**Реферат**

Пояснительная записка курсового проекта содержит 42 страницы, 8 рисунков, 3 источника литературы, 3 приложения.

БИБЛИОТЕКА, БАЗА ДАННЫХ, ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ, SQL, T-SQL, MICROSOFT SQL SERVER.

Основными целями курсового проекта являются: проектирование базы данных «Библиотека».

Пояснительная записка состоит из введения, пяти разделов, заключения.

Во введении представлена общая информация, дающая представление о предстоящей работе, определены цели.

В первом разделе рассматривается архитектура базы данных.

Во втором разделе представлен процесс и результаты разработки объектов базы данных.

В третьем разделе описывается создание процедур для экспорта и импорта.

В четвертом разделе представляется тестирование базы данных.

В пятом разделе представлено руководство пользователя.

В заключении представлены итоги курсового проектирования и задачи, которые были решены в ходе проектирования и разработки базы данных.

Содержание

[Введение 3](#_Toc500432335)

[1. Разработка модели базы данных 4](#_Toc500432336)

[2. Разработка необходимых объектов 5](#_Toc500432337)

[2.1. Таблицы 5](#_Toc500432338)

[2.2. Процедуры 8](#_Toc500432339)

[2.3. Пользователи 9](#_Toc500432340)

[3. Описание технологии оптимизации запросов. 11](#_Toc500432341)

[4. Описание процедур импорта и экспорта 12](#_Toc500432342)

[4.1. Процедура экспорта 12](#_Toc500432343)

[4.2. Процедура импорта 12](#_Toc500432344)

[5. Руководство пользователя 14](#_Toc500432345)

[5.1. Окно авторизации и общедоступной информации 14](#_Toc500432346)

[5.2. Окно пациента 15](#_Toc500432347)

[5.3. Окно доктора 15](#_Toc500432348)

[5.4. Окно регистратора 16](#_Toc500432349)

[6. Разграничение прав 17](#_Toc500432355)

[Заключение 18](#_Toc500432356)

[Список использованных источников 19](#_Toc500432357)

[ПРИЛОЖЕНИЕ А 20](#_Toc500432358)

[ПРИЛОЖЕНИЕ Б 22](#_Toc500432359)

[ПРИЛОЖЕНИЕ В 26](#_Toc500432360)

[ПРИЛОЖЕНИЕ Г. 32](#_Toc500432361)

# Введение

В настоящее время сложно понять причины не повсеместного перехода на цифровой учёт и обработку информации, так как компьютеры достигли больших вычислительных мощностей, при достаточно низких ценах на их. Также существует на данный момент много бесплатных решений, удовлетворяющим высоким условиям качества и позволяющее достаточно легко его использовать, хранения данных. В данном курсовом проекте предоставляется пример применения информационных технологий в сфере медицинского обслуживания.

В соответствии с заданием курсового проектирования следует создать базу данных (БД) для учреждений здравоохранения с использование программного продукта Microsoft SQL Server 2017. Так же долже создать клиентское приложение, представляющее собой интерфейс для взаимодействия с базой данных. Для его работы необходим интерпретатор Python 3.6.3 (Cython) и доступные ему библиотеки: pymssql, PyQt5.

Сегодня трудно себе представить сколько-нибудь значимую информационную систему, которая не имела бы в качестве основы или важной составляющей базу данных. Концепции и технологии БД складывались постепенно и всегда были тесно связаны с развитием систем автоматизированной обработки информации.

# Разработка модели базы данных

Для создания, хранения и управления данными, была использованная реляционная СУБД Microsoft SQL Server. Ниже на рисунке 1.1 представлена модель базы данных (таблицы, отмеченные на них первичных и вторичных ключей) курсового проекта. База данных была подвергнута процедуре нормализации данных.

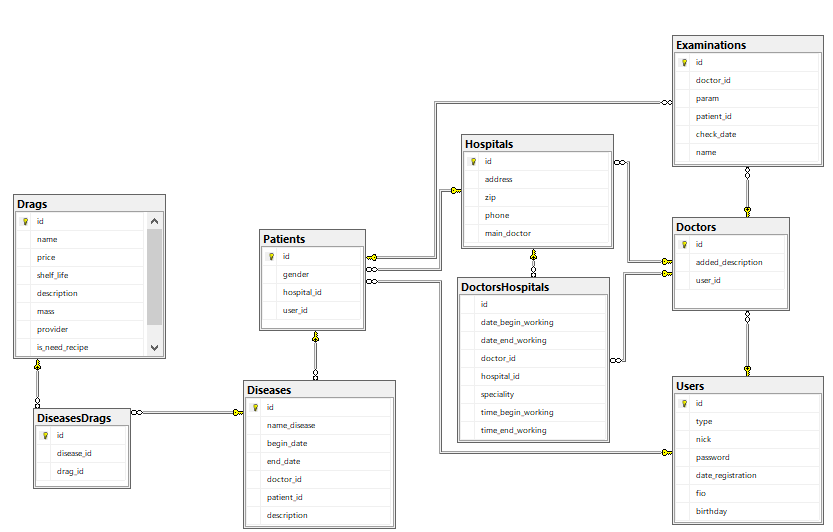


Рисунок 1.1 – Диаграмма базы данных

Более подробное описание назначения таблиц базы данных представлено ниже в разделе 2.1.

# 2. Разработка необходимых объектов

База данных данного курсового проекта содержит следующие объекты: таблиц, хранимых процедур, пользователей. Их более подробное описание представлено в этой главе ниже.

## 2.1. Таблицы

Данная база данных содержит 9 таблиц.

Таблица Users *–* хранит основные данные о пользователях приложения. В ней хранится логин, пароль, тип пользователя, дата регистрации, ФИО пользователя и дата рождения. Пользователи бывают 3 типов. Графическое описание представлено на рисунке 2.1 ниже.

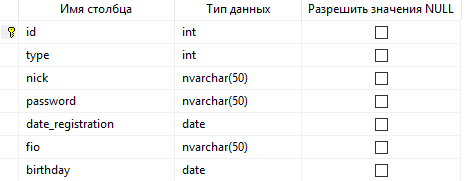


Рисунок 2.1. Таблица пользователей

Таблица Doctors *–* содержит информацию о докторах. Хранит описание о враче и вторичный ключ связанный с id пользователя. Графическое описание представлено на рисунке 2.2 ниже.

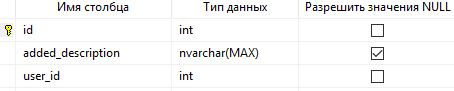


Рисунок 2.2. Таблица врачей

Таблица Hospitals *–* содержит информацию о больницах. В ней хранится адрес больницы, почтовый индекс больницы, телефон, вторичный ключ связанный с id доктора. Графическое описание представлено на рисунке 2.3 ниже.

