**Реферат**

Пояснительная записка курсового проекта содержит 43 страницы, 25 рисунков, 31 листинга, 5 источников литературы, 4 приложений.

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

Основными целями курсового проекта являются: проектирование базы данных для учреждения здравоохранения.

Пояснительная записка состоит из введения, шести 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 и восстановление базы данных из резервной копии.

В четвёртом разделе описывается создание процедур для экспорта и импорта.

В пятом разделе описана технология применения оптимизации запросов.

В шестом разделе представлено руководство пользователя.

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

Содержание

[Введение 6](#_Toc500461335)

[1. Разработка модели базы данных 7](#_Toc500461336)

[2. Разработка необходимых объектов 8](#_Toc500461337)

[2.1. Таблицы 8](#_Toc500461338)

[2.2. Процедуры 11](#_Toc500461339)

[2.3. Пользователи 12](#_Toc500461340)

[3. Резервное копирование базы данных. 14](#_Toc500461341)

[3.1 Создание резервной копии 14](#_Toc500461342)

[3.2 Восстановление из резервной копии 15](#_Toc500461343)

[3.3 Описание типов используемых резервных копий 15](#_Toc500461344)

[4. Описание процедур импорта и экспорта 17](#_Toc500461345)

[4.1. Процедура экспорта 17](#_Toc500461346)

[4.2. Процедура импорта 17](#_Toc500461347)

[5. Описание технологии оптимизации запросов. 19](#_Toc500461348)

[5.1 Описание типов индексов 19](#_Toc500461349)

[5.2 Описание перестройки и реорганизации индексов 19](#_Toc500461350)

[5.3 Пример использованиях индексов в базе данных 20](#_Toc500461351)

[6. Руководство пользователя 22](#_Toc500461352)

[6.1. Окно авторизации и общедоступной информации 22](#_Toc500461353)

[5.2. Окно пациента 23](#_Toc500461354)

[5.3. Окно доктора 23](#_Toc500461355)

[5.4. Окно регистратора 24](#_Toc500461356)

[Заключение 25](#_Toc500461357)

[Список использованных источников 26](#_Toc500461358)

[ПРИЛОЖЕНИЕ А 27](#_Toc500461359)

[ПРИЛОЖЕНИЕ Б 29](#_Toc500461360)

[ПРИЛОЖЕНИЕ В 42](#_Toc500461361)

[ПРИЛОЖЕНИЕ Г 43](#_Toc500461362)

# 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.

Данная тема была выбрана в связи с тем, что сегодня нет приложений, решающих проблему удалённого консультирования больных. А просмотр информации о болезнях требует поездки в учреждение здравоохранения. Так же не так редки случаи потери информации о пациенте, в связи с большим документооборотом. И поиск и удобство оставляют желать лучшего.

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.

# 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.

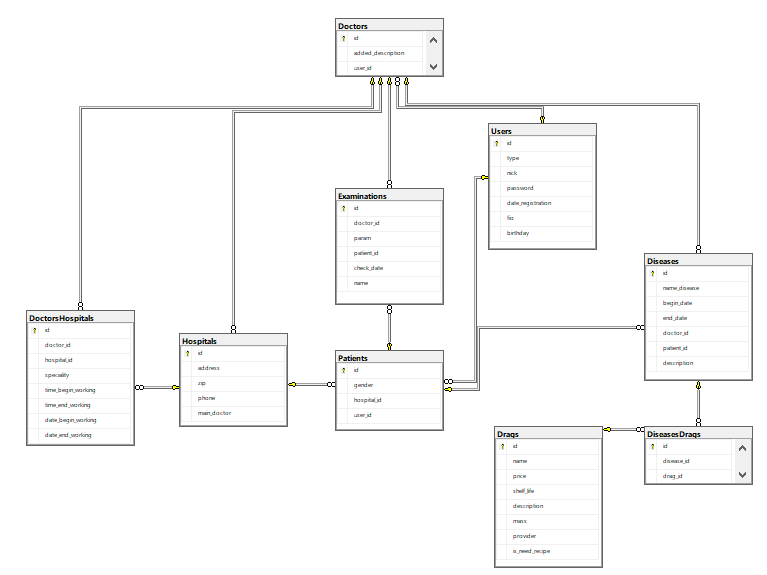


Рисунок 1.1 – Диаграмма модели базы данных

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

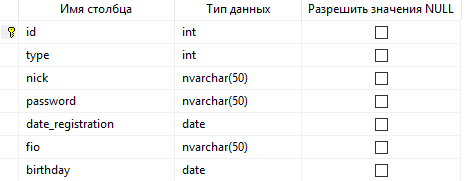


Рисунок 2.1. Таблица пользователей

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.

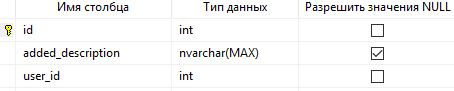


Рисунок 2.2. Таблица врачей

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.

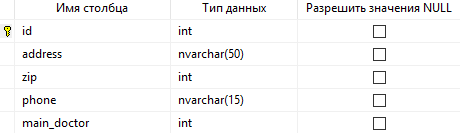


Рисунок 2.3. Таблица больниц

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.

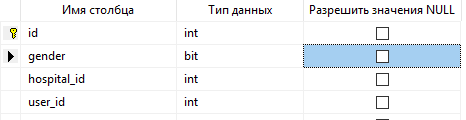


Рисунок 2.4. Таблица пациентов

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.

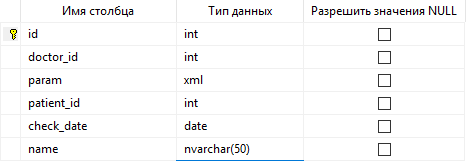


Рисунок 2.5. Таблица обследований

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.

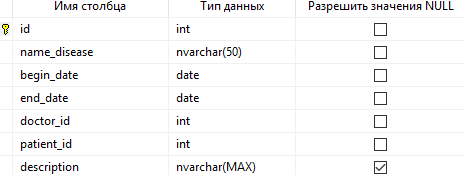


Рисунок 2.6. Таблица заболеваний

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.

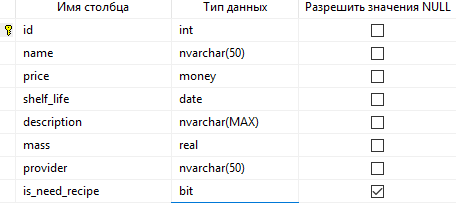


Рисунок 2.7. Таблица лекарств

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.

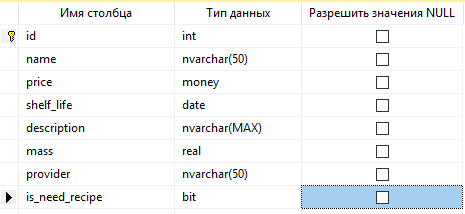


Рисунок 2.8. Таблица нанятых врачей

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.

