ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

УХТИНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

**Кафедра ИСТ**

**Курсовой проект**

**Дисциплина: «Системы управления базами данных»**

**Тема:**

**«Разработка физической модели базы данных «Учёт затрат на медицинские услуги»**

**Выполнил студент группы ИСТ-2-04**

**Петров М.В.**

**Проверила доцент кафедры ИСТ, к. т. н.**

**Николаева Н.А.**

**Ухта 2007**

Содержание

[Введение](#_Toc254462519)

[Часть 1. Постановка задачи](#_Toc254462520)

[1.1. Анализ существующих аналогов](#_Toc254462521)

[1.2. Обоснование выбора бизнес-процесса](#_Toc254462522)

[Часть 2. Технологическая часть](#_Toc254462523)

[2.1. Выбор средств разработки](#_Toc254462524)

[2.2. Основные методы и способы разработки](#_Toc254462525)

[2.3. Модель жизненного цикла](#_Toc254462526)

[Часть 3. Основная часть](#_Toc254462527)

[3.1. Поддержание целостности БД](#_Toc254462528)

[3.2. Поддержание бизнес-логики](#_Toc254462529)

[3.3. Описание интерфейса пользователя](#_Toc254462530)

[3.4. Формирование выходной документации и входных форм](#_Toc254462531)

[3.5. Пользователи и права доступа](#_Toc254462532)

[Заключение](#_Toc254462533)

[Список используемой литературы](#_Toc254462534)

[Приложения](#_Toc254462535)

## Введение

В настоящее время базы данных находят широкое применение в различных организациях (предприятиях, банках, учебных заведениях и т.д.). Это связано с тем, что они позволяют хранить информацию в таком виде, чтобы ее было удобно использовать, например, осуществлять поиск по каким-либо заданным параметрам, производить вычисления, используя данные из базы данных или собирать различную статистику.

Княжпогостский филиал ФОМСа является одной из организаций, осуществляющих финансирование бюджетных медицинских учреждений. При этом специалист по правам застрахованных, который выполняет процесс учёта, обработки и хранения поступившей документации и формирования необходимых отчётов, сталкивается со многими проблемами, такими как:

* Велики затраты времени и ручного труда на ведение всей необходимой документации. Наблюдается однотипность выполняемых операций. Ежедневно в филиал поступает около 100-150 статистических талонов, несколько десятков карт выбывших больных; а также ежемесячно поступают приказы и извещения об оплате от каждого ЛПУ.
* Трудоёмок процесс формирования отчётов. Ежемесячно специалисту приходится вести подсчёт суммы, которая в дальнейшем войдёт в заявку на финансирование. Вследствие большого объёма документации, накопленной за месяц, велика вероятность ошибки в начисленной сумме.

Из приведенных выше проблем следует, что необходимо создать систему, которая будет хранить, обрабатывать и предоставлять пользователю всю необходимую информацию. Поэтому **цель** данного курсового проекта будет заключаться в разработке физической модели базы данных для процесса учета затрат на оказанные медицинские услуги.

На предыдущих этапах работы было проведено изучение и анализ предметной области, построены контекстная диаграмма и DFD 1-го уровня. Были составлены словарь данных и написаны спецификации процессов. Затем были построены концептуальная и логическая модели базы данных, которые и послужили основой для создания физической модели, а также написаны запросы к базе данных на языке реляционной алгебры, которые были преобразованы в запросы на языке T-SQL. Бизнес-правила нашли свое выражение в виде триггеров и ограничений.

Курсовой проект состоит из трех частей. В первой части, производится постановка задачи, обоснование разработки, обоснование выбора автоматизируемого бизнес-процесса.

Во второй части описываются основные методы и способы разработки, средства разработки, модель жизненного цикла системы.

В третьей части, основной, описываются основные принципы поддержания целостности базы данных, реализация бизнес-правил, описание интерфейса пользователя и принципы формирования выходной документации и входных форм.

В заключении приводятся выводы и итоги проделанной работы.

## Часть 1. Постановка задачи

## 1.1. Анализ существующих аналогов

В ходе изучения предметной области были изучены аналоги создаваемой системы. В настоящее время в Княжпогостском филиале ФОМС существует система, имеющая примерно ту же функциональность, что и наша. Недостатком существующей системы является крайне неудобный интерфейс (выполнен на основе библиотеки Turbo Vision или подобной), некрасивый внешний вид, а также избыточность предоставляемой информации. Все это позволяют надеяться, что создаваемая система способна заменить существующую при соответствующей доработке.

## 1.2. Обоснование выбора бизнес-процесса

При разработке курсового проекта стояла альтернатива, какой именно процесс подлежит автоматизации. Надо отметить, что это очень важный вопрос, так как от выбора зависит выразительность и полнота курсового проекта, а также наглядность клиентского приложения. В ходе анализа предметной области был выделен основной процесс, подлежащий автоматизации - «Учесть затраты на оказанные медицинские услуги», в дальнейшем он был декомпозирован на 5 подпроцессов:

* Фиксировать полученную документацию;
* Получить документацию;
* Сформировать заявку;
* Оформить платёжное поручение;
* Получить извещение об оплате.

В ходе разработке данного курсового проекта были автоматизированы все подпроцессы, протекающие в Княжпогостском филиале ФОМС при учете затрат на медицинские услуги. Это было сделано, так как при отказе от автоматизации какого-либо подпроцесса станет невозможным формирование основного выходного документа - заявки на финансирование. Автоматизация всех процессов сделало разрабатываемую систему законченной и полной. Кроме того, правильный выбор средств разработки позволил в сжатые сроки создать полноценную систему.

## Часть 2. Технологическая часть

## 2.1. Выбор средств разработки

В качестве целевой СУБД была выбрана Microsoft SQL Server 2005. SQL Server 2005 - это новейшая версия одной из систем управления базами данных, достигшая того непревзойдённого уровня развития, к которому она постепенно приближалась на протяжении двух десятилетий. Данная версия явилась результатом коренной переработки, которой подвергается этот программный продукт, начиная с версии 7.0. Но в программном обеспечении SQL Server 2005 удалось значительно улучшить совместимость компонентов и расширить набор средств, обеспечивающих взаимодействие с языком XML, инфраструктурой .NET, определяемыми пользователем типами данных, а также многими другими дополнительными службами.

Вообще говоря, SQL Server 2005 позволяет не только хранить данные, но и управлять ими, регламентировать типы данных, а также упрощать процесс получения этих данных. Если задача состоит в том, чтобы просто сохранить данные в надёжном месте, то достаточно воспользоваться практически любой системой хранения данных. Однако SQL Server 2005 как реляционная СУБД позволяет не только хранить данные, но и непосредственно задавать структуру данных, иными словами, устанавливать бизнес-правила, которым должны подчиняться данные.

Функционирование выбранной СУБД организованно так, что запись модифицированных файлов не осуществляется непосредственно в файл базы данных. Вместо этого вся информация обо всех изменениях записывается в журнал транзакций. В какой-то последующий момент времени применительно к базе данных выполняется контрольная точка, и в этот момент времени все изменения и дополнения, зафиксированные в журнале, переносятся в физический файл (файлы) базы данных.

В версии SQL Server 2005 предусмотрено много инструментальных средств проектирования, которые существенно изменились по сравнению с предыдущими версиями. К сожалению, методология создания диаграмм, предусмотренная в этих программных средствах, не соответствует ни одному из общепринятых стандартов формирования ER-диаграмм. Тем не менее эти инструментальные средства формирования диаграмм обеспечивают выполнения всех «обязательных» операций; по крайней мере, с их помощью можно приступить к освоению соответствующих методов.