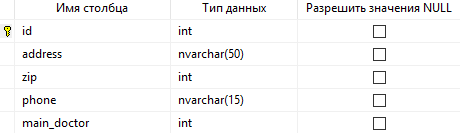


Рисунок 2.3. Таблица больниц

Таблица Patients *–* содержит информацию о пациентах. Хранит информацию о гендерной принадлежности, вторичный ключ связанный с id больницы, вторичный ключ связанный с id пользователя. Графическое описание представлено на рисунке 2.4 ниже.

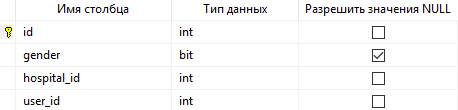


Рисунок 2.4. Таблица пользователей

Таблица Examinations *–* содержит информацию об обследованиях пациента. Хранит информацию о имени обследования, дата проведения обследования, параметры обследования (в виде xml), вторичный ключ связанный с id пациента, вторичный ключ связанный с доктором. Графическое описание представлено на рисунке 2.5 ниже.

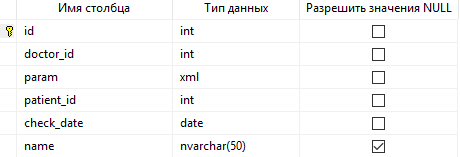


Рисунок 2.5. Таблица обследований

Таблица Diseases *­­–* содержит информацию о заболеваниях пациента. Хранит имя заболевания, дата начала заболевания, дата конца заболевания, его тестовое описание, вторичный ключ связанный с id пациента, вторичный ключ связанный с id доктора. Графическое описание представлено на рисунке 2.6 ниже.

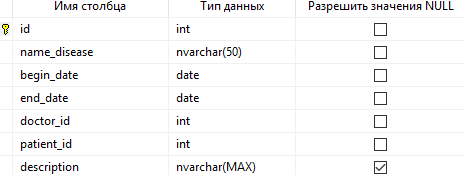


Рисунок 2.6. Таблица заболеваний

Таблиц Drags *–* содержит информацию о лекарствах. Хранит имя лекарства, цену, срок годности, описание, массу, имя поставщика и информацию о необходимости ресурса. Графическое описание представлено на рисунке 2.7 ниже.

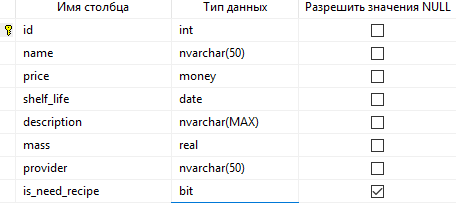


Рисунок 2.7. Таблица лекарств

Таблица DoctorsHospitals *–* содержит информацию о докторах, нанятых в больницах. Хранит информацию о дате начала работы, дате конца работы, название специальности, время начала смены, время конца смены, вторичный ключ для связи с id больницы, вторичный ключ для связи с id доктора. Графическое описание представлено на рисунке 2.8 ниже.

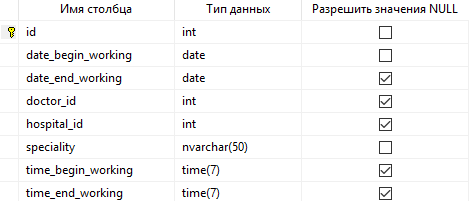


Рисунок 2.8. Таблица нанятых врачей

Таблица DiseasesDrags *–* содержит информацию о лекарствах, назначенных пациентам. Хранит вторичный ключ связанный с id болезни, вторичный ключ для связи с id лекарства. Графическое описание представлено на рисунке 2.9 ниже.

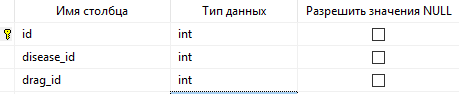


Рисунок 2.9. Таблица назначенных лекарств

T-sql код для создания таблиц представлен в приложении А.

## 2.2. Процедуры

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 хранимых некоторых процедур, обеспечивающих косвенный и строго регламентированный доступ к данным.

Все процедуры курсовой работы можно условно поделить на два типа: возвращающие результат выборки из таблицы (рисунок 2.10) или выполняющие изменение данных в таблицах (использующие dml операторы), которые возвращают «0» при успешном завершении операции или код ошибки (рисунок 2.11).

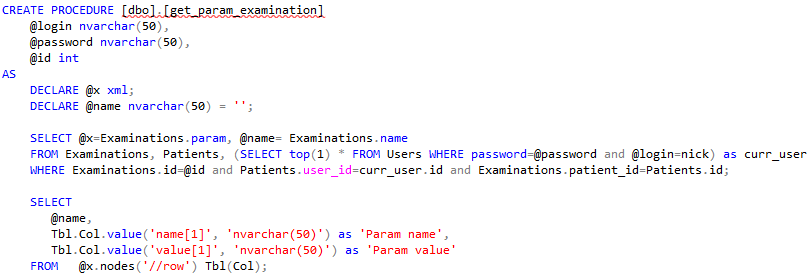


Рисунок 2.10. Процедура получения параметров обследования

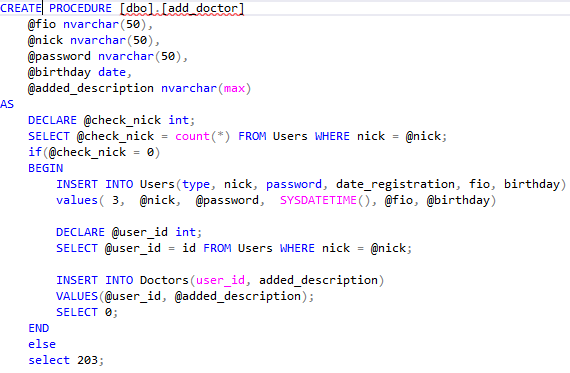


Рисунок 2.11. Процедура добавления доктора

Все исходные коды процедур и их краткое описание находятся в приложении Б.

## 2.3. 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 4 пользователя (authorization, patient, doctor, regisrator). У каждого пользователь имеет свои права, предназначенных для выполнения соответствующих ему требованиям.

Пользователь authorization предназначен для проверки на валидность данных при попытке входа под другим пользователем, и выдачей общедоступной информации. Доступны следующие процедуры:

– dbo.auth\_user;

– dbo.get\_drag\_by\_id;

– dbo.get\_drag\_by\_name;

– dbo.get\_time\_work\_by\_zip;

– dbo.get\_time\_work\_by\_fio;

– dbo.get\_time\_work\_by\_zip\_fio.

Пользователь patient предназначен для получения информации о собственной истории болезней пациента и медицинских обследованиях. Доступны следующие процедуры:

– dbo.auth\_user;

– dbo.get\_inverse\_patient\_diseases;

– dbo.get\_drags\_by\_disease;

– dbo.get\_patient\_examinations;

– dbo.get\_inverse\_patient\_examinations;

– dbo.get\_patient\_diseases.