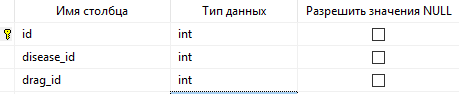


Рисунок 2.9. Таблица назначенных лекарств

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).

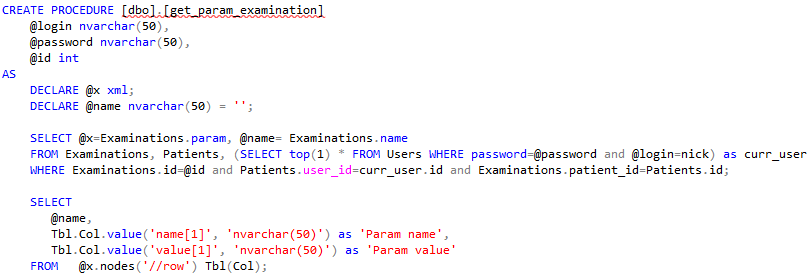


Рисунок 2.10. Процедура получения параметров обследования

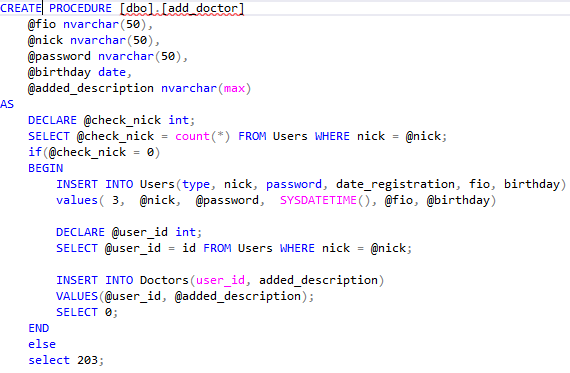


Рисунок 2.11. Процедура добавления доктора

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,

– dbo.get\_get\_patient.

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.

В приложение Г содержаться скрипты наделения привилегиями перечисленным выше пользователям.

# 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.

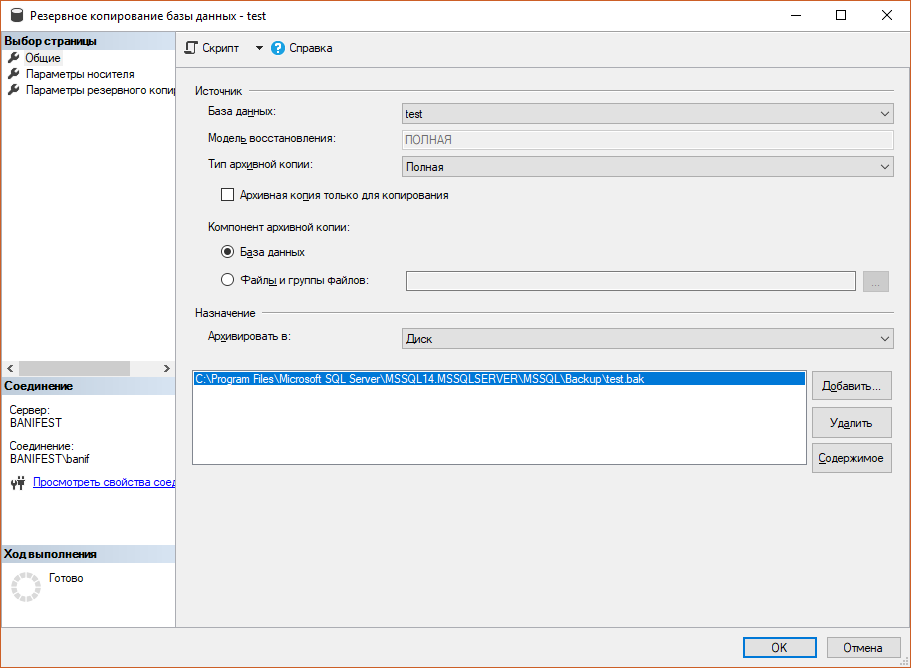
.

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.

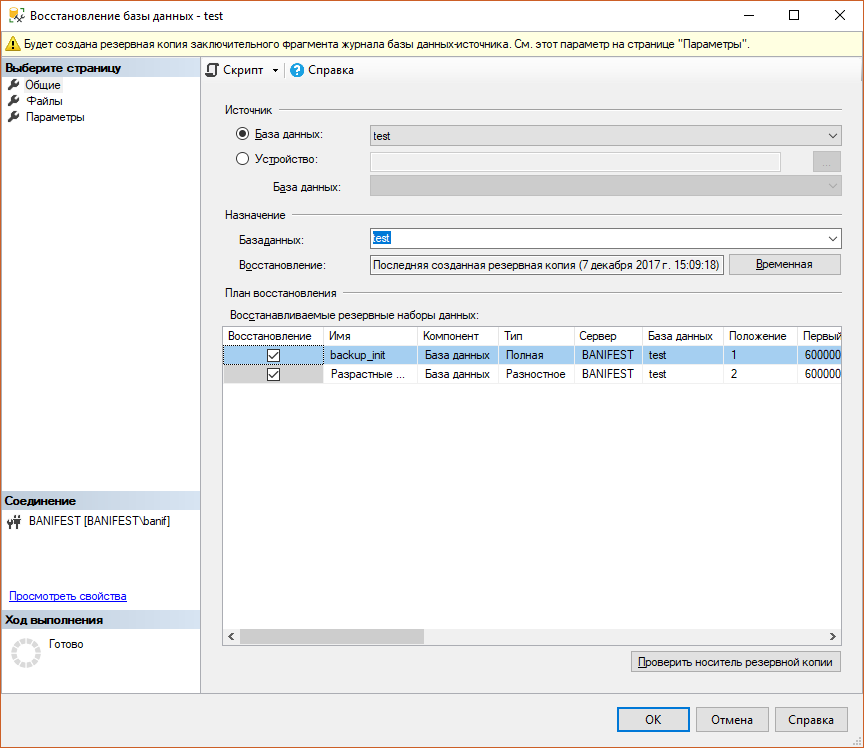


Рисунок 3.2. Окно восстановления из резервной копии

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.

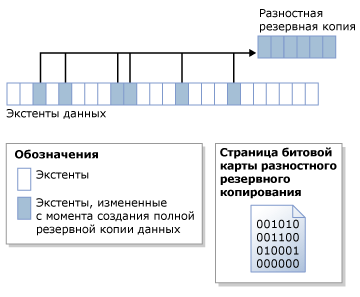


Рисунок 3.3. Пример выбора данных для разностной копии

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 4.1.

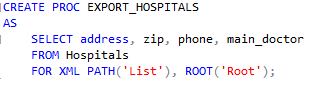


Рисунок 4.1 – Процедура экспорта

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 4.2.