Таким образом, выбранная целевая СУБД удовлетворяет всем требованиям программиста, желающего изготовить качественный программный продукт.

Клиентское приложение было разработано в среде Microsoft Visual Studio 2005. Эта среда использует технологию программирования .NET, которая вместе со связанной с ней средой .NET Framework, является одной из самых важных технологий для разработчиков ПО за много лет. .NET спроектирована как новая среда, в рамках которой можно разработать практически любое приложение для Windows. Данная версия среды Visual Studio использует .NET Framework 2.0 - третья версия этой среды. Далее мы вкратце перечислим преимущества технологии .NET перед другими технологиями разработки:

* Объектно-ориентированное программирование - и среда .NET Framework изначально полностью базировалась на объектно-ориентированных принципах.
* Хороший дизайн - библиотека базовых классов, которая спроектирована «с нуля», исключительно интуитивно понятным образом.
* Независимость от языка - с .NET код всех языков компилируется в общий язык промежуточного уровня - Intermediate Language. Это значит, что ранее все эти языки обладают возможностями взаимодействия, как никогда ранее.
* Эффективный доступ к данным - набор компонентов .NET, известный под общим названием ADO.NET предоставляет эффективный доступ к реляционным базам данных и широкому разнообразию других источников данных. Также доступны компоненты, предоставляющие доступ к файловой системе и каталогам.
* Повышенная безопасность - каждая сборка также может содержать встроенную информацию безопасности, которая в точности описывает, кому и каким пользователем или процессов какие методы каких классов разрешено вызывать.
* C# - новый объектно-ориентированный язык, предназначенный для применения с .NET.

Заметим, что Visual Studio 2005 использует .NET Framework 2.0. Эта среда также имеет некоторые преимущества по сравнению с предыдущими версиями .NET Framework, а именно:

* Интеграция с SQL Server. Для нас важно прежде всего то, что Visual Studio 2005, .NET Framework 2.0 и SQL Server 2005 тесно связаны между собой в том смысле, что реализованы в сочетании.
* Поддержка 64-разрядных вычислений. Сегодня больше и больше предприятий переходят на современные 64-разрядный процессоры. А среда Visual Studio 2005 может компилировать код так, чтобы он работал на любых процессорах.

В заключение можно сказать, что выбор средств разработки является важной задачей при создании программного продукта, а выбранные нами средства позволили создать современное приложение при минимуме усилий, что подтверждает правильность выбора.

## 2.2. Основные методы и способы разработки

После выбора средств разработки появилась необходимость выбора основных методов и способов разработки базы данных. Надо сказать, что СУБД Microsoft SQL Server 2005 даёт нам два основных способа разработки - написание сценариев на языке T-SQL и визуальные средства разработки. В нашей работе использовалось оба метода, и это позволило в достаточно сжатые сроки создать корректную и целостную базу данных.

Преимуществом написания сценариев является меньшая вероятность ошибки при разработке, так как создание таблиц, атрибутов, ограничений прописывается явно, обдумывается каждая строка сценария. Кроме того, при написании сценариев существует возможность отладки созданных объектов базы данных, например, хранимых процедур. Недостатком же сценариев являются большие затраты времени и сил, уходящих на написание кода.

В среду SQL Server 2005 включены также и визуальные средства разработки базы данных. Они, как ясно из названия, предполагаю создание базы данных без написания сценариев, а при помощи нужных панелей инструментов. Теоретически всю работу по созданию базы данных можно выполнить, вообще не прикасаясь к клавиатуре! Но такой способ разработки чреват большим количеством ошибок, так как легко выбрать не тот пункт выпадающего списка или совершить подобныю ошибку. Кроме того, создавать сложные запросы, представления, триггеры при помощи визуальных средств очень трудно и также чревато большим количеством ошибок.

Поэтому при разработке базы данных применялся другой путь - основная работа была проделана с помощью сценариев, а потом созданная физическая модель базы данных была доработана и отредактирована визуальными средствами. Необходимо отметить, что такой способ является оптимальным для начинающих разработчиков, так как он позволяет избежать ошибок, но в то же время сэкономить время и изучить визуальные средства разработки.

## 2.3. Модель жизненного цикла

Согласно RUP (Rational Unified Process) жизненный цикл информационной системы делится на следующие стадии:

* Постановка задачи;
* Анализ;
* Проектирование;
* Реализация (кодирование);
* Отладка;
* Тестирование;
* Внедрение;
* Эксплуатация.

Естественно, в ходе разработки нашей системы было сложно полностью провести все этапы жизненного цикла, но большинство стадий все-таки было проведено. Далее рассмотрим все пройденные в процессе разработки этапы.

На этапе **постановки задачи**, как ясно из названия, происходит постановка задачи, определяются функциональные и нефункциональные требования, пишется техническое задание на разработку системы. Эта стадия была подробно рассмотрена в курсовом проекте по дисциплине «Информационные технологии».

На стадии **анализа** происходит изучение и анализ предметной области, построение контекстной диаграммы и DFD нижних уровней. Разрабатываются прецеденты, диаграммы прецедентов, последовательностей, взаимодействия и другие. Стадия анализа описана в курсовых проектах по дисциплинам «Информационные технологии» и «Теория информации».

На стадии **проектирования** происходит разработка проекта системы, строятся модели базы данных (концептуальная и логическая), а также разрабатывается общая архитектура системы. Стадия проектирования подробно описана в курсовых проектах по дисциплинам «Управление данными» и «Теория информации».

Стадия **реализации** или кодирования характеризуется непосредственным созданием компонентов системы. В нашем случае стадия реализации заключалась в создании базы данных, хранимых процедур и триггеров, а также в создании клиентского приложения для управления данными. Эта стадия была самой трудоёмкой и долгой, так как необходимо было изучить различные технологии для реализации (язык запросов SQL, технологию доступа к данным ADO.NET, язык C#).

На стадии **отладки** происходит первоначальный поиск ошибок и их исправление. Эта стадия тесно связана со стадией тестирования. В нашем случае тестирования не проводилось, поэтому стадия отладки приобрела особую важность, она выявила некоторые ошибки, допущенные на этапе реализации, и позволила их исправить.

Необходимо отметить, что модель разработки данной системы напоминает **итерационную** модель. Она характеризуется тем, что в ней присутствуют ярко выраженные связи между этапами. То есть на любой из стадий возможно уточнение и дополнение предыдущей стадии. Это позволяет существенно снизить трудоёмкость отладки и реализации. В нашем случае это выразилось в уточнении описания предметной области на стадии реализации системы, дополнение стадии анализа прецедентами и диаграммами при проектировании и кодировании, а также исправление ошибок реализации на этапе отладки. Создание корректно работающей системы, удовлетворяющей всем требованиям, позволяет сказать, что выбранная модель разработки в нашем случае является оптимальной.

## Часть 3. Основная часть

## 3.1. Поддержание целостности БД

Поддержание целостности базы данных является очень важной задачей, так как это является одним из условий нормального функционирования разрабатываемой системы. База данных находится в состоянии целостности (согласованном состоянии), если выполнены все ограничения целостности, определённые для БД.

Все меры по поддержке целостности базы данных можно разделить на 2 большие группы:

1. Декларативная целостность (ограничения);
2. Процедурная целостность (триггеры, правила и т.д.).