Пользователь doctor предназначен для получения информации о истории болезней пациентов и медицинских обследованиях, а также имеет возможность назначать лекарства, создавать записи об болезнях и обследованиях. Доступны следующие процедуры:

– dbo.auth\_user;

– dbo.select\_disease;

– dbo.add\_disease;

– dbo.change\_disease;

– dbo.appoint\_drag;

– dbo.add\_examination;

– dbo.add\_param;

– dbo.get\_doctor\_examination\_param;

– dbo.get\_doctor*\_*examinations;

– dbo.get\_exclude\_doctor\_examinations;

– dbo.get\_drags\_by\_disease\_doctor;

– dbo.get\_statistic\_by\_all\_time;

– dbo.get\_statistic\_by\_index;

– dbo.get\_statistic\_by\_year;

– dbo.get\_statistic\_by\_index\_year.

Пользователь regisrator предназначен для создания пациентов, врачей, привязка врачей к больнице и изменение основной больницы пациента. Доступны следующие процедуры:

– dbo.auth\_user;

– dbo.add\_patient;

– dbo.add\_doctor;

– dbo.take\_doctor;

– dbo.del\_doctor;

– dbo.change\_patient;

– dbo.import\_hospital;

– dbo.export\_hospital.

# 3. Резервное копирование базы данных.

Во многих современных СУБД присутствует возможность создания плана резервного копирования данных для обеспечения их целостности и актуальности БД в случаях технических сбоев. В данном курсовом проекте было использована функция создания полной копии базы данных и затем проведена настройка плана создания разностных копий с частотой раз в 1 неделю. Ниже более подробно описаны действия по созданию и восстановлению из резервных копий.

## 3.1 Создание резервной копии

Для создания резервной копии с помощью Microsoft sql server management studio необходимо выбрать базу данных в обозревателе объектов, для которой создаём backup, кликнуть правой кнопкой мыши базу данных, выберите пункт «задачи», а затем команду «Создать резервную копию». Откроется диалоговое окно, представленное на рисунке 3.1.

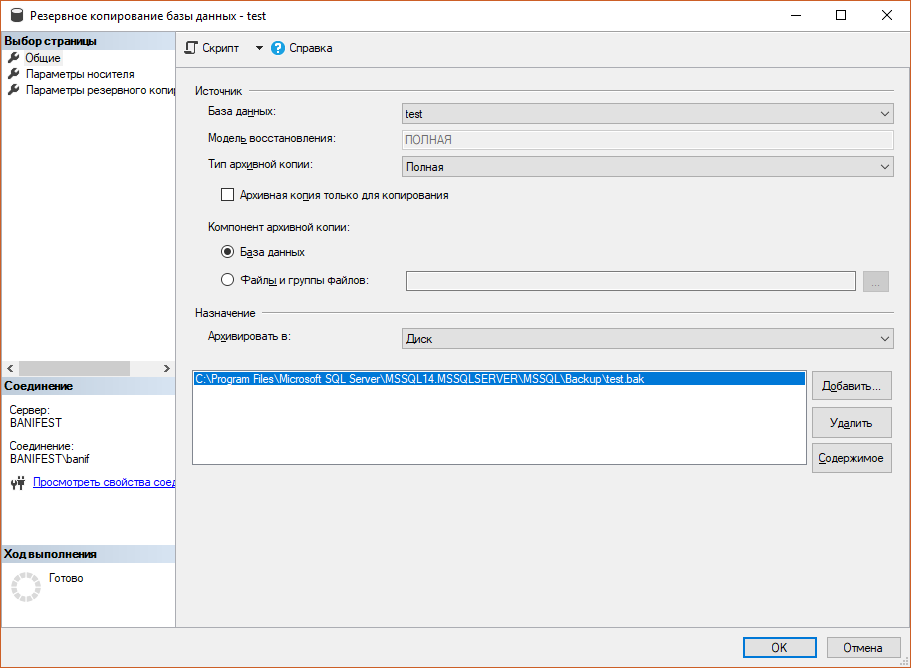
.

Рисунок 3.1. Окно создания резервной копии

Мы увидим окно создания и настройки резервной копии. В нем мы имеем возможность установить тип архивной копии, что именно из базы данных копировать, место расположения бедующей копии. У резервной копии можно указать её имя, описание, её срок действия, параметры сжатия и методы шифрования. Так как копии базы данных курсового проекта хранятся на том же носителе информации, шифрование не применялось. Было создано 2 архивирования: полная (исходные значения БД) и разностная с частотой обновления 7 дней.

## 3.2 Восстановление из резервной копии

Для восстановления резервной копии с помощью Microsoft sql server management studio необходимо выбрать базу данных в обозревателе объектов, которую пытаемся восстановить из архивной копии, кликнуть правой кнопкой мыши базу данных, выберите пункт «задачи», а затем команду «Восстановить», а потом, «базу данных». Откроется диалоговое окно, представленное на рисунке 3.2.

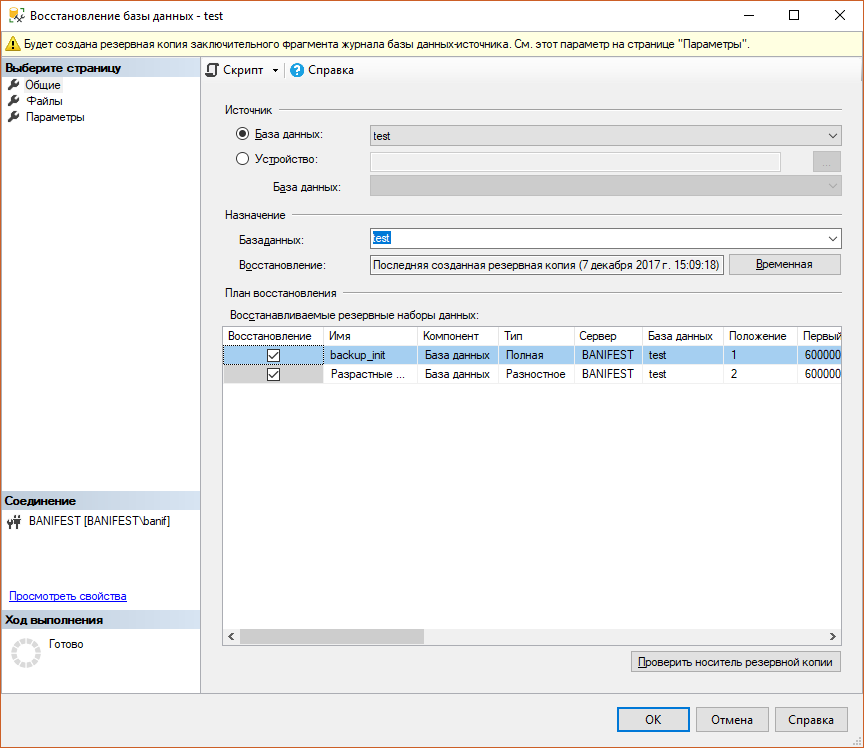


Рисунок 3.2. Окно восстановления из резервной копии

В данном окне мы можем выбрать к какой копии мы хотим откатить базу данных, информацию о самих архивных копий и параметры восстановления (перезаписать, добавить к существующим записям и др.).

