МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Рыбинский государственный авиационный технический университет

имени П.А. Соловьева»

Авиационный колледж

**КУРСОВОЙ ПРОЕКТ**

по дисциплине

МДК 11.01 Технология разработки и защита баз данных

на тему

**БАЗА ДАННЫХ «ПОЛИКЛИНИКА»**

Студент группы ИС-3 Р.Н. Шибалов

Руководитель М.Н. Жебраков

Работа допущена к защите «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2024 г.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Рыбинск, 2024

График выполнения работы

«База данных "Поликлиника"»

|  |  |  |
| --- | --- | --- |
| №п/п | Наименование разделов (глав) курсового проекта | Дата окончания |
| 1 | Составление списка литературных источников |  |
| 2 | Описание предметной области |  |
| 3 | Построение логической модели |  |
| 4 | Создание БД |  |
| 5 | Заполнение БД |  |
| 6 | Создание триггеров к БД |  |
| 7 | Создание представлений |  |
| 8 | Создание хранимых процедур |  |
| 9 | Создание презентации |  |
| 10 | Оформление пояснительной записки |  |
| 11 | Защита курсового проекта |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Руководитель проекта\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г.

Задание к исполнению принял\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Рыбинский государственный авиационный технический университет

имени П.А. Соловьева»

Авиационный колледж

ЗАДАНИЕ

на курсовой проект

по дисциплине

МДК.11.01 Разработка администрирование и защита баз данных

по специальности

09.02.07. Информационные системы и программирование

студенту группы ИС-3 Шибалову Роману Николаевичу

1 Тема курсового проекта – База данных «ПОЛИКЛИНИКА»

Содержание:

1. Разработать БД ПОЛИКЛИНИКА. В базу данных медицинской поликлиники занесены имя, пол, дата рождения, паспортные данные, домашний адрес каждого пациента, а также сведения о врачах.

Всякий раз, когда врач осматривает больного, явившегося на прием, или сам приходит по вызову на дом, записываются имя и данные пациента, имя врача, дата и место проведения осмотра, симптомы заболевания, диагноз и предписания больному. Если врач прописывает больному лекарство, записывается его название, способ приема и дозировка, описание предполагаемого действия лекарства и возможных побочных эффектов. Система должна обеспечивать поиск и выдачу сведений по разным вопросам. Разработать БД «ПОЛИКЛИНИКА».

1. Срок сдачи студентом законченного курсового проекта «\_\_» \_\_\_\_\_\_ 20\_\_ г.
2. Исходные данные: отсутствуют.
3. Перечень подлежащих разработке задач/вопросов: изучение теоретического материала: атрибуты базы данных, даталогическая модель

Содержание

[Введение 6](#_Toc154585491)

[1 Проектная часть 8](#_Toc154585492)

[1.1 Описание предметной области 8](#_Toc154585493)

[1.2 Анализ атрибутов 9](#_Toc154585494)

[1.3 Создание информационно-логической модели предметной области в каноническом виде 11](#_Toc154585495)

[1.4 Создание даталогической модели реляционной базы данных 13](#_Toc154585496)

[1.5 Разработка данных контрольного примера 14](#_Toc154585497)

[2 Программные разработки 17](#_Toc154585498)

[2.1 Создание базы данных 17](#_Toc154585499)

[2.2 Заполнение данными 18](#_Toc154585500)

[2.3 Разработка триггеров 20](#_Toc154585501)

[2.4 Разработка представлений 21](#_Toc154585502)

[2.5 Разработка хранимых процедур 24](#_Toc154585503)

[Заключение 30](#_Toc154585504)

[Список используемых источников 31](#_Toc154585505)

[Приложение А 32](#_Toc154585506)

[Приложение Б 33](#_Toc154585507)

[Приложение В 34](#_Toc154585508)

[Приложение Г 36](#_Toc154585509)

[Приложение Д 38](#_Toc154585510)

# Введение

Современные медицинские учреждения сталкиваются с растущими требованиями к организации и ведению учета пациентов, что делает автоматизацию процесса управления особенно актуальной. Поликлиники играют ключевую роль в системе здравоохранения, обеспечивая первичную медицинскую помощь населению. В связи с увеличением объема информации, связанной с пациентами и врачами, разработка эффективной базы данных (БД) становится необходимостью.

Медицинская база данных поликлиники будет включать в себя обширную информацию о каждом пациенте: имя, пол, дата рождения, паспортные данные и домашний адрес. Также важна информация о врачах, что позволит составлять полные профили для каждого обращения. При каждом осмотре пациента фиксируются имя и данные больного, имя врача, дата и место проведения осмотра, а также симптомы, диагноз и предписания. В случае назначения лекарств будет записано их название, способ приема и дозировка, а также информация о предполагаемом действии и возможных побочных эффектах.