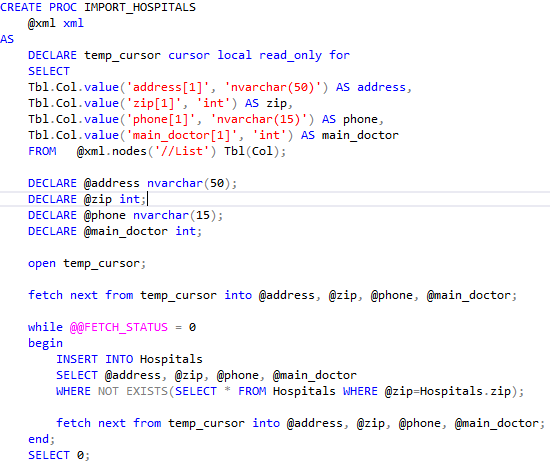


Рисунок 4.2 – Процедура экспорта

Приложение открывает окно файлового диалога, и считывает xml файл. Процедура принимает со стороны клиентского приложения xml код и дополняет таблицу всеми записями, у которых индекс больницы 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. Программист имеет возможности указать серверу, какой индекс следует использовать в запросе с помощью подсказок (hint), однако сервер почти во всех случаях выбирает оптимальны план запроса, так что применять их следует только в очень специфических случаях. 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.

В создании композитного (сложного) индекса участвуют несколько полей таблицы. При создании индекса следует обращать внимание на порядок следования полей в индексе. Например, если создается индекс по полям Field1, Field2, то он может быть применен только в запросе, где в критериях используются оба этих поля. Также этот индекс будет полезен для условий, построенных для одного Field1. Для одного Field2 этот индекс не может быть применен.

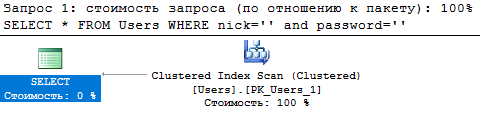
Если в дополнение к индексу по полям Field1, Field2 добавить индекс по полям Field2, Field1, то SQL Server при построении плана запроса будет анализировать, какой из них более продуктивен в применении к ограничениям на условия запроса. Последний момент в построении композитного индекса по полям Field1, Field2 – он не равен сумме индексов по указанным полям. В случае, когда в запросе могут быть использованы оба поля как критерий поиска при раздельных индексах по полям, будет построено пересечение по индексам, что медленнее чем выборка из композитного индекса.

## 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 полностью уничтожает старый и заново создаёт индекс, а реорганизация изменяет предыдущий. При сильном фрагментации индекса индекс перестанет участвовать в запросах, из-за потери своей полезности.

## 5.3 Пример использования индексов в базе данных

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.



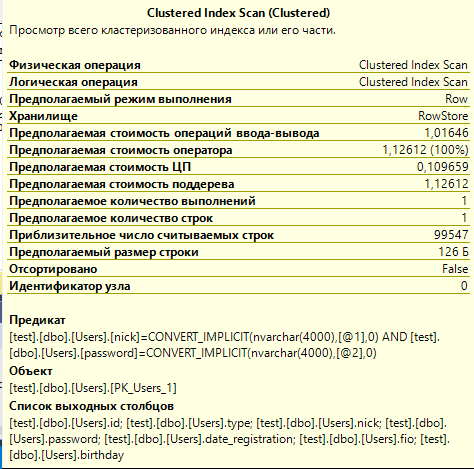
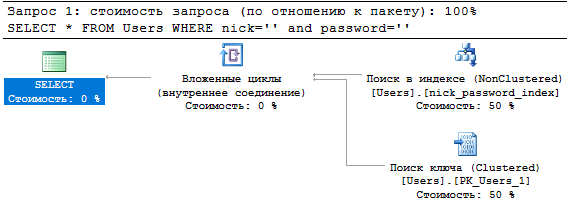


Рисунок 5.1. Карта запроса без некластеризованного индекса

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



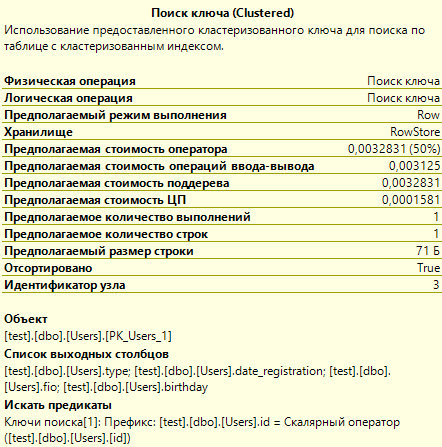
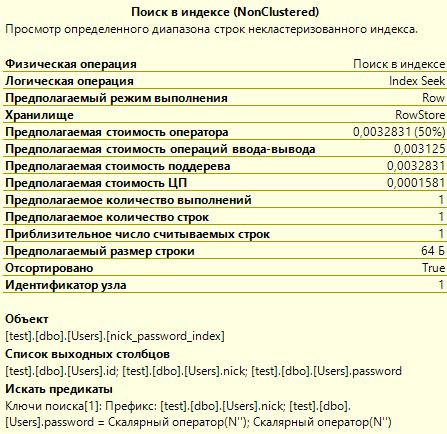
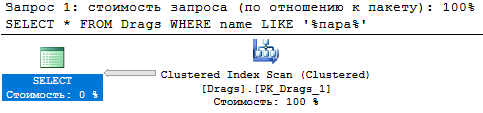


Рисунок 5.2. Карта запроса с некластеризованным индексом

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 с условием. В данный курсовой проект содержит 11 некластерированных индексов они представлены в приложении В.

## 5.4 Полнотекстовые индексы

Если появляется необходимость делать выборку по шаблону по какой-нибудь строке с использованием ключевого слово LIKE (за исключение точного сравнивания), то некластеризованные индексы не будут давать совершенно никакого выигрыша. Демонстрационный пример запроса и его стоимости на основе базы данных данного курсового проекта представлен ниже на рисунке 5.3.



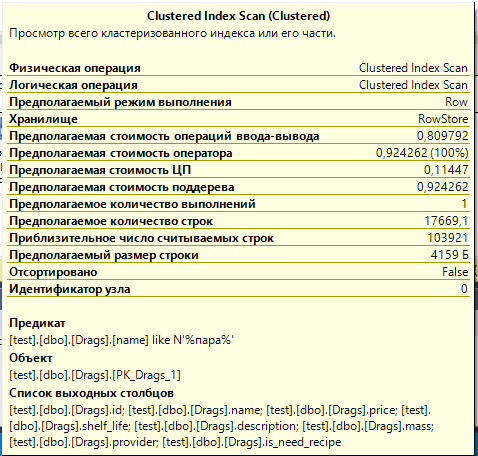


Рисунок 5.3. Карта запроса с некластеризованным индексом по полю name

Для ускорения подобных задач следует использовать полнотекстовые индексы, специально предназначенные для работы со строками и обладающие много большим потенциалом выборки, чем возможности ключевого слова LIKE. Создать полнотекстовой индекс можно через среду MS sql manager studio. Ниже представлен пример стоимости запроса того же запроса, только используя полнотекстовой индекс в процедуре get\_drag\_by\_name курсового проекта на рисунке 5.4.