## 3.3 Описание типов используемых резервных копий

Разностное резервное копирование основано на самой последней предыдущей полной резервной копии данных. В разностной резервной копии сохраняются только те изменения, которые были произведены с момента создания последней полной резервной копии.

На рисунке 3.4 показано, как работает разностное резервное копирование. В базе данных содержится 24 экстента данных, 6 из которых изменены. Разностная резервная копия содержит только эти шесть экстентов данных. Разностное резервное копирование зависит от страницы битовой карты, которая содержит один бит для каждого экстента. Для каждого экстента, обновленного с момента создания основы для разностной копии, в битовой карте биту присваивается значение 1.

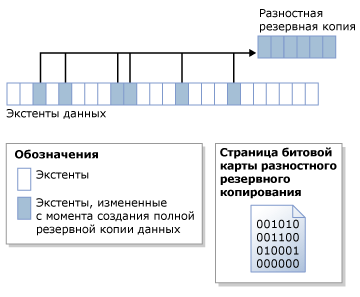


Рисунок 3.3. Пример выбора данных для разностной копии

Разностная копия имеет ряд преимуществ над полной копией:

* создание разностных резервных копий выполняется гораздо быстрее по сравнению с созданием полной резервной копии;
* разностная резервная копия баз данных сохраняет только те данные, которые изменились по сравнению с полной резервной копией, которая служила основой для разностной резервной копии:
* при наличии часто изменяемого участка данных по сравнению с другими, разностная копия позволяет существенно уменьшить издержки архивирования;
* в рамках модели полного восстановления использование разностных резервных копий позволяет сократить число резервных копий журналов, которые требуется восстановить.

Самый ощутимый недостаток разностных копий — при потери промежуточной копии, все следующие за ней разностные копии становятся некорректными.

При полном резервном копировании создается резервная копия всей базы данных целиком. В нее входит часть журнала транзакций, что позволяет восстановить полную базу данных из полной резервной копии базы данных. Полные резервные копии базы данных отображают состояние базы данных на момент завершения резервного копирования.

# 4. Описание процедур импорта и экспорта

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

## 4.1. Процедура экспорта

Код создания процедуры экспорта данных таблицы в xml формат представлен на рисунке 3.1.

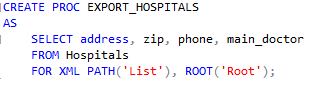


Рисунок 3.1 – Процедура экспорта

Приложение вызывает вышеуказанную процедуру и принимает результат запроса, которым является строка в формате xml. Далее приложение вызывает диалоговое файловое окно, которое производит сохранение документа.

## 4.2. Процедура импорта

Код создания процедуры импорта представлен на рисунке 3.2.

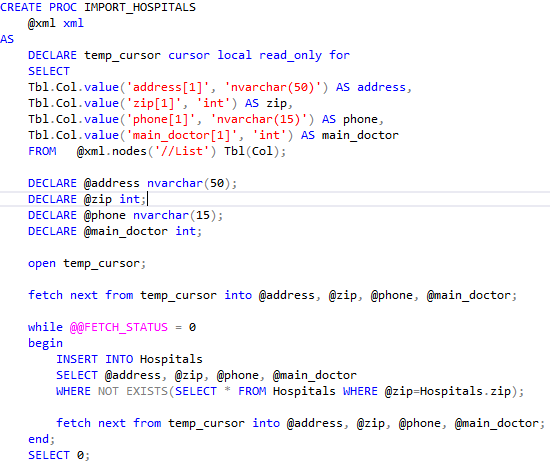


Рисунок 3.2 – Процедура экспорта

Процедура принимает со стороны клиентского приложения xml код и дополняет таблицу всеми записями, у которых индекс не совпадает с уже имеющимися записями.

# 5. Описание технологии оптимизации запросов.

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.

## 5.1 Описание типов индексов

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.

В реляционной СУБД оптимальный план выполнения запроса — это такая последовательность применения операторов реляционной алгебры к исходным и промежуточным отношениям, которое для конкретного текущего состояния БД (её структуры и наполнения) может быть выполнено с минимальным использованием вычислительных ресурсов.

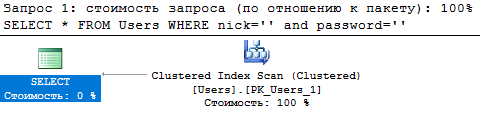
При создании индекса указывается один или несколько столбцов таблицы, по значениям которых будет построен и поддерживаться индекс. Индекс представляет собой структуру памяти, организованную в виде сбалансированного дерева. В узлах дерева содержатся страницы со значениями из выбранных столбцов. SELECT-запросы, использующие в секции WHERE столбцы таблицы, для которых построен индекс, не требуют сканирования всей таблицы, так как индекс позволяет получить указатели на все запрашиваемые строки за небольшое количество операций чтения. Кроме того, индексы, как правило, имеют размеры значительно меньшие, чем таблицы, что в большинстве случаев позволяет их размещать в оперативной памяти сервера. Использование индексов является прозрачным для программиста, пишущего запросы. Он не имеет возможности указать серверу, какой индекс следует использовать в запросе. Вопрос о применении индекса для выборки данных решается специальной компонентой сервера, называемой оптимизатор запросов. Программист может лишь выяснить порядок выборки данных, получив план исполнения запроса.

## 5.2 Описание перестройки и реорганизации индексов

Если после построения индекса на таблице будет проведено много dml операций, то индекс может значительно потерять свою эффективность в оптимизации запроса, это явление называется фрагментацией индекса. В этом случаи следует проводить реорганизацию или перестройку индексов. Перестройка уничтожает старый и заново создаёт новый индекс, а реорганизация изменяет старый.

## 5.3 Пример использованиях индексов в базе данных

В базе данных на 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.