Эта система должна обеспечить удобный поиск и выдачу сведений по различным запросам, что значительно оптимизирует работу медицинского персонала и повысит качество обслуживания пациентов. Разработка БД «ПОЛИКЛИНИКА» станет важным шагом к улучшению управления медицинской информацией и повышению эффективности работы поликлиник.

Целью курсового проекта является разработка базы данных «ПОЛИКЛИНИКА», в которой будет храниться информация о пациентах, врачах, медицинских осмотрах и назначениях.

Для достижения данной цели были поставлены следующие задачи:

- изучить язык Transact-SQL и Microsoft SQL Server;

- изучить предметную область;

- проанализировать атрибуты предметной области;

- создать инфологическую и даталогическую модели данных;

- создать базу данных «ПОЛИКЛИНИКА».

# 1 Проектная часть

## 1.1 Описание предметной области

Концепция информационной системы для управления медицинской поликлиникой представляет собой важный инструмент, который позволяет эффективно организовать учет и управление данными о пациентах и врачах. Эта система предназначена для хранения информации о каждом пациенте, включая его имя, пол, дату рождения, паспортные данные и домашний адрес, а также профиль каждого врача.

Каждый раз, когда врач принимает пациента или выезжает на дом, фиксируются ключевые данные: имя и информация о пациенте, имя врача, дата и место осмотра, симптомы, диагноз и предписания. Если врач назначает лекарственные средства, система обеспечивает запись информации, такой как название препарата, способ приема, дозировка, предполагаемое действие и возможные побочные эффекты. Основная задача данной информационной системы — обеспечить возможность простого и быстрого доступа к медицинской информации, позволяя врачам и медперсоналу эффективно управлять данными.

Система также предоставляет функции поиска и выдачи сведений по различным запросам, что облегчает работу медицинского персонала и улучшает качество обслуживания пациентов. Важно, чтобы база данных была актуальной и могла легко обновляться, включая добавление новых пациентов, врачей, а также корректировку и удаление старых записей. Это позволит поликлинике оптимизировать процессы приема, лечения и назначения медикаментов, обеспечивая максимально эффективное взаимодействие с пациентами и поддержание высокого уровня медицинской помощи.

## 1.2 Анализ атрибутов

Проектирование базы данных начинается с создания всепокрывающего отношения, включающего все ключевые атрибуты и содержащего необходимые данные для выполнения задачи. Атрибут базы данных — это название колонки и информация, хранящаяся в соответствующих полях таблицы базы данных. Таблица представляет собой экземпляр правильного отношения, которое называется универсальным отношением проектируемой базы данных. В универсальное отношение включаются все ключевые атрибуты, и оно способно содержать все данные, которые планируется хранить в базе данных.

Нормализация данных, в свою очередь, представляет собой процесс организации и структурирования базы данных с целью устранения избыточной информации. Проще говоря, это способ обеспечить логическую организацию каждого поля и записи, чтобы не только избежать избыточности, но и сделать использование любой реляционной базы данных более эффективным: предотвращение ошибок при вводе данных, случайного удаления или облегчение процесса обновления информации. Нормализация данных подчиняется определенным правилам, которые определяют, как должна быть организована база данных.

Первая нормальная форма диктует, что каждое поле базы данных должно хранить одно значение и что в одной базе данных не должно быть двух полей, одинаково хранящих информацию.

Вторая нормальная форма направлена на уменьшение избыточности, гарантируя, что каждое поле хранит информацию, которая говорит нам что-то о первичном ключе. Другими словами:

- Каждая база данных должна иметь только один первичный ключ;

- Все не первичные ключи должны полностью зависеть от первичного ключа.

Эти два принципа обеспечивают целостность данных в базе, гарантируя, что информация об одном элементе, хранящаяся в первичном ключе, представлена последовательно. База данных считается соответствующей третьей нормальной форме, если в ней отсутствуют переходные зависимости. Это означает, что данные организованы таким образом, чтобы значения одного столбца не зависели от другого столбца, который в свою очередь зависит от первичного ключа. Чтобы достичь соответствия базы данных третьей нормальной форме, необходимо убрать любые столбцы, которые зависят от данных, не связанных напрямую с первичным ключом, и хранить такие сведения в другой базе данных с собственным ключом. Помимо описанных нормальных форм, существуют четвертая и пятая нормальные формы, но их применение в реальной практике не так распространено.

Проектирование базы данных «ПОЛИКЛИНИКА» начинается с определения основных объектов и их атрибутов, которые должны храниться в системе. В базе данных учитываются пациенты, врачи, консультации, назначения лекарств, симптомы, диагнозы и сведения, обеспечивающие управление данными о медицинских осмотрах. Каждая таблица в БД представляет собой отдельную сущность, хранящую структурированную информацию с соблюдением принципов нормализации.