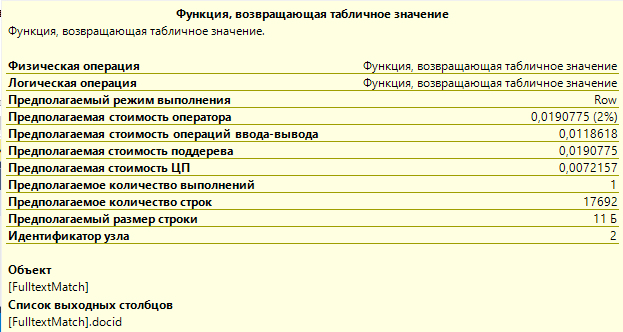


Рисунок 5.3. Стоимость запроса с полнотекстовым индексом по полю name

Как можно увидеть на выборку одного и того же количество строк по одинаковому ключу ушло гораздо меньше времени на примере 5.4. Следовательно для строк, не использующих строгое сравнение полнотекстовой индекс является оптимальным решением для быстрой выборки.

# 6. 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. Исходный код клиента прилагается на электронном носителе.

## 6.1. 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.

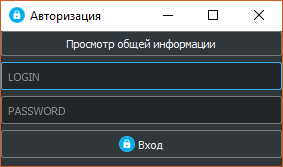


Рисунок 6.1. Окно авторизации

Так же пользователь может просмотреть общедоступную информацию кликнув по верхней кнопке в окне, представленном на рисунке 6.1. При нажатии откроется новое окно, представленное на рисунке 6.2

.

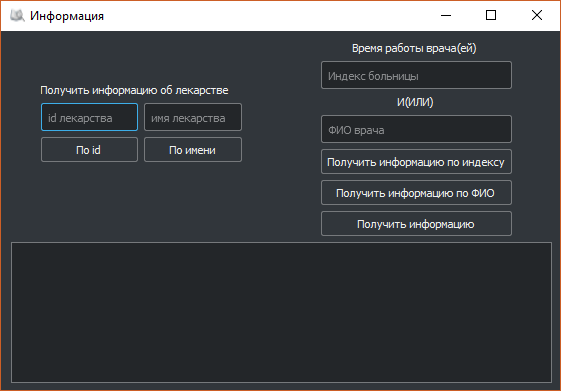


Рис. 6.2. Окно общедоступной информации

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 56.3.

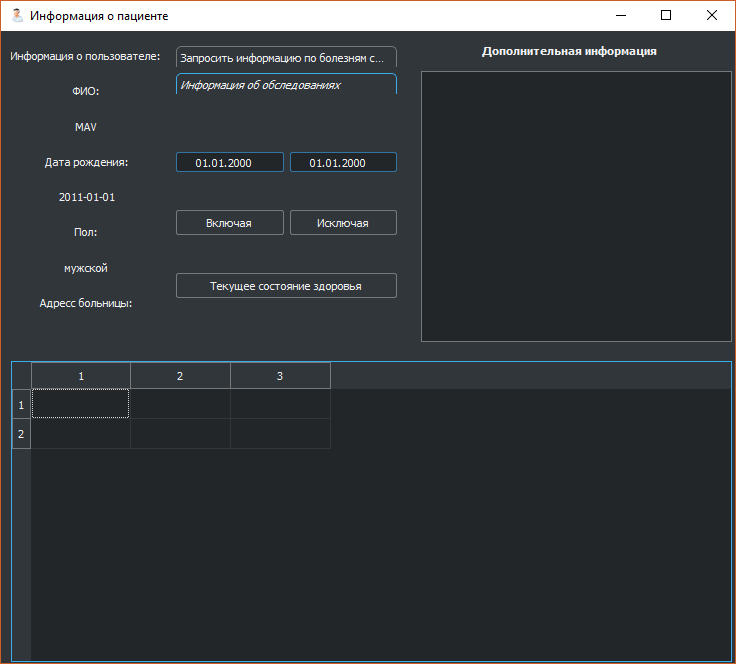


Рисунок 6.3. Окно пациента

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 6.4.

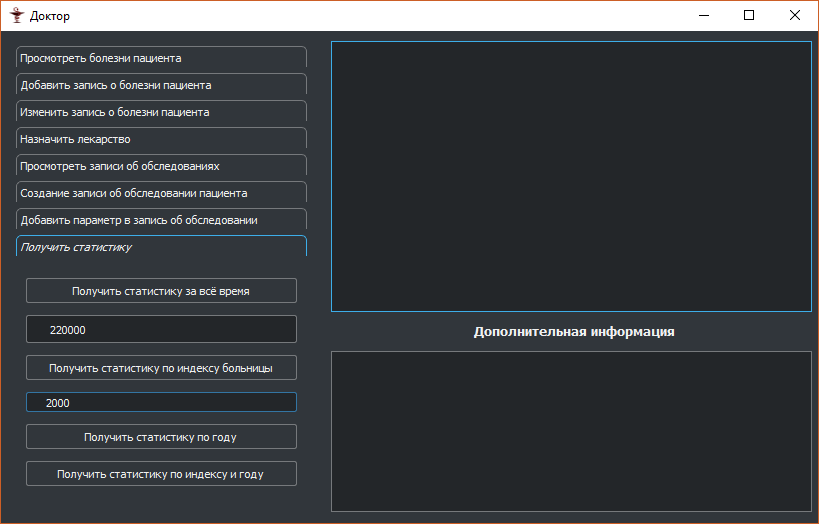


Рисунок 6.4. Окно врача

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 6.5.

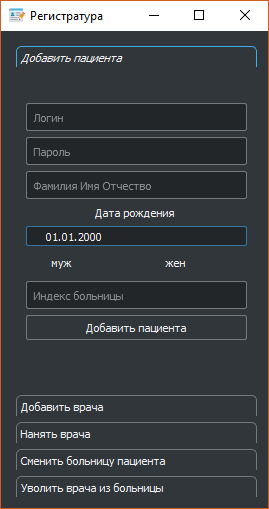


Рисунок 6.5. Окно регистратора

Регистратор имеет возможность создавать пациентов и врачей. Нанимать и увольнять врача на работу в больницу и менять привязку пациента к больнице. Так же ему доступны процедуры импорта и экспорта таблицы больниц в и из xml файлов.