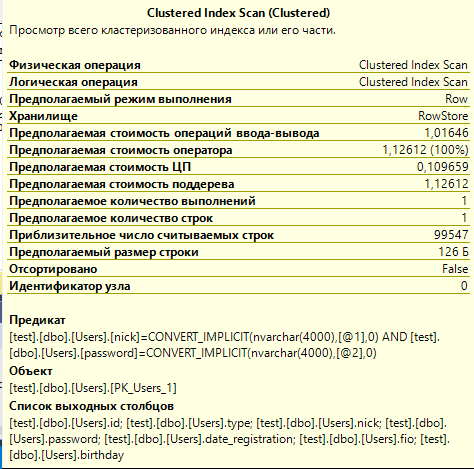
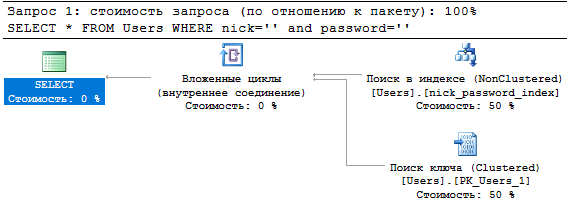


Рисунок 5.1. Карта запроса без неклстеризованного индекса

Как можно увидеть стоимость запроса достаточно велика по сравнению с таблицей, уже содержащей некластеризованный индекс, охватывающий столбцы nick и password на рисунках 5.2.



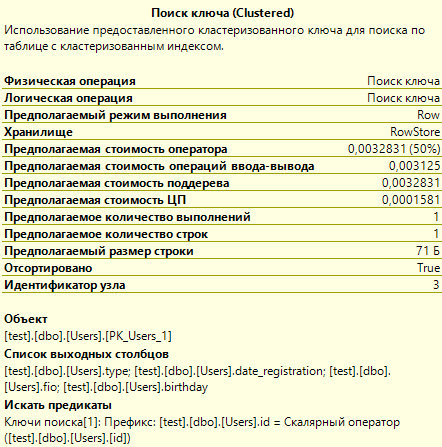
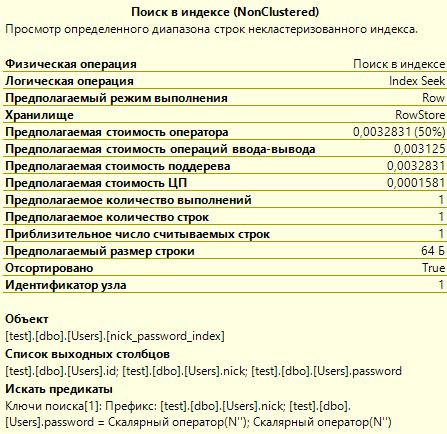


Рисунок 5.2. Карта запроса с некластеризованным индексом

Как можно заметить то сумма запроса стоимость запроса уменьшилась более в 100 раз. Как показал этот демонстрационный запрос построение индексов для таблиц весьма эффективна для операций выборки. В приложении Г представлены коды создания индексов.

# 5. Руководство пользователя

Пользовательское приложение предоставляет собой интерфейс для взаимодействия с базой данных для пользователей всех типов, описанных в части 2.3. Оно было реализовано с помощью языка Python 3.6.3 (Cython). Для корректной работы так же необходимы следующие библиотеки доступные интерпретатору: pymssql, PyQt5.

## 5.1. Окно авторизации и общедоступной информации

При запуске приложения пользователю станет доступно окно авторизации. При вводе корректных логина и пароля пользователь получит доступ к расширенным функциям, а это окно закроется. При закрытии любого нового окна, заново откроется окно авторизации.

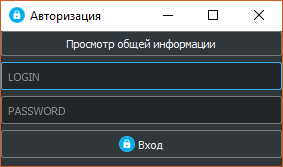


Рисунок 5.1. Окно авторизации

Так же пользователь может просмотреть общедоступную информацию кликнув по верхней кнопке в окне, представленном на рисунке 5.1. При нажатии откроется новое окно, представленное на рисунке 5.2

.

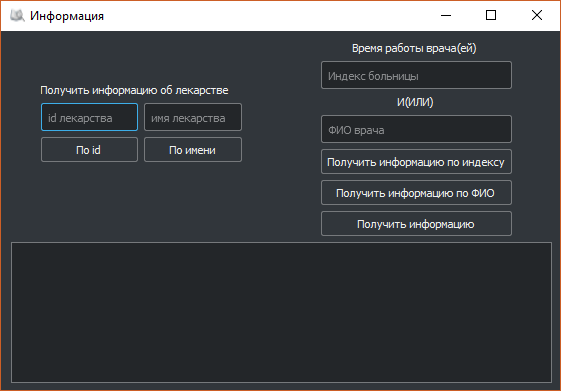


Рис. 5.2. Окно общедоступной информации

В вышеуказанном окне можно просмотреть информацию о лекарствах и время работы врача(ей) по заданным пользователем параметрам.

## 5.2. Окно пациента

Если пользователь производит вход под данными, соответствующими пациенту, то ему открывается окно, представленное на рисунке 5.3.

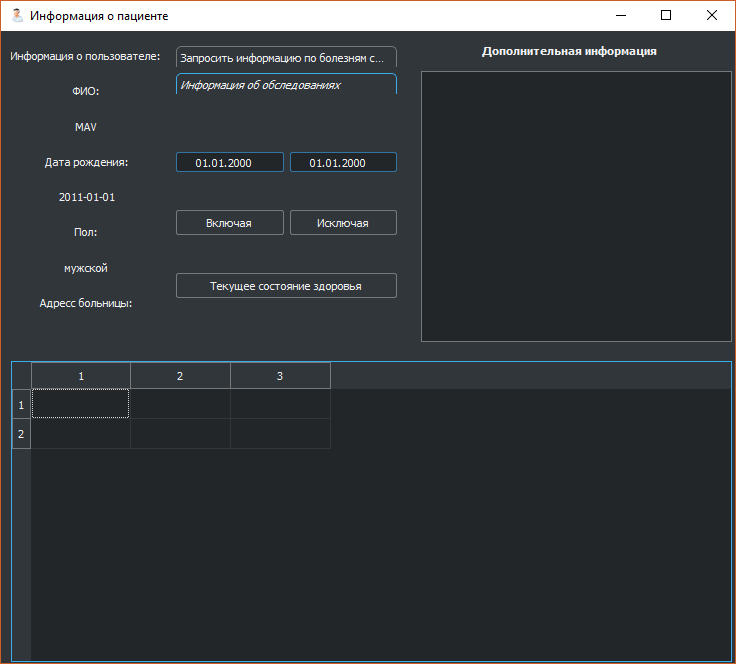


Рисунок 5.3. Окно пациента

В данном окне пользователь может просмотреть информацию о собственных болезнях, назначенных лекарствах, информацию об обследованиях и текущее состояние здоровья согласно последним данных обследований.

## 5.3. Окно доктора

Если пользователь производит вход под данными, соответствующими врачу, то ему открывается окно, представленное на рисунке 5.4.

