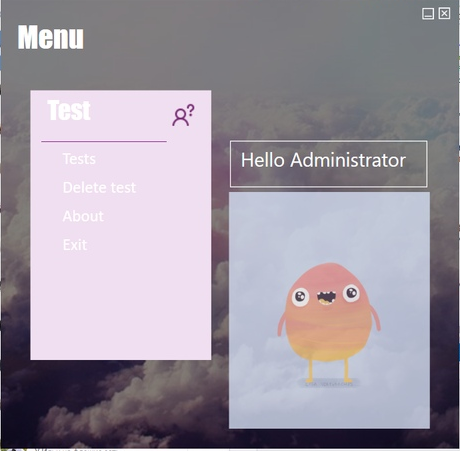


Рисунок 7.4 – Меню администратора

Различие меню состоит в том, что пользователь может создавать, искать и проходить тесты, а также смотреть свою историю. Администратор может удалять и проходить тесты. Далее подробно описано содержание каждого пункта меню.

Рассмотрим меню пользователя. Оно содержит в себе такие вкладки, как: тесты, создание теста, поиск теста, история, о программе и выход.

Для прохождения теста следует перейти на соответствующую вкладку(«Tests») в меню. Откроется окно, представленное на рисунке 7.5.

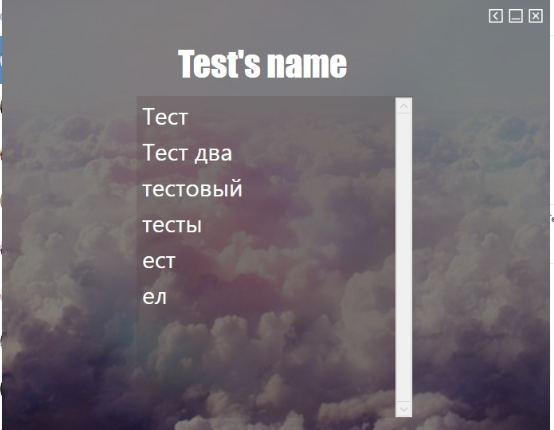


Рисунок 7.5 – Страница с тестами

При выборе одного из тестов откроется само поле теста (рисунок 7.6).