# 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

CREATE PROCEDURE [dbo].[add\_examination]

@doc\_login nvarchar(50),

@doc\_password nvarchar(50),

@login nvarchar(50),

@name nvarchar(50),

@date date

AS

DECLARE @doc\_id int = 0;

DECLARE @pat\_id int = 0;

SELECT top(1) @doc\_id = Doctors.id FROM Doctors, Users

WHERE nick = @doc\_login and password = @doc\_password

and Doctors.user\_id = Users.id;

SELECT top(1) @pat\_id = Patients.id FROM Patients, Users

WHERE nick = @login and Patients.user\_id = Users.id;

if(@doc\_id!=0 and @pat\_id!=0)

begin

INSERT INTO Examinations

(doctor\_id,patient\_id,param, check\_date, name)

values(@doc\_id,@pat\_id,'<root><empty id="0"></empty></root>', @date, @name);

SELECT 0;

end;

else SELECT 219;

Листинг Б.3. Процедура добавления записи об обследовании пациента

CREATE PROCEDURE [dbo].[add\_param]

@id int,

@doc\_login nvarchar(50),

@doc\_password nvarchar(50),

@name nvarchar(50),

@val nvarchar(50)

AS

DECLARE @doctor\_id int = 0;

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(@doctor\_id!=0)

BEGIN

UPDATE Examinations

SET param.modify('

insert (<row><name>"{sql:variable("@name")}"</name><value>"{sql:variable("@val")}"</value></row>)

after (//empty)[1]

')

WHERE id=@id;

SELECT 0;

END

else

select 217;

Листинг Б.4. Процедура добавления параметра в запись об обследовании пациента

CREATE PROCEDURE [dbo].[add\_patient]

@fio nvarchar(50),

@gender bit,

@hospital\_zip int,

@nick nvarchar(50),

@password nvarchar(50),

@birthday date

AS

DECLARE @check\_nick int;

DECLARE @check\_zip int;

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

SELECT @check\_zip = count(\*) FROM Hospitals WHERE @hospital\_zip=zip;

print @check\_zip;

if (@check\_zip =0) SELECT 202;

else

if(@check\_nick = 0)

BEGIN

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

values(1,@nick,@password,SYSDATETIME(), @fio,@birthday);

DECLARE @user\_id int;

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

DECLARE @hospital\_id int;

SELECT top(1) @hospital\_id = id FROM Hospitals WHERE @hospital\_zip=zip;

INSERT INTO Patients(gender,hospital\_id,user\_id)

VALUES(@gender,@hospital\_id,@user\_id);

SELECT 0;

END;

else SELECT 201;

Листинг Б.5. Процедура добавления пациента

CREATE PROCEDURE [dbo].[appoint\_drag]

@id int,

@name nvarchar(50)

AS

DECLARE @drag\_id int = 0;

SELECT top(1) @drag\_id = Drags.id

FROM Drags

WHERE name=@name;

DECLARE @dragdisease\_id int = 0;

SELECT @dragdisease\_id = count(\*)

FROM DiseasesDrags

WHERE @id=disease\_id and @drag\_id=drag\_id;

if(@drag\_id!=0 and @dragdisease\_id=0)

BEGIN

INSERT INTO DiseasesDrags

VALUES(@id, @drag\_id);

SELECT 0;

END

else select 214;

Листинг Б.6. Процедура назначения лекарства пациенту

CREATE PROCEDURE [dbo].[auth\_user]

@login nvarchar(50),

@password nvarchar(50)

AS

SELECT \* FROM Users

WHERE @login=Users.nick and @password=Users.password;

Листинг Б.6. Процедура назначения лекарства пациенту

CREATE PROCEDURE [dbo].[change\_disease]

@id int,

@date\_end date

AS

declare @begin\_date date = NULL;

SELECT top(1) @begin\_date= Diseases.begin\_date FROM Diseases WHERE Diseases.id=@id;

if(@begin\_date<@date\_end)

BEGIN

UPDATE Diseases

SET end\_date = @date\_end

WHERE Diseases.id = @id;

SELECT 0;

END

else select 213;

Листинг Б.7. Процедура добавления даты окончания болезни пациента

CREATE PROCEDURE [dbo].[change\_patient]

@login nvarchar(50),

@zip int

AS

DECLARE @patient\_id int = 0;

DECLARE @hospital\_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) @hospital\_id = Hospitals.id FROM Hospitals WHERE zip=@zip;

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

BEGIN

UPDATE Patients

SET hospital\_id = @hospital\_id

WHERE id = @patient\_id;

SELECT 0;

END

else select 205;

Листинг Б.8. Процедура изменения привязки пациента к больнице

CREATE PROC [dbo].[EXPORT\_HOSPITALS]

AS

SELECT address, zip, phone, main\_doctor

FROM Hospitals

FOR XML PATH('List'), ROOT('Root');

Листинг Б.9. Процедура экспорта таблицы больниц в xml

CREATE PROCEDURE [dbo].[del\_doctor]

@login nvarchar(50),

@zip int

AS

DECLARE @check\_type int;

DECLARE @doctor\_id int = 0;

DECLARE @hospital\_id int = 0;

DECLARE @doctorhospital\_id int = 0;

SELECT top(1) @check\_type = Users.type FROM Users WHERE nick = @login;

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

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

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

SELECT top(1) @hospital\_id = Hospitals.id FROM Hospitals WHERE zip=@zip;

SELECT top(1) @doctorhospital\_id = DoctorsHospitals.id FROM DoctorsHospitals

WHERE hospital\_id=@hospital\_id and @doctor\_id=doctor\_id;

if(@check\_type = 1 and @doctorhospital\_id != 0)

BEGIN

UPDATE top(1) DoctorsHospitals

SET date\_end\_working=GETDATE()

WHERE @doctorhospital\_id = DoctorsHospitals.id;

SELECT 0;

END

else select 206;

Листинг Б.10. Процедура увольнения врача из больницы

CREATE PROCEDURE [dbo].[get\_doctor\_examination\_param]

@login nvarchar(50),

@password nvarchar(50),

@id int

AS

DECLARE @x xml;

DECLARE @name nvarchar(50) = '';

SELECT @x=Examinations.param, @name= Examinations.name

FROM Examinations, Doctors, (SELECT top(1) \* FROM Users WHERE password=@password and @login=nick) as curr\_user

WHERE Examinations.id=@id and Doctors.user\_id=curr\_user.id and Examinations.doctor\_id=Doctors.id;

SELECT

@name,

Tbl.Col.value('name[1]', 'nvarchar(50)') as 'Param name',

Tbl.Col.value('value[1]', 'nvarchar(50)') as 'Param value'

FROM @x.nodes('//row') Tbl(Col);

Листинг Б.11. Процедура получения параметров обследования пациента

CREATE PROCEDURE [dbo].[get\_drag\_by\_id]

@id int

AS

SELECT Drags.id, Drags.name, Drags.price, Drags.shelf\_life, Drags.description, Drags.mass, Drags.provider, Drags.is\_need\_recipe