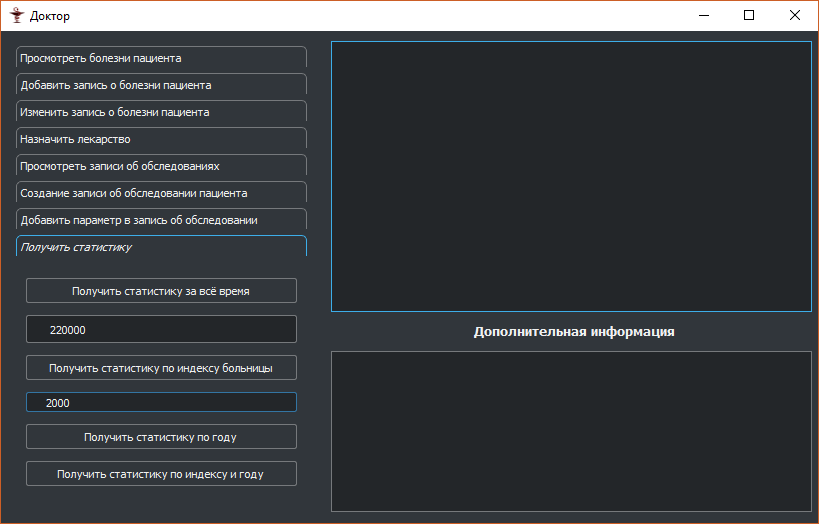


Рисунок 5.4. Окно врача

Врач имеет права смотреть информацию о болезнях и обследованиях пациентах, которые проводил сам врач по логину. Также способен назначить лекарство, констатировать заболевание, провести обследование, добавить параметр к обследованию и получить разнообразную статистику по заболеваниям (больницы и времени).

## 5.4. Окно регистратора

Если пользователь производит вход под данными, соответствующими регистратору, то ему открывается окно, представленное на рисунке 5.5.

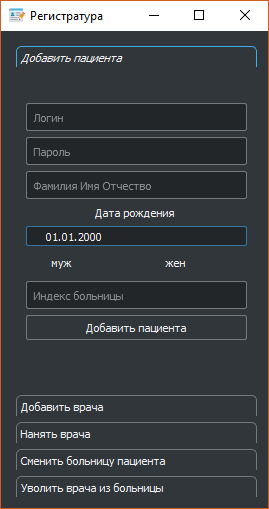


Рисунок 5.5. Окно регистратора

Регистратор имеет возможность создавать пациентов и врачей. Нанимать и увольнять врача на работу в больницу и менять привязку пациента к больнице.

# Заключение

В данном курсовом проекте была разработана база данных для учреждений здравоохранения. Программа автоматизирует труд врача при заполнении документов, обеспечивает более быстрый доступ к информации о пациентах и организует надежное хранение и обработку данных.

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

Выполненные цели:

* выдача актуального состояния здоровья пациента;
* перенос и синхронизация истории болезней от одного учреждения здравоохранения к другому;
* выдача статистики заболеваний;
* удалённое назначение лечения;
* поддержка различных типов пользователей.

# Список использованных источников

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

2. Оптимизация запросов производительности [Электронный ресурс] msdn.microsoft.com – Режим доступа: <https://technet.microsoft.com/ru>-ru/library/ms172984(v=sql.110).aspx Дата доступа 07.12.2017.

3. Benjamin Nevarez. Inside the SQL Server Query Optimizer / – First published by Simple Talk Publishing 2010 – 252 с.

4. Оптимизация SQL запросов [Электронный ресурс] <http://ts-soft.ru/blog/sql-optimization-1>. Дата доступа 07.12.2017.

5. Оптимизация SQL запросов [Электронный ресурс] [http://ts-soft.ru/blog/sql-optimization-2. Дата доступа 07.12.2017](http://ts-soft.ru/blog/sql-optimization-2.%20Дата%20доступа%2007.12.2017).

6. Разностные резервные копии (SQL Server) [Электронный ресурс] <https://docs.microsoft.com/ru-ru/sql/relational-databases/backup-restore/differential-backups-sql-server>. Дата доступа 08.12.2017.

# ПРИЛОЖЕНИЕ А

**SQL – скрипты созданных таблиц**

CREATE TABLE [dbo].[Users](

[id] [int] IDENTITY(1,1) primary key NOT NULL,

[type] [int] NOT NULL,

[nick] [nvarchar](50) NOT NULL,

[password] [nvarchar](50) NOT NULL,

[date\_registration] [date] NOT NULL,

[fio] [nvarchar](50) NOT NULL,

[birthday] [date] NOT NULL)

CREATE TABLE [dbo].[Doctors](

[id] [int] IDENTITY(1,1) primary key NOT NULL,

[added\_description] [nvarchar](max) NULL,

[user\_id] [int] NOT NULL,

CONSTRAINT [FK\_Doctors\_Users] FOREIGN KEY([user\_id])

REFERENCES [dbo].[Users] ([id]))

CREATE TABLE [dbo].[Patients](

[id] [int] IDENTITY(1,1) primary key NOT NULL,

[gender] [bit] NULL,

[hospital\_id] [int] NOT NULL,

[user\_id] [int] NOT NULL,

CONSTRAINT [FK\_Patient\_Hospital] FOREIGN KEY([hospital\_id])

REFERENCES [dbo].[Hospitals] ([id]))

CREATE TABLE [dbo].[Hospitals](

[id] [int] IDENTITY(1,1) primary key NOT NULL,

[address] [nvarchar](50) NOT NULL,

[zip] [int] NOT NULL,

[phone] [nvarchar](15) NOT NULL,

[main\_doctor] [int] NOT NULL,

CONSTRAINT [FK\_Hospitals\_Doctors] FOREIGN KEY([main\_doctor])

REFERENCES [dbo].[Doctors] ([id]))

CREATE TABLE [dbo].[Drags](

[id] [int] IDENTITY(1,1) primary key NOT NULL,

[name] [nvarchar](50) NOT NULL,

[price] [money] NOT NULL,

[shelf\_life] [date] NOT NULL,

[description] [nvarchar](max) NOT NULL,

[mass] [real] NOT NULL,

[provider] [nvarchar](50) NOT NULL,

[is\_need\_recipe] [bit] NULL)

CREATE TABLE [dbo].[Examinations](

[id] [int] IDENTITY(1,1) primary key NOT NULL,

[doctor\_id] [int] NOT NULL,

[param] [xml] NOT NULL,

[patient\_id] [int] NOT NULL,

[check\_date] [date] NOT NULL,

[name] [nvarchar](50) NULL,

CONSTRAINT [FK\_Examinations\_Doctors] FOREIGN KEY([doctor\_id])

REFERENCES [dbo].[Doctors] ([id]),

CONSTRAINT [FK\_Examinations\_Patients] FOREIGN KEY([patient\_id])

REFERENCES [dbo].[Patients] ([id])) );

CREATE TABLE [dbo].[Diseases](

[id] [int] IDENTITY(1,1) primary key NOT NULL,

[name\_disease] [nvarchar](50) NOT NULL,

[begin\_date] [date] NOT NULL,

[end\_date] [date] NOT NULL,

[doctor\_id] [int] NOT NULL,

[patient\_id] [int] NOT NULL,

[description] [nvarchar](max) NULL,

CONSTRAINT [FK\_Diseases\_Patients] FOREIGN KEY([patient\_id])