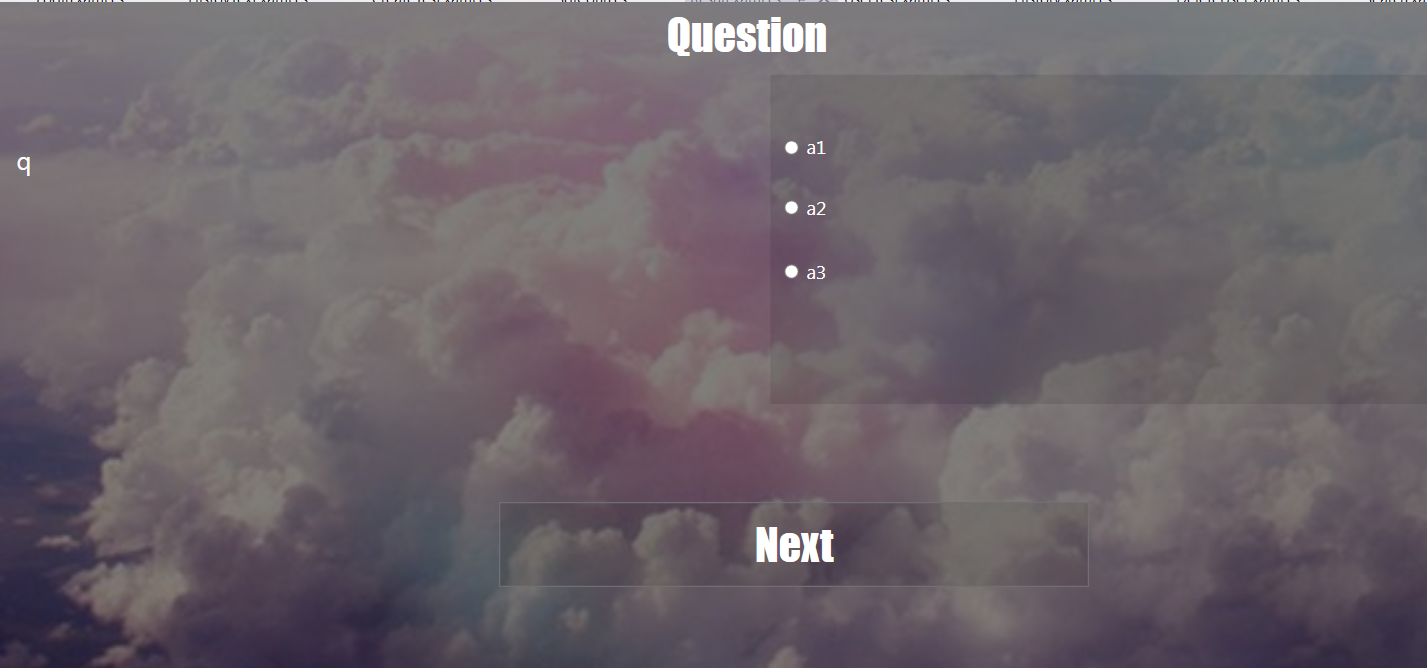


Рисунок 3.6 – Страница прохождения теста

После того, как пользователь ответит на все вопросы, появляется окно с результатом пройденного теста (рисунок 7.8).

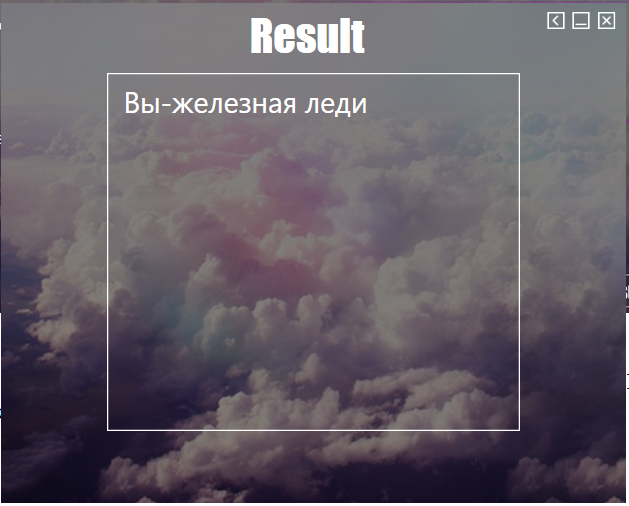


Рисунок 7.8 – Результат теста

Для создания теста необходимо открыть вкладку «Create test» в главном меню. Откроется страница для создания нового теста, представленная на рисунке 7.9.



Рисунок 7.9 – Создание теста

В самом верху есть поле, предназначенное для ввода названия теста. Ниже располагается поле, предназначенное для ввода вопросов, а напротив – по 3 поля для ввода вариантов ответа и баллов, которые даются за каждый ответ.

В конце страницы есть поля для добавления текстового результата и две кнопки. Одна для добавления текстовых результатов(«Add result»), а другая для создания теста(«Create») (Рис. 7.10).