FROM Drags

WHERE id=@id;

Листинг Б.12. Процедура получения информации о лекарстве по id

CREATE PROCEDURE [dbo].[get\_current\_state\_health]

@login nvarchar(50),

@password nvarchar(50)

AS

DECLARE param\_cursor cursor local read\_only for

SELECT Examinations.name, Examinations.param

FROM Patients, Examinations, Users

WHERE Users.nick=@login and Users.password=@password

and Patients.user\_id=Users.id and Examinations.patient\_id=Patients.id

ORDER BY Examinations.check\_date DESC;

CREATE TABLE #tmp (name nvarchar(50), par nvarchar(50), val nvarchar(50));

DECLARE @params xml;

DECLARE @name nvarchar(50);

open param\_cursor;

fetch next from param\_cursor into @name, @params;

while @@FETCH\_STATUS = 0

begin

DECLARE temp\_cursor cursor local read\_only for

SELECT

@name,

Tbl.Col.value('name[1]', 'nvarchar(50)') as name,

Tbl.Col.value('value[1]', 'nvarchar(50)') as value

FROM @params.nodes('//row') Tbl(Col);

open temp\_cursor;

DECLARE @exm\_name nvarchar(50);

DECLARE @exm\_par nvarchar(50);

DECLARE @exm\_val nvarchar(50);

fetch next from temp\_cursor into @exm\_name, @exm\_par, @exm\_val;

while @@FETCH\_STATUS = 0

begin

INSERT INTO #tmp SELECT @exm\_name, @exm\_par, @exm\_val

WHERE NOT EXISTS(SELECT top(1) \* FROM #tmp WHERE #tmp.name=@exm\_name and #tmp.par=@exm\_par)

fetch next from temp\_cursor into @exm\_name, @exm\_par, @exm\_val;

end;

close temp\_cursor;

DEALLOCATE temp\_cursor;

fetch next from param\_cursor into @name, @params;

end;

SELECT \* FROM #tmp;

DROP TABLE #tmp;

Листинг Б.13. Процедура получения информации о текущем состоянии здоровья

CREATE PROCEDURE [dbo].[get\_doctor\_examinations]

@login nvarchar(50),

@doc\_login nvarchar(50),

@doc\_password nvarchar(50),

@date\_begin date,

@date\_end date

AS

DECLARE @pat\_id int = 0;

DECLARE @doc\_id int = 0;

SELECT top(1) @pat\_id = Patients.id FROM Users, Patients

WHERE nick = @login and Patients.user\_id=Users.id;

SELECT top(1) @doc\_id = Doctors.id FROM Users, Doctors

WHERE nick = @doc\_login and password=@doc\_password and Doctors.user\_id=Users.id;

if (@doc\_id!=0 and @pat\_id!=0)

begin

SELECT id, check\_date, name FROM Examinations

WHERE doctor\_id=@doc\_id and patient\_id=@pat\_id

and @date\_begin < Examinations.check\_date and @date\_end > Examinations.check\_date;

end;

else SELECT 217;

Листинг Б.14. Процедура получения информации об обследованиях пациента

CREATE PROCEDURE [dbo].[get\_drag\_by\_name]

@name nvarchar(50)

AS

set @name = '"\*'+@name+'\*"'

SELECT Drags.id, Drags.name, Drags.price, Drags.shelf\_life, Drags.description, Drags.mass, Drags.provider, Drags.is\_need\_recipe

FROM Drags

WHERE CONTAINS(name, @name);

Листинг Б.15. Процедура получения информации о лекарстве по имени

CREATE PROCEDURE [dbo].[get\_drags\_by\_disease]

@login nvarchar(50),

@password nvarchar(50),

@id int

AS

SELECT Drags.id, Drags.name, Drags.price, Drags.shelf\_life, Drags.description, Drags.mass, Drags.provider, Drags.is\_need\_recipe

FROM (SELECT top(1) Diseases.id

FROM Patients, Diseases, (SELECT top(1) \* FROM Users WHERE @login=nick) as curr\_user

WHERE @id = Diseases.id and curr\_user.id = Patients.user\_id and Patients.id = Diseases.patient\_id)

as curr\_disease, DiseasesDrags, Drags

WHERE curr\_disease.id=DiseasesDrags.disease\_id and Drags.id=DiseasesDrags.drag\_id;

Листинг Б.16. Процедура получения информации о назначенных лекарствах на данное заболевание пациенту

CREATE PROCEDURE [dbo].[get\_drags\_by\_disease\_doctor]

@login nvarchar(50),

@password nvarchar(50),

@id int

AS

SELECT Drags.id, Drags.name, Drags.price, Drags.shelf\_life, Drags.description, Drags.mass, Drags.provider, Drags.is\_need\_recipe

FROM (SELECT top(1) Diseases.id

FROM Doctors, Diseases, (SELECT top(1) \* FROM Users WHERE password=@password and @login=nick) as curr\_user

WHERE @id = Diseases.id and curr\_user.id = Doctors.user\_id and Doctors.id = Diseases.doctor\_id)

as curr\_disease, DiseasesDrags, Drags

WHERE curr\_disease.id=DiseasesDrags.disease\_id and Drags.id=DiseasesDrags.drag\_id;

Листинг Б.17. Процедура получения информации о назначенных лекарствах на данное заболевание пациенту

CREATE PROCEDURE [dbo].[get\_exclude\_doctor\_examinations]

@login nvarchar(50),

@doc\_login nvarchar(50),

@doc\_password nvarchar(50),

@date\_begin date,

@date\_end date

AS

DECLARE @pat\_id int = 0;

DECLARE @doc\_id int = 0;

SELECT top(1) @pat\_id = Patients.id FROM Users, Patients

WHERE nick = @login and Patients.user\_id=Users.id;

SELECT top(1) @doc\_id = Doctors.id FROM Users, Doctors

WHERE nick = @doc\_login and password=@doc\_password and Doctors.user\_id=Users.id;

if (@doc\_id!=0 and @pat\_id!=0)

begin

SELECT id, check\_date, name FROM Examinations

WHERE doctor\_id=@doc\_id and patient\_id=@pat\_id

and (@date\_begin > Examinations.check\_date or @date\_end < Examinations.check\_date);

end;

else SELECT 217;

Листинг Б.17. Процедура получения врачом информации об обследованиях пациента

CREATE PROCEDURE [dbo].[get\_inverse\_patient\_diseases]

@login nvarchar(50),

@password nvarchar(50),

@date\_begining date,

@date\_ending date

AS

SELECT Diseases.id, Diseases.name\_disease, Diseases.begin\_date, Diseases.end\_date, Diseases.patient\_id

FROM Patients, Diseases, (SELECT top(1) \* FROM Users WHERE password=@password and @login=nick) as curr\_user