REFERENCES [dbo].[Patients] ([id]))

CREATE TABLE [dbo].[DiseasesDrags](

[id] [int] IDENTITY(1,1) primary key NOT NULL,

[disease\_id] [int] NOT NULL,

[drag\_id] [int] NOT NULL,

CONSTRAINT [disease] FOREIGN KEY([disease\_id])

REFERENCES [dbo].[Diseases] ([id]),

CONSTRAINT [drag] FOREIGN KEY([drag\_id])

REFERENCES [dbo].[Drags] ([id]))

CREATE TABLE [dbo].[DoctorsHospitals](

[id] [int] IDENTITY(1,1) primary key NOT NULL,

[doctor\_id] [int] NULL,

[hospital\_id] [int] NULL,

[speciality] [nvarchar](50) NOT NULL,

[time\_begin\_working] [time](7) NULL,

[time\_end\_working] [time](7) NULL,

[date\_begin\_working] [date] NULL,

[date\_end\_working] [date] NULL,

CONSTRAINT [FK\_DoctorsHospitals\_Doctors] FOREIGN KEY([doctor\_id])

REFERENCES [dbo].[Doctors] ([id]),

CONSTRAINT [FK\_DoctorsHospitals\_Hospitals] FOREIGN KEY([hospital\_id]) REFERENCES [dbo].[Hospitals] ([id]))

# ПРИЛОЖЕНИЕ Б

**Процедуры и их краткие описания**

Ниже на листингах данного приложения представлены скрипты создания процедур, используемых в курсовом проекте.

CREATE PROCEDURE [dbo].[add\_disease]

@login nvarchar(50),

@doc\_login nvarchar(50),

@doc\_password nvarchar(50),

@name nvarchar(50),

@date\_begin date,

@date\_end date,

@desc nvarchar(MAX)

AS

DECLARE @patient\_id int = 0;

DECLARE @doctor\_id int = 0;

SELECT top(1) @patient\_id = Patients.id

FROM Patients, (SELECT \* FROM Users WHERE nick = @login) as curr\_user

WHERE Patients.user\_id = curr\_user.id;

SELECT top(1) @doctor\_id = Doctors.id

FROM Doctors, (SELECT \* FROM Users WHERE nick = @doc\_login and @doc\_password=password) as curr\_user

WHERE Doctors.user\_id = curr\_user.id;

if(@patient\_id != 0 and @doctor\_id != 0)

BEGIN

INSERT INTO Diseases

VALUES (@name, @date\_begin, @date\_end, @doctor\_id, @patient\_id, @desc)

SELECT 0;

END

else

select 212;

Листинг Б.1. Процедура добавления записи о болезни пациента

CREATE PROCEDURE [dbo].[add\_doctor]

@fio nvarchar(50),

@nick nvarchar(50),

@password nvarchar(50),

@birthday date,

@added\_description nvarchar(max)

AS

DECLARE @check\_nick int;

SELECT @check\_nick = count(\*) FROM Users WHERE nick = @nick;

if(@check\_nick = 0)

BEGIN

INSERT INTO Users(type, nick, password, date\_registration, fio, birthday)

values( 3, @nick, @password, SYSDATETIME(), @fio, @birthday)

DECLARE @user\_id int;

SELECT @user\_id = id FROM Users WHERE nick = @nick;

INSERT INTO Doctors(user\_id, added\_description)

VALUES(@user\_id, @added\_description);

SELECT 0;

END

Else select 203;

Листинг Б.2. Процедура добавления записи о болезни пациента

# ПРИЛОЖЕНИЕ В

**Наиболее важные функции**

**Клиент**

private void FillAllCars()

{

this.dataGridView1.Rows.Clear();

this.dataGridView1.Columns.Clear();

using (SqlConnection connect = new SqlConnection(connectString))

{

string query = "all\_cars";

connect.Open();

SqlCommand command = new SqlCommand(query, connect);

command.CommandType = CommandType.StoredProcedure;

SqlParameter logparam = new SqlParameter {

ParameterName = "@pagenumb",

Value = page,

SqlDbType = SqlDbType.Int };

command.Parameters.Add(logparam);

SqlParameter logparam1 = new SqlParameter {

ParameterName = "@rowsatpage",

Value = rowsAtPage,

SqlDbType = SqlDbType.Int };

command.Parameters.Add(logparam1);

var result = command.ExecuteReader();

if (result != null)

{

this.dataGridView1.Columns.Add("id", "ID");

this.dataGridView1.Columns["id"].Width = 100;

this.dataGridView1.Columns.Add("car", "Auto");

this.dataGridView1.Columns["car"].Width = 100;

this.dataGridView1.Columns.Add("model", "Model");

this.dataGridView1.Columns["model"].Width = 100;

this.dataGridView1.Columns.Add("color", "Color");

this.dataGridView1.Columns["color"].Width = 100;

this.dataGridView1.Columns.Add("country", "Country");

this.dataGridView1.Columns["country"].Width = 100;

this.dataGridView1.Columns.Add("price", "Price");

this.dataGridView1.Columns["price"].Width = 100;

while (result.Read()) {

this.dataGridView1.Rows.Add(result[0].ToString(), result[1].ToString(), result[2].ToString(), result[3].ToString(), result[4].ToString(), result[6].ToString());

} } else

MessageBox.Show("Error!");

}

}

private void поСтранеToolStripMenuItem\_Click(object sender, EventArgs e)

{

temp te = new temp();

DialogResult res = te.ShowDialog();

if (res != DialogResult.OK)

MessageBox.Show("Введите страну еще раз.");

this.dataGridView1.Rows.Clear();

this.dataGridView1.Columns.Clear();

using (SqlConnection connect = new SqlConnection(connectString))

{

string query = "country\_cars";

connect.Open();

SqlCommand command = new SqlCommand(query, connect);

command.CommandType = CommandType.StoredProcedure;

SqlParameter logparam = new SqlParameter

{

ParameterName = "@country",

Value = Program.q,

SqlDbType = SqlDbType.VarChar

};

command.Parameters.Add(logparam);

var result = command.ExecuteReader();

if (result != null)

{

this.dataGridView1.Columns.Add("id", "ID");

this.dataGridView1.Columns["id"].Width = 100;

this.dataGridView1.Columns.Add("car", "Auto");

this.dataGridView1.Columns["car"].Width = 100;

this.dataGridView1.Columns.Add("model", "Model");

this.dataGridView1.Columns["model"].Width = 100;

this.dataGridView1.Columns.Add("country", "Country");

this.dataGridView1.Columns["country"].Width = 100;

this.dataGridView1.Columns.Add("color", "Color");

this.dataGridView1.Columns["color"].Width = 100;

this.dataGridView1.Columns.Add("price", "Price");

this.dataGridView1.Columns["price"].Width = 100;

while (result.Read())