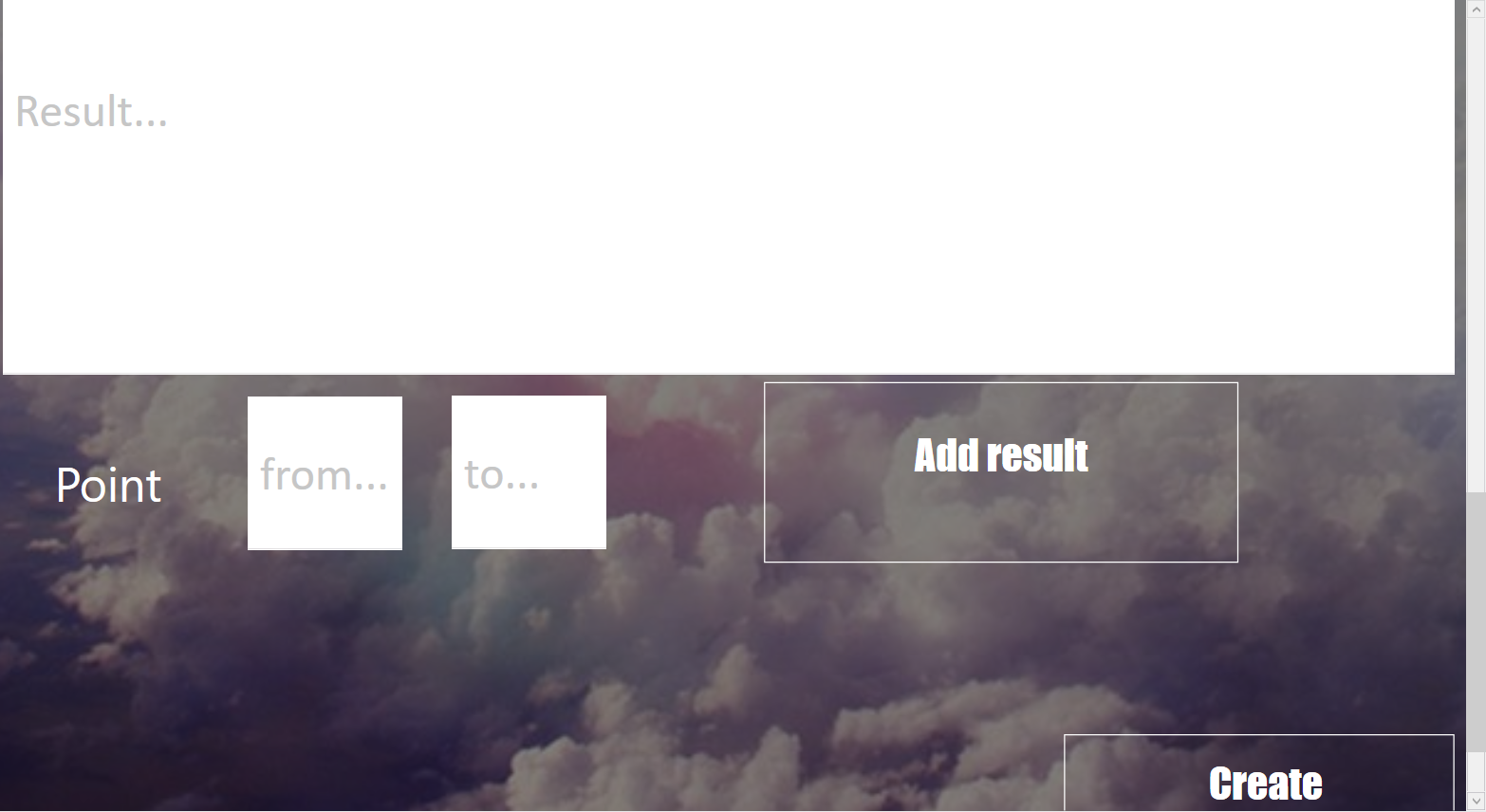


Рисунок 7.10 – Создание и добавление теста

Если пользователь забудет заполнить одно из полей и нажмёт кнопку «Add result» или «Create», система выдаст предупреждение, представленное на рисунке 7.11.