**Ограничения (CONSTRAINTS)** представляют собой некоторые условия, налагаемые на столбцы, таблицы и гарантирующие, что ваша информация будет подчиняться определённым правилам целостности данных. Надо отметить, что имеется 2 типа реакции на попытку нарушения целостности - отказ и выполнение «компенсирующих» действий. Для данного проекта были использованы ограничения отказа, то есть запрещения выполнения некорректных действий. Существует несколько классификаций для ограничений целостности, но для нас наиболее удобно классифицировать их по области действия.

Согласно вышеуказанной классификации все ограничения целостности базы данных можно разделить на 4 группы:

1. Ограничения атрибута;
2. Ограничения домена;
3. Ограничения кортежа;
4. Ограничения отношения.

Далее рассмотрим все типы ограничений целостности, применяемых в данном курсовом проекте (ограничения атрибута и отношения).

Ограничения атрибутаимеют большое значение при организации бизнес-логики системы. Одним из видов ограничения атрибутов является **ограничение уникальности (UNIQUE constraints)**. Еще одно название данного вида ограничения - альтернативный ключ (alternate key). В данном проекте этот вид ограничений широко использовался для поддержки целостности БД. Например, при анализе предметной области было выявлено, что название ЛПУ должно быть уникально. Поэтому при создании таблицы LPU был написан следующий сценарий.

CREATE TABLE LPU

(IDLPU INT IDENTITY PRIMARY KEY,

NameLPU varchar(50) **UNIQUE**,

MestoLPU varchar(30))

Создав данное отношение, мы установили, что название ЛПУ должно быть уникально. Таким образом, при попытке нарушить это ограничение пользователь получит сообщение об ошибке.

Ограничение UNIQUE было установлено в отношениях LPU, Vrach, Pacient, Type, Diagnos и других для обозначения потенциальных ключей отношений.

Еще одним видом ограничения атрибутов является **недопустимость NULL-значений**. Это означает, что данный атрибут не может иметь значение NULL (неопределённость). Это ограничение автоматически устанавливается для первичных ключей (Primary key) отношения, так как при значении первичного ключа NULL он перестаёт однозначно идентифицировать кортеж отношения. Можно также установить ограничение недопустимости NULL-значений на любой из других атрибутов. В данном курсовом проекте этот вид ограничения использовался очень широко, например:

CREATE TABLE Type

(IDType INT IDENTITY PRIMARY KEY,

NameType varchar(40) UNIQUE **NOT NULL**,

TarifType MONEY)

При создании таблицы Type (специальность врача) мы установили, что название специальности не может быть NULL, так как в противном случае теряется весь смысл данного отношения (название специальности является атрибутом, который несёт в себе наибольшую информативность для пользователя).

Установка ограничения NOT NULL была проведена во всех первичных и потенциальных ключах всех отношений, во всех внешних ключах, а также полях, которые несут наибольшую информативность в отношении.

Также при разработке базы данных было использовано **ограничение проверки атрибута** (**CHECK**). Надо заметить, что этот тип ограничения относится к ограничениям уровня отношения. Положительная особенность данного вида ограничений состоит в том, что их применение не ограничивается отдельными столбцами. В принципе можно проверить на соответствие определённому критерию любую комбинацию полей данной записи. В данном курсовом проекте ограничение значения использовалось в основном для атрибутов типа DateTime, чтобы исключить возможность ввода будущих дат. Для этого потребовалось написать следующий сценарий.

ALTER TABLE Isveshenie

WITH CHECK

ADD CONSTRAINT ChekDateBegin

CHECK (Isveshenie.BeginPer<GETDATE())

Ограничение CHECK было установлено для проверки правильности ввода дат в отношениях Prikas, Isveshenie, Karta.

Еще одним видом ограничений, возможно, самым важным, является **ограничение первичного ключа (PRIMARY KEY)**. Мы можем говорить о важности этого типа ограничений, так как реляционные базы данных создавались для реализации возможностей задания связей между данными. Поэтому важно иметь уникальные идентификаторы для каждого кортежа. Первичный ключ должен содержать уникальные значения (и поэтому не может содержать NULL-значений). Приведём пример использования в созданной базе данных ограничения первичного ключа.

CREATE TABLE Diagnos

(IDDiagnos INT IDENTITY **PRIMARY KEY**,

NameDiagnos varchar(50),

ShifrDiagnos varchar(20),

Norma INT )

Здесь можно добавить, что первичным ключом отношения Diagnos является атрибут IDDiagnos, таким образом он идентифицирует любой кортеж отношения.

Естественно, данный вид ограничений использовался во всех отношениях базы данных для обозначения первичного ключа.

Следующим типом ограничения является **ограничение внешнего ключа (FOREIGN KEY)**. Они используются как для обеспечения целостности данных, так и для задания отношений между таблицами. При этом после создания внешнего ключа любая запись, добавляемая в ссылочное отношение, должна иметь соответствующую запись в таблице, на которую существует ссылка, либо значения для столбцов внешнего ключа должны быть установлены в NULL. Последний случай в данном курсовом проекте не используется, так как это может привести к нарушению согласованного состояния базы данных. Поэтому приведем пример для рассматриваемого типа ограничений.

CREATE TABLE Otdel

(IDOtdel INT IDENTITY PRIMARY KEY,

Name varchar(30),

IDLPU INT,

TarifOtdel MONEY,

CONSTRAINT OtdelLPUforeign **FOREIGN KEY(IDLPU)** REFERENCES LPU)

После выполнении этого скрипта мы установили ограничение внешнего ключа для отношения Otdel. Теперь при попытке удаления ЛПУ, первичный ключ которого содержится в рассматриваемой таблице в качестве внешнего (IDLPU), будет выдано сообщение об ошибке.

Ограничения внешнего ключа мы установили в отношениях, которые являются ссылающимися (см. Приложение 1).

Поддержка целостности базы данных также осуществляется с помощью триггеров. **Триггер (Trigger)** представляет собой некую разновидность хранимой процедуры, которая выполняется при наступлении определенных событий. Триггеры очень помогают при реализации бизнес-правил. Например, в статистическом талоне может быть несколько диагнозов, но при этом только один из них может иметь тип «основной», а остальные - сопутствующий. Для поддержки в базе данных этого правила можно написать следующий триггер.

CREATE TRIGGER OsnovDiagnTalon

ON DiagnTalon

FOR INSERT, UPDATE

AS

IF ((SELECT COUNT(D.Type) FROM DiagnTalon D

INNER JOIN INSERTED I

ON I.IDTalon=D.IDTalon

WHERE D.Type = 'основной' AND D.IDTalon=I.IDTalon

GROUP BY D.IDTalon)<>1)

BEGIN

RAISERROR('Нельзя иметь больше одного основного диагноза в талоне!',16,1)

ROLLBACK TRAN

END

Теперь мы при добавлении или редактировании отношения DiagnTalon (Диагноз в талоне) SQL Server будет следить за тем, чтобы основной диагноз для определённого талона был только один. Точно так же реализовано правило, согласно которому в карте выбывшего больного должен быть только один основной диагноз.

При создании базы данных мы столкнулись также со следующей проблемой. Отношение Talon (Статистический талон) имеет атрибуты IDLPU, IDVrach, IDPacient, IDType, которые представляют собой внешние ключи на отношения LPU, Vrach, Type. Мы можем внести в базу данных информацию, согласно которой, например, врач А лечил в ЛПУ B в качестве специалиста C, но при этом врач А может не быть врачом ЛПУ B, и не быть обладать специальностью C. Чтобы исправить данный недостаток, были написаны соответствующие триггеры, приведём пример одного из них.