WHERE curr\_user.id = Patients.user\_id and Patients.id = Diseases.patient\_id

and (@date\_begining>Diseases.end\_date or @date\_ending<Diseases.begin\_date);

Листинг Б.18. Процедура получения информации об обследованиях пациента не входящих в указанный временной диапазон

CREATE PROCEDURE [dbo].[get\_inverse\_patient\_examinations]

@login nvarchar(50),

@password nvarchar(50),

@date\_begin date,

@date\_end date

AS

DECLARE @pat\_id int = 0;

SELECT top(1) @pat\_id = Patients.id FROM Users, Patients

WHERE nick = @login and @password = password and Patients.user\_id=Users.id;

if (@pat\_id!=0)

begin

SELECT Examinations.id, check\_date, name, Users.fio

FROM Examinations, Doctors, Users

WHERE patient\_id=@pat\_id and Examinations.doctor\_id = Doctors.id and Users.id = Doctors.user\_id

and (@date\_begin > Examinations.check\_date or @date\_end < Examinations.check\_date);

end;

else SELECT 250;

Листинг Б.19. Процедура получения информации об обследованиях пациента не входящий в указанный диапазон дат

CREATE PROCEDURE [dbo].[get\_param\_examination]

@login nvarchar(50),

@password nvarchar(50),

@id int

AS

DECLARE @x xml;

DECLARE @name nvarchar(50) = '';

SELECT @x=Examinations.param, @name= Examinations.name

FROM Examinations, Patients, (SELECT top(1) \* FROM Users WHERE password=@password and @login=nick) as curr\_user

WHERE Examinations.id=@id and Patients.user\_id=curr\_user.id and Examinations.patient\_id=Patients.id;

SELECT

@name,

Tbl.Col.value('name[1]', 'nvarchar(50)') as 'Param name',

Tbl.Col.value('value[1]', 'nvarchar(50)') as 'Param value'

FROM @x.nodes('//row') Tbl(Col);

Листинг Б.20. Процедура получения параметров обследования

CREATE PROCEDURE [dbo].[get\_patient]

@login nvarchar(50),

@password nvarchar(50)

AS

SELECT top(1) curr\_user.fio, curr\_user.birthday, Patients.gender, Hospitals.address

FROM Hospitals, Patients, (SELECT top(1) \* FROM Users WHERE password=@password and @login=nick) as curr\_user

WHERE curr\_user.id = Patients.user\_id and Hospitals.id=Patients.hospital\_id;

Листинг Б.21. Процедура получения информации о пациенте

CREATE PROCEDURE [dbo].[get\_patient\_diseases]

@login nvarchar(50),

@password nvarchar(50),

@date\_begining date,

@date\_ending date

AS

SELECT Diseases.id, Diseases.name\_disease, Diseases.begin\_date, Diseases.end\_date, Diseases.patient\_id

FROM Patients, Diseases, (SELECT top(1) \* FROM Users WHERE password=@password and @login=nick) as curr\_user

WHERE curr\_user.id = Patients.user\_id and Patients.id = Diseases.patient\_id

and @date\_begining<Diseases.end\_date and @date\_ending>Diseases.begin\_date;

Листинг Б.22. Процедура получения информации о заболеваниях пациента

CREATE PROCEDURE [dbo].[get\_patient\_examinations]

@login nvarchar(50),

@password nvarchar(50),

@date\_begin date,

@date\_end date

AS

DECLARE @pat\_id int = 0;

SELECT top(1) @pat\_id = Patients.id FROM Users, Patients

WHERE nick = @login and @password = password and Patients.user\_id=Users.id;

if (@pat\_id!=0)

begin

SELECT Examinations.id, check\_date, name, Users.fio

FROM Examinations, Doctors, Users

WHERE patient\_id=@pat\_id and Examinations.doctor\_id = Doctors.id and Users.id = Doctors.user\_id

and @date\_begin < Examinations.check\_date and @date\_end > Examinations.check\_date;

end;

else

SELECT 250;

Листинг Б.23. Процедура получения информации об обследованиях пациента

CREATE PROCEDURE [dbo].[get\_statistic\_by\_all\_time]

AS

SELECT \*

FROM (

SELECT

name\_disease

, YearMounth = DATENAME(MONTH, begin\_date)

FROM Diseases

) t

PIVOT (

COUNT(YearMounth)

FOR YearMounth IN (

January, February, March, April, May, June, July, August, September, October, November, December

)

) p

Листинг Б.24. Процедура отображения статистики по заболеваниям за всё время

CREATE PROCEDURE [dbo].[get\_statistic\_by\_index]

@index int

AS

SELECT \*

FROM (

SELECT

name\_disease

, YearMounth = DATENAME(MONTH, begin\_date)

FROM Diseases, Patients, Hospitals WHERE Diseases.patient\_id=Patients.id and Patients.hospital\_id=Hospitals.id

and Hospitals.zip= @index

) t

PIVOT (

COUNT(YearMounth)

FOR YearMounth IN (

January, February, March, April, May, June, July, August, September, October, November, December

)) p

Листинг Б.25. Процедура отображения статистики по заболеваниям по конкретной больнице

CREATE PROCEDURE [dbo].[get\_statistic\_by\_index\_year]

@index int,

@date date

AS

SELECT \*

FROM (

SELECT

name\_disease

, YearMounth = DATENAME(MONTH, begin\_date)

FROM Diseases, Patients, Hospitals WHERE Diseases.patient\_id=Patients.id and Patients.hospital\_id=Hospitals.id

and Hospitals.zip= @index and Diseases.begin\_date>=@date and Diseases.end\_date<=(DATEADD(YEAR, 1,@date))

) t

PIVOT (

COUNT(YearMounth)

FOR YearMounth IN (

January, February, March, April, May, June, July, August, September, October, November, December

)) p

Листинг Б.26. Процедура отображения статистики по заболеваниям по конкретной больнице и году

CREATE PROCEDURE [dbo].[get\_statistic\_by\_year]

@date date

AS

SELECT \*

FROM (

SELECT

name\_disease

, YearMounth = DATENAME(MONTH, begin\_date)

FROM Diseases WHERE

Diseases.begin\_date>=@date and Diseases.end\_date<=(DATEADD(YEAR, 1,@date))

) t

PIVOT (

COUNT(YearMounth)

FOR YearMounth IN (

January, February, March, April, May, June, July, August, September, October, November, December)) p

Листинг Б.27. Процедура отображения статистики по заболеваниям по конкретному году

CREATE PROCEDURE [dbo].[get\_time\_work\_by\_fio]

@fio nvarchar(50)

AS

SELECT Users.fio, DoctorsHospitals.time\_begin\_working, DoctorsHospitals.time\_end\_working, Hospitals.zip