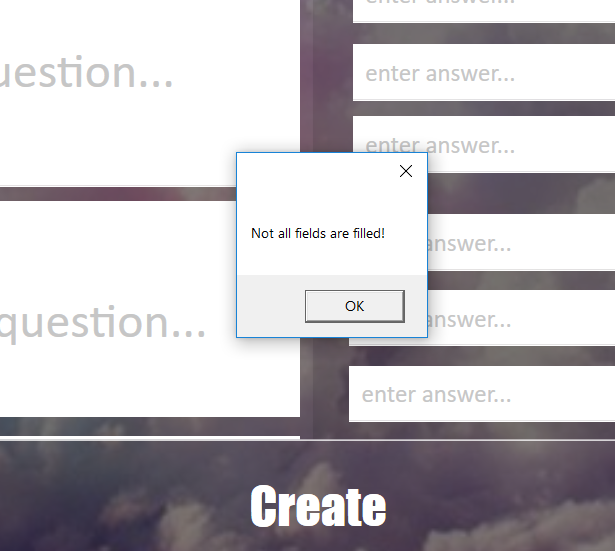


Рисунок 7.11 – Проверка на пустые поля

При выборе пункта «About» главного меню откроется окно, представленное на рисунке 7.12.



Рисунок 7.12 – О программе

На этом обзор пользовательского меню завершён.

При входе в меню администратора добавляется одна новая вкладка «Delete Test». При нажатии на неё откроется окно со списком тестов и кнопкой удалить(«Delete») (рисунок 7.13).

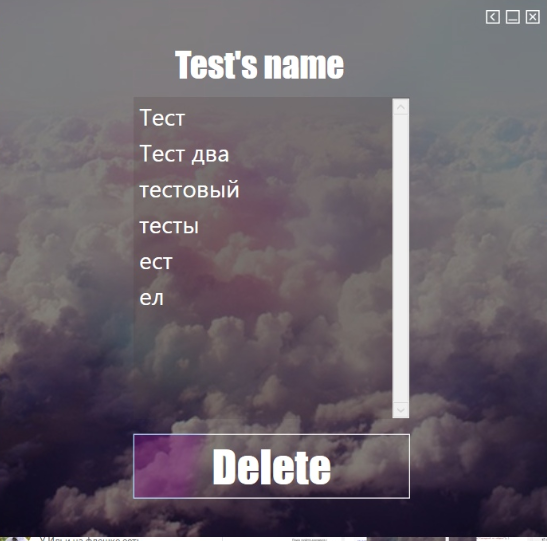


Рисунок 7.13 – Удаление тестов

# Заключение

В ходе выполнения курсовой работы было разработано приложение «Test», являющееся помощником в составлении и прохождении тестов. Во время выполнения данной курсовой работы было изучено немало теоретического материала, а также просмотрено и разобрано большое количество уже готовых решений тех или иных задач.

Разработаны следующие процедуры:

* добавление, изменение и удаление тестов;
* составление истории пользователя;
* поиск тестов;
* импорт и экспорт данных формата xml;

Так же были проведены, проведена процедура резервного копирования и восстановления базы данных.

Разработка приложения­ – трудоемкое занятие, выполнение которого требует всестороннего знания той предметной области, к которой относится тема разрабатываемого приложения. Программная система имеет простой, интуитивно понятный интерфейс пользователя.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

# Список используемых источников

1. Блинова Е.А. Курс лекций по базам данных / Е.А. Блинова

2. MSDN сеть разработчиков в Microsoft [Электронный ресурс] / Режим доступа: <http://msdn.microsoft.com/library/rus/> – Дата доступа: 15.11.2018.

3. Примеры массового импорта и экспорта XML-документов (SQL Server) [Электронный ресурс] – режим доступа - [https://docs.microsoft.com/ru-ru/sql/relational-databases/import-export/examples-of-bulk-import-and-export-of-xml-documents-sql-server](https://docs.microsoft.com/ru-ru/sql/relational-databases/import-export/examples-of-bulk-import-and-export-of-xml-documents-sql-server%20Дата%20доступа%20–%2007.12.017)  Дата доступа 25.11.2018

4. METANIT.COM Сайт о программировании[Электронный ресурс] / Режим доступа: <https://metanit.com> Дата доступа 10.11.2018

5. Запрос с полнотекстовым поиском [Электронный ресурс]: https://docs.microsoft.com/ru-ru/sql/relational-databases/search/query-with-full-text-search?view=sql-server-2017 Дата доступа 26.11.2018.