ALTER TRIGGER TalonVrachLPU

ON Talon

FOR INSERT, UPDATE

AS

IF (EXISTS(SELECT 'true' FROM INSERTED

WHERE INSERTED.IDVrach NOT IN (SELECT IDVrach FROM

Vrach

WHERE IDLPU=INSERTED.IDLPU)

))

BEGIN

RAISERROR('Такого врача нет в выбранном ЛПУ!',16,1)

ROLLBACK TRAN

END

Аналогичным образом написаны триггеры, запрещающие добавление или редактирование отношений Talon и Karta таким образом, чтобы информация в базе данных стала противоречивой (врач не принадлежащий данному ЛПУ или не обладающий данной специальностью).

В ходе анализа предметной области выяснилось, что нельзя вводить документацию по пациенту, который уже умер. Ведь умершие не могут посещать врачей или лежать в больнице. Для предотвращения ввода такой информации был написан триггер, запрещающий ввод данных по умершему пациенту в отношение Talon (Статистический талон), его сценарий приведён ниже:

CREATE TRIGGER StopSmert

ON Talon

FOR INSERT

AS

IF (EXISTS(SELECT 'true' FROM INSERTED

WHERE INSERTED.IDPacient IN (SELECT DISTINCT IDPacient FROM

Talon T

INNER JOIN DiagnTalon DT

ON T.IDTalon=DT.IDTalon

WHERE DT.Ishod='Смерть'

UNION

SELECT DISTINCT IDPacient FROM

Karta K

INNER JOIN DiagnKarta DK

ON K.IDKarta=DK.IDKarta

WHERE DK.Ishod='Смерть')

))

BEGIN

RAISERROR(‘Пациент умер!',16,1)

ROLLBACK TRAN

END

Аналогично написан триггер, запрещающий ввод информации по умершему в отношение Karta (Карта выбывшего больного).

Как уже было сказано, финансирование ЛПУ осуществляется на основании приказа об оплате. Но у специалиста не должно быть возможности удалить входную документацию (талоны и карты). Для этого был написан триггер, заперщающий удаление талонов и карт, по которым было проведено финансирование:

CREATE TRIGGER StopDelTalon

ON Talon

FOR DELETE

AS

IF (EXISTS( SELECT 'true' FROM DELETED

WHERE DELETED.Date BETWEEN (SELECT MAX(BeginPer) FROM Prikas WHERE BeginPer<DELETED.Date AND IDLPU=DELETED.IDLPU)

AND (SELECT MAX(BeginPer) FROM Prikas

WHERE EndPer>DELETED.Date AND IDLPU=DELETED.IDLPU)))

BEGIN

RAISERROR('Нельзя удалить талон, так как оплата по нему уже производилась!',16,1)

ROLLBACK TRAN

END

Аналогично был написан триггер, запрещающий удаление карт выбывших больных, по которым уже производилось финансирование.

Таким образом, при помощи ограничений и триггеров мы получили базу данных, обладающую свойством целостности.

## 3.2. Поддержание бизнес-логики

Поддержание бизнес-логики является еще одной задачей при разработке базы данных. Бизнес-логика - логика выполнения бизнес-процесса по определённым правилам, называемым еще бизнес-правилами. Так, при анализе предметной области из главного процесса «Учесть затраты на оказанные медицинские услуги» было выделено 5 подпроцессов:

* Фиксировать полученную документацию;
* Получить документацию;
* Сформировать заявку;
* Оформить платёжное поручение;
* Получить извещение об оплате.

База данных строилась таким образом, чтобы хранить всю входную и формировать выходную документацию, полученную в результате выполнения основного бизнес-процесса. В этой главе мы рассмотрим, какие объекты базы данных были созданы для обеспечения этого требования.

При рассмотрении первого подпроцесса (Фиксировать полученную документацию) появилась необходимость создания отношений, соответствующих входным потокам данных. Так появились отношения Talon и Karta с атрибутами, входящими в словарь данных (Курсовой проект по дисциплине ИТ). Создание отношения Talon осуществлялось следующим образом:

CREATE TABLE Talon

(Number INT PRIMARY KEY NOT NULL,

Date DATETIME,

Type varchar(30),

IDLPU INT,

IDVrach INT,

IDPacient INT,

IDType INT,

CONSTRAINT TalonLPUforeign FOREIGN KEY(IDLPU) REFERENCES LPU,

CONSTRAINT TalonVrachforeign FOREIGN KEY(IDVrach) REFERENCES Vrach,

CONSTRAINT TalonTypeforeign FOREIGN KEY(IDType) REFERENCES Type,

CONSTRAINT TalonPacientforeign FOREIGN KEY(IDPacient) REFERENCES Pacient)

Зесь следует отметить, что каждый статистический талон может содержать несколько диагнозов, таким образом появилось отношение DiagnTalon (Диагноз в талоне):

CREATE TABLE DiagnTalon

(IDTalon INT,

IDDiagnos INT,

Ishod varchar(30),

Type varchar(30),

CONSTRAINT PK\_Foreign PRIMARY KEY(IDTalon,IDDiagnos),

CONSTRAINT TalonDiagnforeign FOREIGN KEY(IDTalon) REFERENCES Talon **ON DELETE CASCADE**,

CONSTRAINT DiagnTalonforeign FOREIGN KEY(IDDiagnos) REFERENCES Diagnos)

Особенностью этого отношения является опция ON DELETE CASCADE при ограничении внешнего ключа. Это позволяет автоматически удалить все диагнозы при удалении какого-либо статистического талона. Такое удаление подобно реальному удалению документа, при этом все данные, связанные с ним, также удаляются. Конечно же, использование этой опции необходимо не особенно часто, лучшее применение она находит именно в слабых сущностях, как и в этом случае.

Аналогичным образом были созданы отношения Karta, Prikas и Isveshenie, которые предназначены для хранения информации из входной следующей документации - карта выбывшего больного, приказ и извещение об оплате. Входной поток информации *Нормы на лечение* материализовался в виде отношения Diagnos, где стала храниться информация о диагнозах и сроках их лечения. Создание отношения Diagnos:

CREATE TABLE Diagnos

(IDDiagnos INT IDENTITY PRIMARY KEY,

NameDiagnos varchar(50),

ShifrDiagnos varchar(20),

**Norma** INT )

Документ *Прейскурант цен на медицинские услуги* нашел свое выражение в отношениях Type (Специальность врача) и Otdel (Отделение), в которые в качестве атрибутов вошли тарифы по определённой специальности и отделению.

CREATE TABLE Otdel

(IDOtdel INT IDENTITY PRIMARY KEY,

Name varchar(30),

IDLPU INT,

**TarifOtdel** MONEY,

CONSTRAINT OtdelLPUforeign FOREIGN KEY(IDLPU) REFERENCES OtdelLPU)

Выходной документ *Платежное поручение* формируется с помощью хранимой процедуры на основании определённого приказа об оплате. Входными параметрами для процедуры являются NameLPU (Наименование ЛПУ) и Date (Дата приказа об оплате).