FROM Users, Hospitals, Doctors, DoctorsHospitals

WHERE Users.type=3 and Doctors.user\_id=Users.id and DoctorsHospitals.doctor\_id = Doctors.id

and Hospitals.id = DoctorsHospitals.hospital\_id and Users.fio=@fio

and (date\_end\_working> GETDATE() or date\_end\_working=NULL);

Листинг Б.28. Процедура отображения рабочего времени врача по ФИО

CREATE PROCEDURE [dbo].[get\_time\_work\_by\_zip]

@zip int

AS

SELECT Users.fio, DoctorsHospitals.time\_begin\_working, DoctorsHospitals.time\_end\_working, Hospitals.zip

FROM Users, Hospitals, Doctors, DoctorsHospitals

WHERE Users.type=3 and Doctors.user\_id=Users.id and DoctorsHospitals.doctor\_id = Doctors.id

and Hospitals.id = DoctorsHospitals.hospital\_id and Hospitals.zip=@zip

and (date\_end\_working> GETDATE() or date\_end\_working=NULL);

Листинг Б.29. Процедура отображения рабочего времени врача по индексу больницы

CREATE PROCEDURE [dbo].[get\_time\_work\_by\_zip\_fio]

@zip int,

@fio nvarchar(50) AS

SELECT Users.fio, DoctorsHospitals.time\_begin\_working, DoctorsHospitals.time\_end\_working, Hospitals.zip

FROM Users, Hospitals, Doctors, DoctorsHospitals

WHERE Users.type=3 and Doctors.user\_id=Users.id and DoctorsHospitals.doctor\_id = Doctors.id

and Hospitals.id = DoctorsHospitals.hospital\_id and Users.fio=@fio and Hospitals.zip=@zip

and (date\_end\_working> GETDATE() or date\_end\_working=NULL);

Листинг Б.30. Процедура отображения рабочего времени врача по индексу больницы и ФИО

CREATE PROCEDURE [dbo].[select\_disease]

@login nvarchar(50),

@doc\_login nvarchar(50),

@doc\_password nvarchar(50) 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

SELECT Id, name\_disease, begin\_date, end\_date, Diseases.description FROM Diseases

WHERE @patient\_id=Diseases.patient\_id and @doctor\_id=Diseases.doctor\_id;

END

else select 211;

Листинг Б.31. Процедура просмотра всех заболеваний врачом

CREATE PROCEDURE [dbo].[take\_doctor]

@login nvarchar(50),

@zip int,

@spec nvarchar(50),

@begin\_date date,

@begin\_time time(7),

@end\_time time(7)

AS

DECLARE @check\_type int;

DECLARE @doctor\_id int = 0;

DECLARE @hospital\_id int = 0;

SELECT top(1) @check\_type = Users.type FROM Users WHERE nick = @login;

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

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

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

SELECT top(1) @hospital\_id = Hospitals.id FROM Hospitals WHERE zip=@zip;

if(@check\_type = 3 and @doctor\_id!=0 and @hospital\_id!=0

and NOT EXISTS(SELECT \* FROM DoctorsHospitals, Doctors

WHERE DoctorsHospitals.doctor\_id=@doctor\_id and

((time\_begin\_working<=@begin\_time and @begin\_time<=time\_end\_working)

or (time\_begin\_working<=@end\_time and @end\_time<=time\_end\_working))))

BEGIN

INSERT INTO DoctorsHospitals

VALUES(@doctor\_id, @hospital\_id, @spec, @begin\_time, @end\_time, @begin\_date, NULL)

SELECT 0;

END

else select 204;

Листинг Б.31. Процедура назначения время работы врача в больницу

# ПРИЛОЖЕНИЕ В

**Скрипты создания индексов**

Ниже представлен код создания некластеризованных индексов.

CREATE INDEX nick\_password\_index

ON dbo.Users

(nick, password)

CREATE INDEX nick\_\_index

ON dbo.Users

(nick)

CREATE INDEX patien\_id\_disease\_index

ON dbo.Diseases (patient\_id)

CREATE INDEX patien\_id\_examination\_index

ON dbo.Diseases (patient\_id)

CREATE INDEX user\_id\_patient\_index

ON dbo.Patients (user\_id)

CREATE INDEX user\_id\_doctor\_index

ON dbo.Doctors(user\_id)

CREATE INDEX zip\_index

ON dbo.Hospitals(zip)

CREATE INDEX drag\_name\_index

ON dbo.Drags(name)

CREATE INDEX date\_begin\_end\_index

ON dbo.Diseases(begin\_date, end\_date)

CREATE INDEX check\_date\_index

ON dbo.Examinations(check\_date)

CREATE INDEX DoctorsHospitals\_index

ON dbo.DoctorsHospitals(doctor\_id, hospital\_id)

# ПРИЛОЖЕНИЕ Г

**Наделение пользователей правами**

Ниже представлен t-sql код для наделения пользователей правами.

Наделение правами authorization.

GRANT EXECUTE ON [dbo].[auth\_user] TO [authorization] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_drag\_by\_id] TO [authorization] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_drag\_by\_name] TO [authorization] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_time\_work\_by\_fio] TO [authorization] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_time\_work\_by\_zip] TO [authorization] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_time\_work\_by\_zip\_fio] TO [authorization] WITH GRANT OPTION

Наделение правами patient.

GRANT EXECUTE ON [dbo].[get\_inverse\_patient\_diseases] TO [patient] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_inverse\_patient\_examinations] TO [patient] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_current\_state\_health] TO [patient] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_patient] TO [patient] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_patient\_examinations] TO [patient] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_param\_examination] TO [patient] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_drags\_by\_disease] TO [patient] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_patient\_diseases] TO [patient] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[auth\_user] TO [patient] WITH GRANT OPTION

Наделение правами registrator.

GRANT EXECUTE ON [dbo].[add\_doctor] TO [registrator] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[add\_patient] TO [registrator] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[auth\_user] TO [registrator] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[change\_patient] TO [registrator] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[del\_doctor] TO [registrator] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[EXPORT\_HOSPITALS] TO [registrator] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[IMPORT\_HOSPITALS] TO [registrator] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[take\_doctor] TO [registrator] WITH GRANT OPTION

Наделение правами doctor.

GRANT EXECUTE ON [dbo].[get\_patient\_diseases] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_exclude\_doctor\_examinations] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[select\_users] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[see\_exm\_doc] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_statistic\_by\_index\_year] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_patient] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[auth\_user] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[appoint\_drag] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_param\_examination] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_patient\_examinations] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_statistic\_by\_index] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_statistic\_by\_year] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[select\_disease] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[add\_param] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[change\_disease] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_statistic\_by\_all\_time] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[add\_disease] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[add\_examination] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_doctor\_examinations] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_doctor\_examination\_param] TO [doctor] WITH GRANT OPTION

GRANT EXECUTE ON [dbo].[get\_drags\_by\_disease\_doctor] TO [doctor] WITH GRANT OPTION