{

this.dataGridView1.Rows.Add(result[0].ToString(), result[1].ToString(), result[2].ToString(), result[3].ToString(), result[4].ToString(), result[5].ToString());

}

}

else

MessageBox.Show("Error!"); }

}

private void всеАвтоToolStripMenuItem\_Click(object sender, EventArgs e)

{

FillAllCars();

string query = "count\_cars";

using (SqlConnection connect = new SqlConnection(connectString))

{

connect.Open();

SqlCommand command = new SqlCommand(query, connect);

command.CommandType = CommandType.StoredProcedure;

var result = command.ExecuteScalar();

count = (int)result;

MessageBox.Show(count.ToString());

}

}

private void button1\_Click(object sender, EventArgs e) {

if (count != 0)

{

page++;

FillAllCars();

}

}

private void button2\_Click(object sender, EventArgs e) {

if (count != 0)

{

page--;

FillAllCars();

}

}

private void button3\_Click(object sender, EventArgs e) {

if (count != 0)

{

page = 1;

FillAllCars();

}

}

private void button4\_Click(object sender, EventArgs e) {

if (count != 0) {

page = count / rowsAtPage;

if (count % rowsAtPage > 0) // if remainder is more than zero

{

page += 1;

}

FillAllCars(); } }

private void всеТипыКузововToolStripMenuItem\_Click(object sender, EventArgs e)

{

this.dataGridView1.Rows.Clear();

this.dataGridView1.Columns.Clear();

string query = "all\_types";

using (SqlConnection connect = new SqlConnection(connectString))

{

connect.Open();

SqlCommand command = new SqlCommand(query, connect);

command.CommandType = CommandType.StoredProcedure;

var result = command.ExecuteReader();

if (result != null)

{

this.dataGridView1.Columns.Add("types", "Types");

this.dataGridView1.Columns["types"].Width = 100;

while (result.Read())

{

this.dataGridView1.Rows.Add(result[0].ToString());

}

}

}

}

**Менеджер**

private void FillAllActs()

{

this.dataGridView1.Rows.Clear();

this.dataGridView1.Columns.Clear();

using (SqlConnection connect = new SqlConnection(connectString))

{

string query = "all\_acts";

connect.Open();

SqlCommand command = new SqlCommand(query, connect);

command.CommandType = CommandType.StoredProcedure;

SqlParameter logparam = new SqlParameter

{

ParameterName = "@pagenumb",

Value = page,

SqlDbType = SqlDbType.Int

};

command.Parameters.Add(logparam);

SqlParameter logparam1 = new SqlParameter

{

ParameterName = "@rowsatpage",

Value = rowsAtPage,

SqlDbType = SqlDbType.Int

};

command.Parameters.Add(logparam1);

var result = command.ExecuteReader();

if (result != null)

{

this.dataGridView1.Columns.Add("id", "ID");

this.dataGridView1.Columns["id"].Width = 100;

this.dataGridView1.Columns.Add("car", "CarID");

this.dataGridView1.Columns["car"].Width = 100;

this.dataGridView1.Columns.Add("client", "ClientID");

this.dataGridView1.Columns["client"].Width = 100;

this.dataGridView1.Columns.Add("date", "Date");

this.dataGridView1.Columns["date"].Width = 100;

this.dataGridView1.Columns.Add("pay", "Pay Type");

this.dataGridView1.Columns["pay"].Width = 100;

while (result.Read())

{

this.dataGridView1.Rows.Add(result[0].ToString(), result[1].ToString(), result[2].ToString(), result[3].ToString(), result[4].ToString());

}

}

else

MessageBox.Show("Error!");

}

}

private void всеКлиентыToolStripMenuItem\_Click(object sender, EventArgs e) {

type = "client";

FillAllClients();

string query = "count\_clients";

using (SqlConnection connect = new SqlConnection(connectString)) {

connect.Open();

SqlCommand command = new SqlCommand(query, connect);

command.CommandType = CommandType.StoredProcedure;

var result = command.ExecuteScalar();

count = (int)result;

MessageBox.Show(count.ToString()); }

}

private void всеМаркиToolStripMenuItem\_Click(object sender, EventArgs e)

{

this.dataGridView1.Rows.Clear();

this.dataGridView1.Columns.Clear();

string query = "all\_mark";

using (SqlConnection connect = new SqlConnection(connectString))

{

connect.Open();

SqlCommand command = new SqlCommand(query, connect);

command.CommandType = CommandType.StoredProcedure;

var result = command.ExecuteReader();

if (result != null)

{

this.dataGridView1.Columns.Add("titl", "Title");

this.dataGridView1.Columns["titl"].Width = 100;

this.dataGridView1.Columns.Add("country", "Country");

this.dataGridView1.Columns["country"].Width = 100;

while (result.Read())

{

this.dataGridView1.Rows.Add(result[0].ToString(),result[1].ToString());

}

}

}

}

# ПРИЛОЖЕНИЕ Г.

Генератор данных

Random rnd = new Random();

string st = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";

List<Client> techs = new List<Client>();

for (int i = 1; i <= 100000; i++)

{

string fio = "";

int n = rnd.Next(0, 26);

fio = st.Substring(n, 1);

for (int j = 0; j < 7; j++)

{

n = rnd.Next(27, 53);

string s1 = st.Substring(n, 1);

fio += s1;

}

fio += " ";

n = rnd.Next(0, 23);

fio += st.Substring(n, 1);

fio += ". ";

n = rnd.Next(0, 23);

fio += st.Substring(n, 1);

string passp = "";

for (int j = 0; j < 2; j++)

{

n = rnd.Next(0, 26);

string s1 = st.Substring(n, 1);

passp += s1;

}

for (int j = 0; j < 7; j++)

{

n = rnd.Next(54, 61);

string s1 = st.Substring(n, 1);

passp += s1;

}

string address = "";

n = rnd.Next(0, 23);

address += st.Substring(n, 1);

for (int j = 0; j < 9; j++)

{

n = rnd.Next(27, 53);

string s1 = st.Substring(n, 1);

address += s1;

}

address += ", st.";

string phone = "";

for (int j = 0; j < 9; j++)

{

n = rnd.Next(54, 61);

string s1 = st.Substring(n, 1);

phone += s1;

}

string login = "";

for (int j = 0; j < 10; j++)

{

n = rnd.Next(0, 61);

string s1 = st.Substring(n, 1);

login += s1;

}

string pass = "";

for (int j = 0; j < 10; j++)

{

n = rnd.Next(0, 61);

string s1 = st.Substring(n, 1);

pass += s1;

}

Client tec = new Client(i, fio, passp, address, phone, login, CaclMD5.CalculateMD5Hash(pass));

techs.Add(tec);

}