# Приложение А

|  |
| --- |
| CREATE TABLE [dbo].[USER] (  [LOGIN] NVARCHAR (50) NOT NULL,  [PASSWORD] NVARCHAR (50) NULL,  [ACCESS] INT NULL,  CONSTRAINT [PK\_USER] PRIMARY KEY CLUSTERED ([LOGIN] ASC)  );  CREATE TABLE [dbo].[TEST] (  [ID] INT IDENTITY (1, 1) NOT NULL,  [NAME\_TEST] NVARCHAR (50) NULL,  [AUTHOR] NVARCHAR (50) NULL,  [ID\_TYPE] INT ,  PRIMARY KEY CLUSTERED ([ID] ASC),  FOREIGN KEY ([AUTHOR]) REFERENCES [dbo].[USER] ([LOGIN]),  FOREIGN KEY ([ID\_TYPE]) REFERENCES [dbo].[TYPE] ([ID])  );  CREATE TABLE [dbo].[TYPE] (  [ID] INT IDENTITY (1, 1) NOT NULL,  [NAME\_TYPE] NVARCHAR (50),  PRIMARY KEY CLUSTERED ([ID] ASC)  );  CREATE TABLE [dbo].[QUESTION] (  [ID\_QUESTION] INT IDENTITY (1, 1) NOT NULL,  [ID\_TEST] INT NULL,  [QUESTION] NVARCHAR (50) NULL,  PRIMARY KEY CLUSTERED ([ID\_QUESTION] ASC),  FOREIGN KEY ([ID\_TEST]) REFERENCES [dbo].[TEST] ([ID])  );  CREATE TABLE [dbo].[POINT] (  [ID\_ANSWER] INT IDENTITY (1, 1) NOT NULL,  [ID\_Quest] INT NULL,  [ANSWER] NVARCHAR (50) NULL,  [POINT] INT NULL,  PRIMARY KEY CLUSTERED ([ID\_ANSWER] ASC),  FOREIGN KEY ([ID\_Quest]) REFERENCES [dbo].[QUESTION] ([ID\_QUESTION])  );  CREATE TABLE [dbo].[RESULT] (  [ID\_Result] INT IDENTITY (1, 1) NOT NULL,  [ID\_Test] INT NULL,  [RESULT1] INT NULL,  [RESULT2] INT NULL,  [TEXT\_RESULT] NVARCHAR (50) NULL,  PRIMARY KEY CLUSTERED ([ID\_Result] ASC),  FOREIGN KEY ([ID\_Test]) REFERENCES [dbo].[TEST] ([ID])  );    CREATE TABLE [dbo].[HISTORY] (  [ID] INT IDENTITY (1, 1) NOT NULL,  [ID\_USER] NVARCHAR (50) NOT NULL,  [ID\_RESULTH] INT NULL,  [ID\_TYPE] INT NULL,  PRIMARY KEY CLUSTERED ([ID] ASC),  FOREIGN KEY ([ID\_USER]) REFERENCES [dbo].[USER] ([LOGIN]),  FOREIGN KEY ([ID\_TYPE]) REFERENCES [dbo].[TYPE] ([ID])  ); |