CREATE PROC Poruchenie

@NameLPU varchar(50),

@Date DateTime

AS

SELECT NameLPU,MestoLPU,Date,BeginPer,EndPer,Summa

FROM Prikas

INNER JOIN LPU

ON Prikas.IDLPU=LPU.IDLPU

WHERE LPU.NameLPU=@NameLPU AND Prikas.Date=@Date

Еще одним выходным документов автоматизируемого процесса является З*аявка на финансирование*, в которую входят суммы для финансирования отдельных ЛПУ за период. Входными параметрами для процедуры являются даты начала и конца периода. Сама же процедура реализована следующим образом: считаются суммы отдельно для стационара (в подзапросе) и для поликлиники, затем две полученные суммы складываются и дают таким образом окончательный результат. Для формирования заявки был написан следующий сценарий, создающий хранимую процедуру:

CREATE PROC GetZayavka

@Begin DateTime,

@End DateTime

AS

SELECT LPU.NameLPU,LPU.MestoLPU,SUM(TarifType)+Stacionar.Summa[Summa]

FROM Talon

INNER JOIN Type

ON Talon.IDType=Type.IDType

INNER JOIN LPU

ON Talon.IDLPU=LPU.IDLPU

INNER JOIN (SELECT LPU.IDLPU,SUM(TarifOtdel\*(Norma\*1.15))[Summa]

FROM Karta

INNER JOIN LPU

ON Karta.IDLPU=LPU.IDLPU

INNER JOIN OtdelLPU

ON OtdelLPU.IDLPU=LPU.IDLPU

INNER JOIN Otdel

ON Otdel.IDOtdel=OtdelLPU.IDOtdel

INNER JOIN DiagnKarta

ON Karta.IDKarta=DiagnKarta.IDKarta

INNER JOIN Diagnos

ON Diagnos.IDDiagnos=DiagnKarta.IDDiagnos

WHERE Karta.DateEnd BETWEEN @Begin AND @End AND DiagnKarta.Type='основной'

GROUP BY LPU.IDLPU) Stacionar

ON LPU.IDLPU=Stacionar.IDLPU

WHERE Talon.Date BETWEEN @Begin AND @End

GROUP BY LPU.NameLPU,LPU.MestoLPU,Stacionar.Summa

В ходе написания курсового проекта по дисциплине «Управление данными» были написаны запросы к базе данных. Часть из них нашла свое отражение при создании базы данных в виде хранимых процедур. Все запросы перечислять не имеет смысла, поэтому поясним одну из них. Приведённая ниже процедура возвращает фамилии, имена и отчества всех пациентов конкретного ЛПУ за период. Таким образом, входными параметрами являются наименование ЛПУ и даты начала и конца периода. В процедуре происходит объединение 3 выборок - выборки по дате начала из карты выбывшего больного, по дате конца из карты и по дате статистического талона. Понятно, что кроме этого в предложение WHERE включено условие отсева пациентов только по конкретному ЛПУ.

CREATE PROC PacientLPU

@NameLPU varchar(50),

@Begin DateTime,

@End DateTime

AS

SELECT Fam,Im,Otch

FROM Pacient

INNER JOIN Karta

ON Pacient.IDPacient=Karta.IDPacient

INNER JOIN LPU

ON Karta.IDLPU=LPU.IDLPU

WHERE LPU.NameLPU=@NameLPU AND Karta.DateEnd BETWEEN @Begin AND @End

UNION

SELECT Fam,Im,Otch

FROM Pacient

INNER JOIN Karta

ON Pacient.IDPacient=Karta.IDPacient

INNER JOIN LPU

ON Karta.IDLPU=LPU.IDLPU

WHERE LPU.NameLPU=@NameLPU AND Karta.DateBegin BETWEEN @Begin AND @End

UNION

SELECT Fam,Im,Otch

FROM Pacient

INNER JOIN Talon

ON Pacient.IDPacient=Talon.IDPacient

INNER JOIN LPU

ON Talon.IDLPU=LPU.IDLPU

WHERE LPU.NameLPU=@NameLPU AND Talon.Date BETWEEN @Begin AND @End

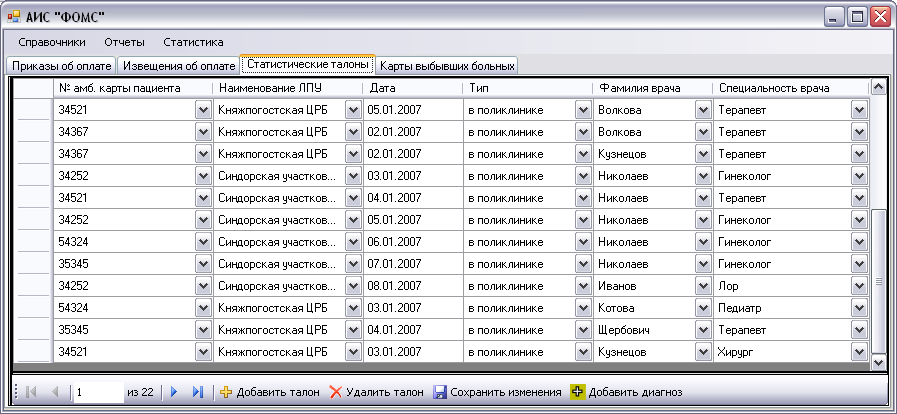
После написания этой процедуры появилась потребность написать подобные процедуры, а именно вывод пациентов за период без указания ЛПУ, с указанием отделения и ЛПУ, с указанием только отделения. Создание этих процедур позволило сделать первоначальный запрос гораздо более гибким и удобным для конечного пользователя.

Таким образом, с помощью созданных отношений, хранимых процедур и триггеров обеспечивается бизнес-логика, подобная логике выполнения реального бизнес-процесса.

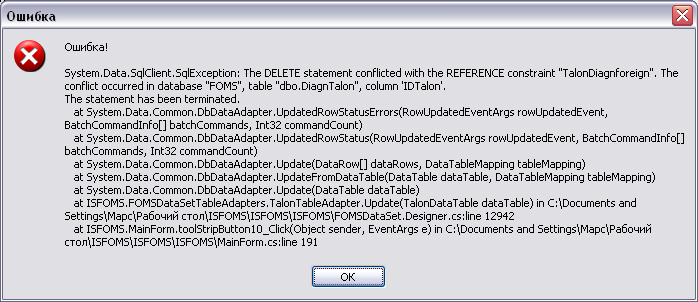
## 3.3. Описание интерфейса пользователя

Создание хорошего интерфейса – это очень тяжелая работа, т.к. необходимо уметь поставить себя на место пользователя и проанализировать с его точки зрения, каким должен быть интерфейс, чтобы ему было удобно использовать его в своей работе. Интерфейс должен быть таким, чтобы при использовании тех или иных функций программы повторялся автоматизируемый бизнес-процесс.

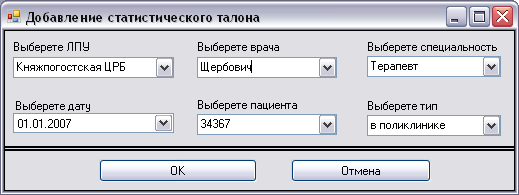
Учитывая это, при запуске созданного приложения пользователю предоставляется главное окно программы, где для начала работы ему необходимо выбрать одну из вкладок или необходимый пункт главного меню.



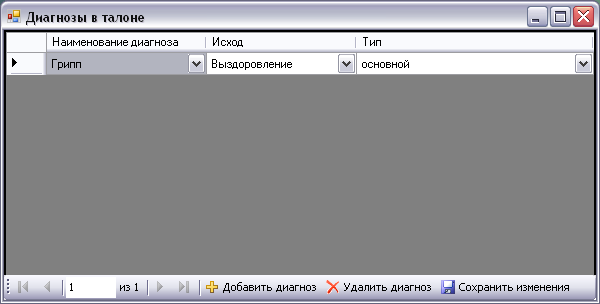
Как видно, рабочая область главного окна состоит из четырех вкладок. Эти вкладки соответствуют документам, которые наиболее часто фиксируются в системе. Внизу рабочей области окна мы видим инструмент для управления данными таблицы. Например, при нажатии на кнопку «Добавить талон» в таблице появится одна пустая строка, в которую будет необходимо внести требуемые данные. Программа предусматривает обработку исключений и выдачу соответствующих сообщений пользователю. Например, при удалении статистического талона, на который есть ссылки в других таблицах, пользователь получит сообщение, в котором указывается не только сам факт ошибки, но и её причина, которую пользователь может понять сам, либо позвать администратора базы данных или программиста.