Листинг 1 – Скрипт создания таблиц

# Приложение Б

|  |
| --- |
| use KP;  CREATE LOGIN [AnnaSavenko] WITH PASSWORD = 'anna';  CREATE USER [AnnaSavenko] FOR LOGIN [AnnaSavenko];  grant execute on InsertUser to [AnnaSavenko] with grant option;  grant execute on IdQuestionByIdTest to [AnnaSavenko] with grant option;  grant execute on IdQResultIdTest to [AnnaSavenko] with grant option;  grant execute on ListTestsSelect to [AnnaSavenko] with grant option;  grant execute on TestSelectListByAuthor to [AnnaSavenko] with grant option;  grant execute on GetIdResultByLogin to [AnnaSavenko] with grant option;  grant execute on IdTestByIdRes to [AnnaSavenko] with grant option;  grant execute on GetNameTestByIdTest to [AnnaSavenko] with grant option;  grant execute on GetTestByAuthor to [AnnaSavenko] with grant option;  grant execute on AddNewQuestion to [AnnaSavenko] with grant option;  grant execute on QuestionSelectListByIdTest to [AnnaSavenko] with grant option;  grant execute on AddNewPoint to [AnnaSavenko] with grant option;  grant execute on AnsverSelectListByIdQuest to [AnnaSavenko] with grant option;  grant execute on AddNewHistory to [AnnaSavenko] with grant option;  grant execute on IdTestByIdRes to [AnnaSavenko] with grant option;  grant execute on IdTestByIdRes to [AnnaSavenko] with grant option; |

Листинг 2 – наделение правами пользователей

# Приложение В

|  |
| --- |
| ----------------------[проверка паролей] ------------------------------------  create function [dbo].[AutoValid]( @login nvarchar(50), @password nvarchar(50))  returns int  as begin  Declare @a int =0;  if exists (select [LOGIN] from [dbo].[USER] where [LOGIN]=@login and [PASSWORD]=@password) set @a=1;  else set @a=0;  return @a;  end  GO  ---------------------------------------Take\_access---A------------------------------  create function [dbo].[Take\_access]  (  @login nvarchar(50)  )  returns int  as begin  Declare @id int;  select @id = [ACCESS] from [USER]  where LOGIN=@login  return @id  end  GO  select dbo.Take\_access('user');  ----------------------------------------IfEmpty---A--------------------  create function IfEmpty (@login nvarchar(50))  returns int  as begin  Declare @a int =0;  if exists (select [LOGIN] from [dbo].[USER] where LOGIN=@login) set @a=1;  else set @a=0;  return @a;  END;  go  select \* from [dbo].[USER];  select dbo.auto\_user('anna','1234');  ------------------------------------------Take\_user\_infoByLogin- НЕ ГРАНТИЛА--------------  alter proc Take\_user\_info  @login nvarchar(50)  as begin  begin try  select \* from [dbo].[USER] where LOGIN=@login;  end try  begin catch  print error\_message()  end catch  END;  exec Take\_user\_info @login='admin';  go  ----------------------------------------Update\_user- НЕ ГРАНТИЛА---------------------  alter procedure Update\_user @login nvarchar(50), @password nvarchar(50)  as begin  begin try  UPDATE [dbo].[USER] SET [LOGIN]=@login,[PASSWORD]=@password WHERE [LOGIN]=@login  end try  begin catch  print error\_message()  end catch  END;  go  exec Update\_user @login='qdmin', @password = 'admin';  ------------------------------------------InsertUser-----A---------------------  CREATE PROCEDURE InsertUser  @login nvarchar(50),  @password nvarchar(50),  @access int  AS BEGIN  begin try  INSERT INTO [dbo].[USER] ([LOGIN], [PASSWORD], [ACCESS])  VALUES(@login, @password, @access);  COMMIT  end try  begin catch  print error\_message()  end catch  END;  GO  exec InsertUser  @login = 'admin',  @password = 'admin',  @access = 1;  SELECT \* FROM [dbo].[USER];  ---------------------------------------DeleteUser-----A---------  CREATE PROCEDURE DeleteUser  @login nvarchar(50)  AS BEGIN  begin try  DELETE FROM [dbo].[User] WHERE [LOGIN] = @login;  IF (@@error <> 0)  ROLLBACK  COMMIT;  end try  begin catch  print error\_message()  end catch  END;  GO  exec DeleteUser @login = 'user';  ------------------------------------------UsersSelect-----A------------------  CREATE PROCEDURE UsersSelect  AS BEGIN  begin try  SELECT [LOGIN]  FROM [dbo].[USER] where [ACCESS]=2  COMMIT  end try  begin catch  print error\_message()  end catch  END;  GO  DROP PROCEDURE UsersSelect;  exec UsersSelect;  ----------------------------------------------------TEST---------A-----------------------------  --------------------------------------------Take\_id\_Test\_ByName----------------  Create function Take\_id\_Test  (  @name nvarchar(50)  )  returns int  as begin  Declare @id int;  select @id = [ID] from [dbo].[TEST]  where [NAME\_TEST]=@name  return @id  end  go  ------------------------------------------NameTestEmpty--------------------  create function IfEmptyNameTest (@name nvarchar(50))  returns int  as begin  Declare @a int =0;  if exists (select [NAME\_TEST] from [dbo].[TEST] where NAME\_TEST=@name) set @a=1;  else set @a=0;  return @a;  END;  go  select \* from HISTORY;  select \* from RESULT;  select \* from POINT;  select \* from QUESTION;  select \* from TEST;  select [dbo].IfEmptyNameTest ('T22');  -----------------------------------DeleteTest----A--------------------------  CREATE PROCEDURE DeleteTest  @name nvarchar(50)  AS BEGIN  begin try  DELETE FROM [dbo].[TEST] WHERE [NAME\_TEST] = @name;  IF (@@error <> 0)  ROLLBACK  COMMIT;  end try  begin catch  print error\_message()  end catch  END;  GO  select \* from TEST;  exec DeleteUser @login = 'user';  -----------------------------------DeleteQuestionByTest--------A----------------------  CREATE PROCEDURE DeleteQuestionByTest  @idTest int  AS BEGIN  begin try  DELETE FROM [dbo].[QUESTION] WHERE [ID\_TEST] = @idTest;  IF (@@error <> 0)  ROLLBACK  COMMIT;  end try  begin catch  print error\_message()  end catch  END;  GO  SET @amm = (select ammount from products where id\_product = @id);  if @amm >@am  BEGIN  update products set ammount = ammount - @am where id\_product = @id;  declare @s int;  declare @date date;  declare @p float;  SET @p = (select price from products where id\_product = @id);  SET @s = @am \* @p;  SET @date = GETDATE();  insert into sales(date\_sales,id\_product,ammount,sum\_sales) values  (@date, @id,@am,@s);  END  end try  begin catch  print error\_message()  rollback;  end catch  commit;  END  go  alter procedure filtrProduct(@id int)  as begin  select id\_product, name\_product, price, description\_product,ammount,  (select name\_cat from categories where products.id\_cat = categories.id\_cat),  (select name\_producer from producer where products.id\_producer = producer.id\_producer)  from products where id\_cat = @id;  end;  go  alter procedure getAllEmplLogPass  as begin  select empl\_login, empl\_passw, post from employee;  end    go  alter procedure startDelivery  (  @id int,  @price float,  @idP int,  @log varchar(10),  @am int  )  as  BEGIN  begin tran  declare @idE int;  SET @idE = (select id\_employee from employee where empl\_login = @log);  declare @sD float;  SET @sD = @price \* @am;  declare @nd date;  set @nd = GETDATE();  insert into delivery (id\_producer,id\_employee,id\_product,ammount,sum\_delivery,date\_delivery)  values(@idP,@idE,@id,@am,@sD,@nd);  update products set ammount = ammount + @am where id\_product = @id;  IF (@@error <> 0)  ROLLBACK  commit;  END |

Листинг 3 – процедуры для таблиц

# Приложение Г

|  |
| --- |
| create nonclustered index PswIndex  on [dbo].[USER]([PASSWORD])  GO  create nonclustered index AccessIndex  on [dbo].[USER]([ACCESS])  GO  create nonclustered index LogIndex  on [dbo].[USER]([LOGIN]) |

Листинг 4 – создание индексов для таблиц