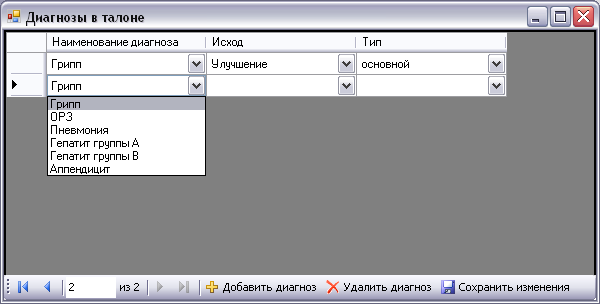
При необходимости добавления статистического талона или карты появляется следующее диалоговое окно:



При необходимости добавить в какой-либо талон диагноз этого талона, необходимо нажать кнопку «Добавить диагноз» или сделать двойной щелчок мышью по нужной строке, при этом появится следующее окно:

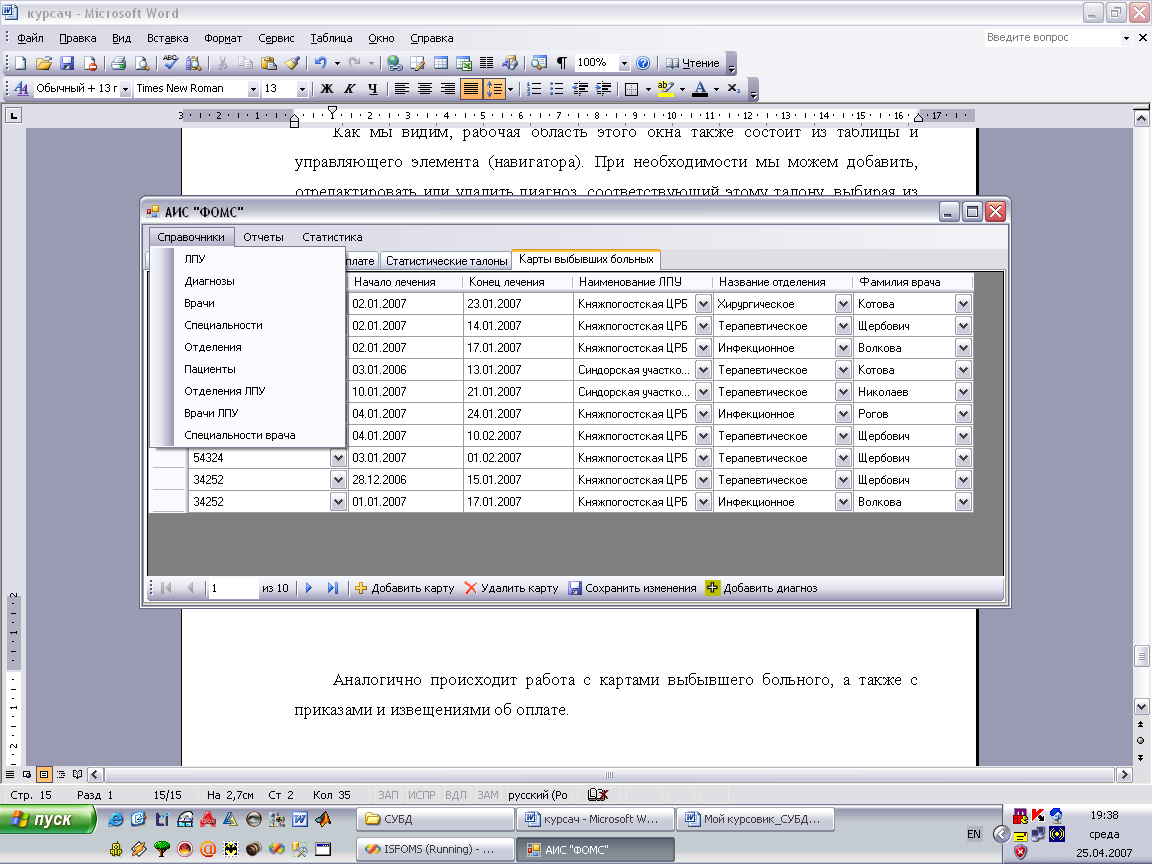


Как мы видим, рабочая область этого окна также состоит из таблицы и управляющего элемента (навигатора). При необходимости мы можем добавить, отредактировать или удалить диагноз, соответствующий этому талону, выбирая из выпадающих списков нужные нам значения.

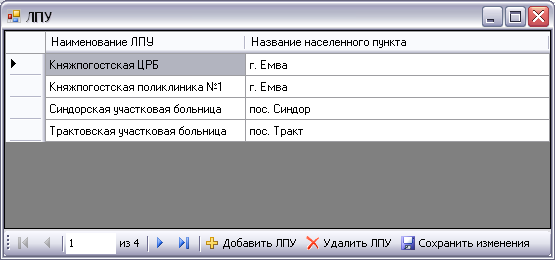


Аналогично происходит работа с картами выбывшего больного, а также с приказами и извещениями об оплате.

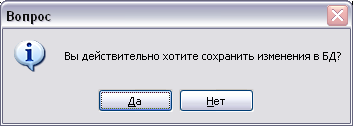
При выборе пункта главного меню «Справочники» мы увидим выпадающий список всех справочников системы:



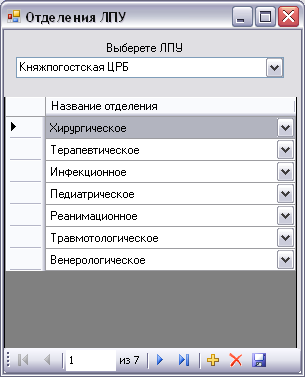
Следует отметить, что все справочники можно условно разделить на простые (первые пять в списке) и сложные (последние три). Работа с простыми справочниками осуществляется следующим образом: при выборе пункта меню происходит появление диалогового окна, также состоящего из таблицы и навигатора. Пользователь с помощью простых и интуитивно понятных действий может произвольным образом управлять данными таблицы.



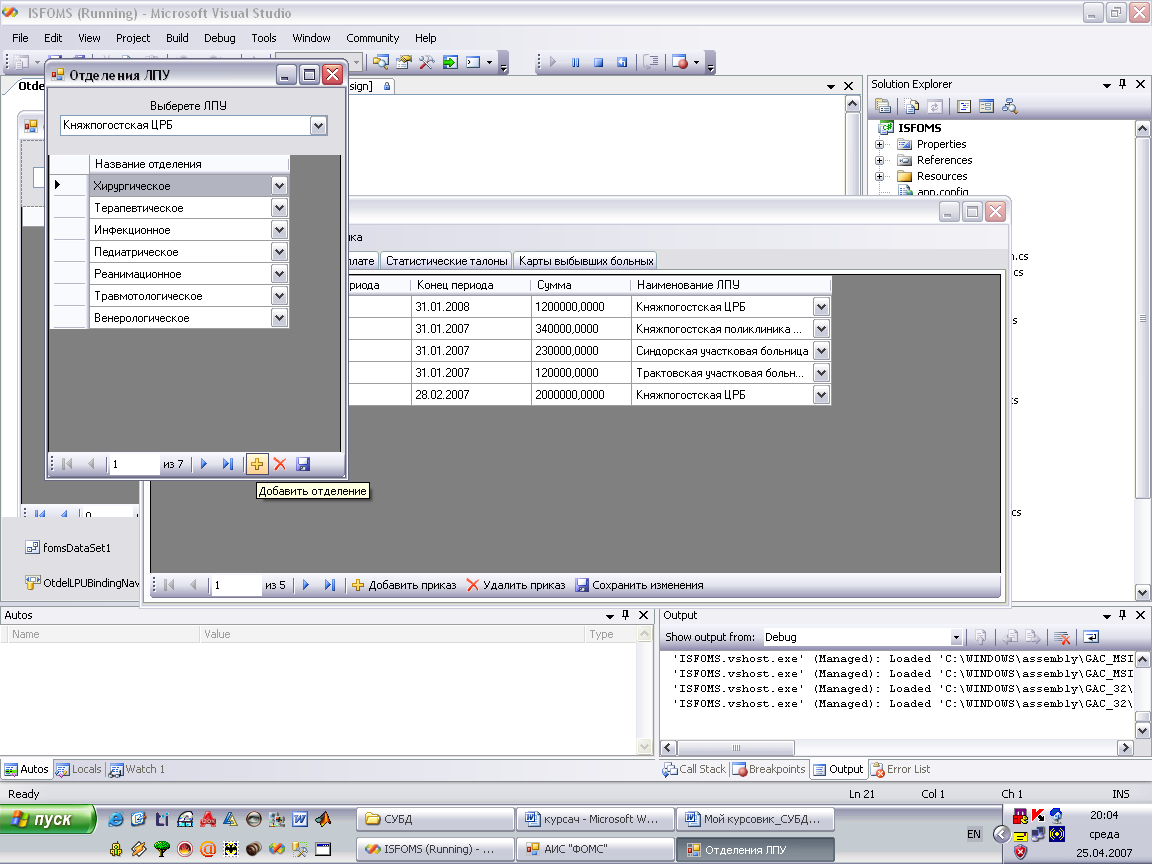
При попытке сохранения изменений в базе данных мы увидим следующее сообщение. Изменения фиксируются только в случае нажатия на кнопку «Да».



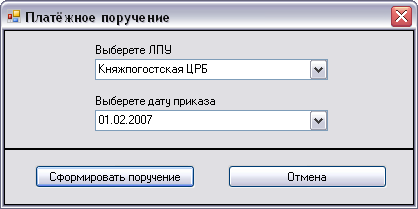
Несколько иначе происходит работа с так называемыми сложными справочниками. При выборе нужного пункта меню также происходит вызов диалогового окна, но с расположенным в верхней части рабочей области выпадающим списком.



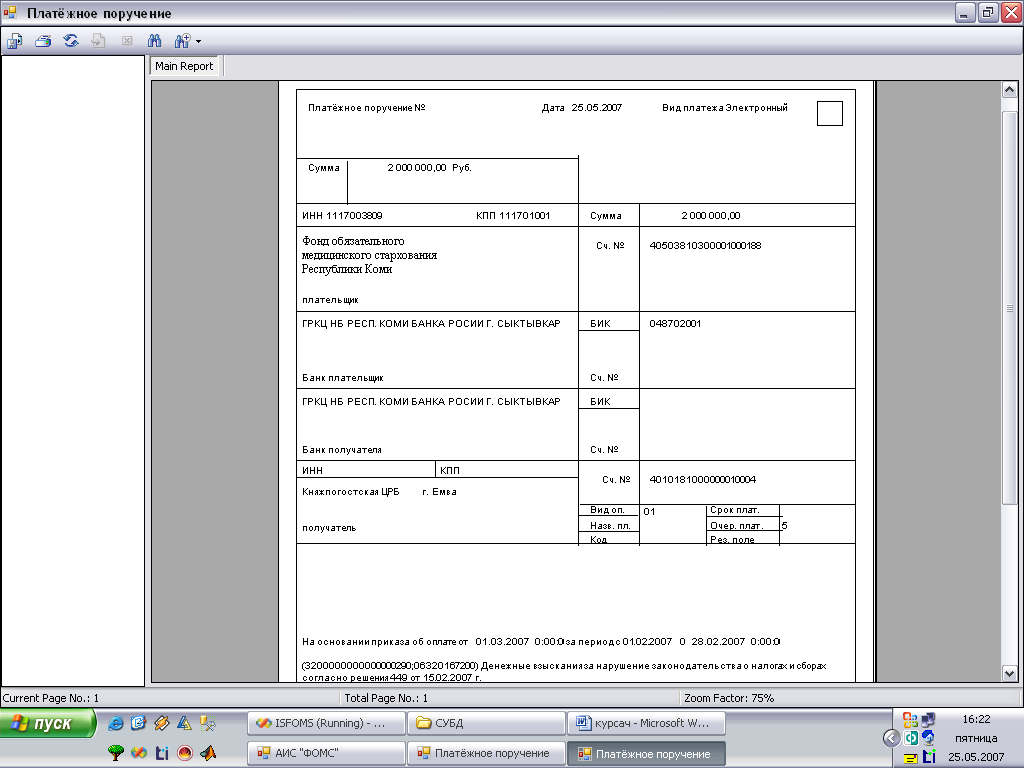
Работа с данными ведётся аналогично другим справочникам, но отображаются только те данные, которые соответствуют выбранному в списке элементу. Необходимо отметить, что все элементы навигаторов всех справочников снабжены всплывающими подсказками, существенно облегчающими работу пользователя.



Перейдем к элементу главному меню «Отчеты». Как мы видим, возможен выбор 2 отчетов - заявки на финансирование и платёжного поручения. При выборе одного из этих пунктов происходит вызов окна, в котором нужно указать требуемые данные, например:



В этом случае следует выбрать наименование ЛПУ и указать дату нужного приказа об оплате, а затем нажать кнопку «Сформировать поручение». Нажатие кнопки «Отмена» закрывает окно.

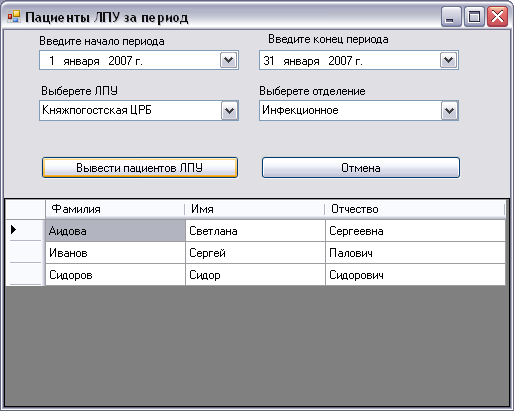


После формирования поручения пользователю доступны функции экспорта отчета в файлы с расширением \*.doc, \*.xls, \*.pdf и другие. Экспорт отчета происходит с помощью стандартного окна сохранения файла, в котором можно указать путь сохранения, имя и тип файла.

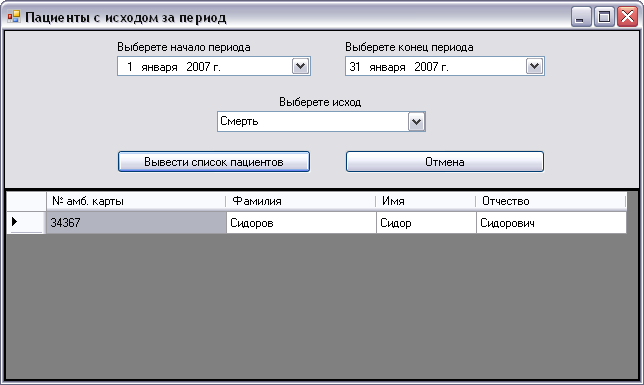


В отчете пользователю доступны также кнопки «Печать», «Обновить», «Поиск текста» и «Масштаб». Использование их подобно этим же функциям в любом стандартном Windows-приложении и интуитивно понятно пользователю, имеющему хотя бы минимальный опыт работы с Windows, поэтому подробное описание их не имеет смысла.

Третий элемент главного меню - статистика. В него входят запросы, которые нет необходимости выводить на печать (в виде отчетов), но которые могут понадобиться пользователю. Описывать все пункты не имеет смысла, так как их интерфейс в большинстве своём предусматривает выбор или ввод каких-то значений и нажатие на кнопку для вывода результатов запроса. Рассмотрим запрос, выдающий пациентов за период. При выборе этого пункта меню происходит вызов окна, состоящего из 2 логических частей - блока ввода информации и блока вывода. Блок ввода состоит из двух компонентов выбора дат - начала и конца периода, и двух выпадающих списков - для выбора ЛПУ и отделения. Надо отметить, что система предусматривает возможность оставлять списки пустыми, при этом отсев информации по соответствующему условию производиться не будет.



Приведём еще один пример - вывод списка пациентов с определённым исходом за выбранный период. Здесь также необходимо ввести значения - даты начала и конца периода, а также выбрать необходимый исход. После этого нажатие кнопки «Вывести список пациентов» выведет нам необходимую информацию, а нажатие кнопки отмена закроет данное окно.



Созданный интерфейс является очень удобным и понятным для пользователя. Это не пустые слова, так как он напоминает интерфейс многих Windows-приложений. Обработка ошибок и показ её причины, всплывающие подсказки, удобные кнопки, выпадающие списки, навигаторы для управления таблицами и сами таблицы являются стандартными элементами современных приложений. Интерфейс пользователя является важной частью при реализации системы, и мы можем с уверенностью сказать, что справились с этой задачей.

## 3.4. Формирование выходной документации и входных форм

Формирование выходной документации и входных форм является задачей, которая обеспечивает соблюдение функциональных требований к системе. Выходная документация представляет собой отчеты, которые пользователь может как выводить на печать, так и экспортировать в файл с расширениями \*.doc, \*.xls, \*.pdf и другие. В нашей системе отчеты были созданы в системе отчетов Crystal Reports. Эта система позволяет довольно легко создавать красивые и наглядные отчеты, в которые можно вставить диаграммы, графики, таблицы, формулы, и другие элементы.

В нашей системе выходная документация представляет собой два документа - платёжное поручение и заявка на финансирование. Информация, включаемая в отчет, выбирается из базы данных посредством хранимых процедур, их листинги приведены в разделе «Поддержание бизнес-логики», отметим лишь, что генерируемые системой отчеты полностью идентичны подлинным документам Княжпогостского филиала ФОМС.

Входные формы системы представляют собой двумерные таблицы. Эти таблицы связаны с соответствующими отношениями базы данных, что позволяет без проблем осуществлять обмен данными между визуальными компонентами (таблицами) и базой данных. С одной стороны, это предоставляет удобный и понятный для пользователя ввод информации, а с другой, предоставляет информацию для просмотра в наглядном виде. Как было отмечено в разделе «Описание интерфейса пользователя», большим удобством является наличие в некоторых столбцах таблицы выпадающих списков вместо обычных полей для ввода текста. Это позволяет пользователю быстро выбрать нужный элемент списка, а также исключает возможность ошибки при ручном вводе.

В заключение отметим, что созданные входные формы и генерируемые системой отчёты удобны для пользователя и подобны формам и документам реального бизнес-процесса, протекающего при учете затрат на медицинские услуги в Княжпогостском филиале ФОМС.

## 3.5. Пользователи и права доступа

В СУБД Microsoft SQL Server 2005 пользователи в значительной степени эквивалентны учётным записям. Иными словами, объект пользователя представляет собой идентификатор для некоторого лица, желающего войти в систему для работы. Любой, кто пожелает зарегистрироваться для работы с SQL Server 2005, должен быть представлен с помощью объекта пользователя. Пользователи, в свою очередь, могут принадлежать к одной или нескольким ролям. Права на выполнение определённых действий в СУБД SQL Server 2005 могут быть предоставлены непосредственно пользователю или роли, к которой могут относиться несколько пользователей.

В нашей базе данных не содержится каких-либо секретных данных, поэтому вполне достаточно для ограничения доступа к данным по сети использовать аутентификацию Windows, и создать соответствующего пользователя в качестве владельца базы данных. Надо сказать, что Microsoft SQL Server 2005 позволяет легко управлять пользователями и их правами, так что в случае необходимости администратор баз данных может легко внести изменения в существующую схему доступа к данным.

В клиентском приложении также нет необходимости организовывать какую-либо политику безопасности, так как система, по сути, является однопользовательской, причём пользователь должен иметь право совершать с данными любые действия в рамках приложения. А для защиты от несанкционированного доступа к данным через клиентское приложение вполне достаточно средств Windows.

## Заключение

Данный курсовой проект является продолжением курсового проекта по дисциплинам «Информационные технологии» и «Управление данными», так как главная цель данных работ касается автоматизации процесса по учету затрат на оказанные медицинские услуги.

В ходе выполнения данного курсового проекта была разработана физическая модель базы данных, причём в качестве целевой СУБД была выбрана Microsoft SQL Server 2005. С помощью наложенных на отношения ограничений и триггеров осуществлена поддержка декларативной целостности базы данных, а при помощи создания хранимых процедур были реализованы бизнес-правила. Реализация запросов к базе данных и создание хранимых процедур на языке T-SQL позволили автоматизировать процесс формирования выходной документации. Анализ входной документации позволил правильно создать входные формы для управления данными системы.

В результате проделанной работы мы получили практически законченную автоматизированную систему, которая позволяет решить ряд проблем рассматриваемой предметной области. Соответствие системы описанным в техническом задании требованиям позволяет сделать вывод, что созданная АИС при соответствующей доработке может быть применена в соответствующей предметной области, а именно в Княжпогостском филиале ФОМС.

## Список используемой литературы

1. Роберт Вьейра. SQL Server 2000. Программирование в 2 ч.: Пер. с англ.; Под ред. С.М. Молявко. – М.: БИНОМ. Лаборатория знаний, 2004. – 735 с.: ил.
2. Роберт Вьейра. Программирование баз данных Microsoft SQL Server 2005. Базовый курс. : Пер. с англ. - М. : ООО «И.Д. Вильямс», 2007. - 832 стр. : ил.
3. Коннолли Томас, Бегг Каролин. Базы данных: проектирование, реализация и сопровождение. Теория и практика, 3-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 1440 с.: ил. – Парал. тит. англ.;
4. Гектор Гарсиа-Молина, Джеффери Ульман, Дженнифер Уидом. Системы баз данных. Полный курс: Пер. с англ. – М.: Издательский дом “Вильямс”, 2004
5. Нейгел Кристиан, Ивьен Билл, Глин Джей и др. C# 2005 для профессионалов. : Пер с англ. - М. : ООО «И.Д. Вильямс», 2006. - 1376 с.
6. Николаева Н.А. Язык структурированных запросов. Лабораторные работы: учебное пособие / Н.А. Николаева, Т.Ю. Калинина. – Ухта: УГТУ, 2006. – 124 с. ил.

## Приложения

# Приложение 1



# Приложение 2



# Приложение 3