Сущность «Passports» предназначена для хранения паспортных данных. Атрибуты данной таблицы представлены в таблице 1.1

Таблица 1.1 – Атрибуты сущности «Passports»

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Характеристика |
| Id | INT | Идентификатор паспорта |
| Series | NVARCHAR(10) | Серия паспорта |
| Number | NVARCHAR(20) | Номер паспорта |
| Surname | NVARCHAR(30) | Фамилия |
| Name | NVARCHAR(30) | Имя |
| FatherName | NVARCHAR(30) | Отчество |
| IssuingAuthority | NVARCHAR(80) | Орган, выдавший паспорт |
| DateOfIssue | DATE | Дата выдачи паспорта |
| DateOfExpiry | DATE | Дата окончания действия паспорта |
| Nationality | NVARCHAR(50) | Национальность |
| PlaceOfBirth | NVARCHAR(100) | Место рождения |
| Sex | NCHAR(1) | Пол |
| IssuingRegion | NVARCHAR(100) | Регион выдачи |
| DateOfBirth | DATE | Дата рождения |

Сущность «Patients» предназначена для хранения данных о пациентах. Атрибуты данной таблицы представлены в таблице 1.2

Таблица 1.2 – Атрибуты сущности «Patients»

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Характеристика |
| Id | INT | Идентификатор пациента |
| PassportId | INT | Идентификатор паспорта |
| CreatedAt | DATETIME2(7) | Дата создания записи |
| UpdatedAt | DATETIME2(7) | Дата обновления записи |

Сущность «Doctors» предназначена для хранения данных о докторах. Атрибуты данной таблицы представлены в таблице 1.3

Таблица 1.3 – Атрибуты сущности «Doctors»

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Характеристика |
| Id | INT | Идентификатор врача |
| Specialty | NVARCHAR(50) | Специализация врача |
| Phone | NVARCHAR(20) | Телефон врача |
| Email | NVARCHAR(100) | Электронная почта врача |
| ExperienceYears | INT | Количество лет опыта |
| Address | NVARCHAR(200) | Адрес врача |
| CreatedAt | DATETIME2(7) | Дата создания записи |
| UpdatedAt | DATETIME2(7) | Дата обновления записи |
| PassportId | INT | Идентификатор паспорта врача |

Таблица «WorkingHours» предназначена для хранения данных о рабочем графике докторов. Атрибуты данной таблицы представлены в таблице 1.4

Таблица 1.4 – Атрибуты сущности «WorkingHours»

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Характеристика |
| Id | INT | Идентификатор рабочего времени |
| StartTime | TIME(7) | Время начала работы |
| EndTime | TIME(7) | Время окончания работы |
| CreatedAt | DATETIME2(7) | Дата создания записи |
| UpdatedAt | DATETIME2(7) | Дата обновления записи |
| DoctorId | INT | Идентификатор врача |
| WorkDay | NVARCHAR(11) | Рабочий день |

Таблица «DozeUnits» предназначена для хранения мер дозировки медицинских препаратов. Атрибуты данной таблицы представлены в таблице 1.5

Таблица 1.5 – Атрибуты сущности «DozeUnits»

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Характеристика |
| Id | INT | Идентификатор единицы дозировки |

Окончание таблицы 1.5

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Характеристика |
| Name | NVARCHAR(20) | Наименование единицы дозировки |

Таблица «MedicineUsingWays» предназначена для хранения путей приема медицинских препаратов. Атрибуты сущности «MedicineUsingWays» представлены в таблице 1.6

Таблица 1.6 – Атрибуты сущности «MedicineUsingWays»

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Характеристика |
| Id | INT | Идентификатор способа применения |
| Name | NVARCHAR(20) | Наименование способа применения |

Таблица «MedicineSideEffects» предназначена для хранения данных о побочных эффектах медицинских препаратов. Атрибуты данной таблицы представлены в таблице 1.7

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Характеристика |
| Id | INT | Идентификатор побочного эффекта |
| Name | NVARCHAR(50) | Наименование побочного эффекта |
| Description | NVARCHAR(200) | Описание побочного эффекта |
| MedicineId | INT | Идентификатор лекарства |

Таблица «Medicines» предназначена для хранения данных о медицинских препаратах. Атрибуты данной таблицы представлены в таблице 1.8

Таблица 1.8 – Атрибуты сущности «Medicines»

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Характеристика |
| Id | INT | Идентификатор лекарства |
| Name | NVARCHAR(100) | Наименование лекарства |
| DozeUnitId | INT | Идентификатор единицы дозировки |
| MedicineUsingWayId | INT | Идентификатор способа применения |

Таблица «Prescriptions» предназначена для хранения данных о выписанных рецептах. Атрибуты данной таблицы представлены в таблице 1.9

Таблица 1.9 – Атрибуты сущности «Prescriptions»

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Характеристика |
| Id | INT | Идентификатор назначения |
| Description | NVARCHAR(200) | Описание назначения |

Окончание таблицы 1.9

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Характеристика |
| TimesInDay | INT | Частота применения в день |
| DozeCount | FLOAT | Количество дозировки |
| CreatedAt | DATETIME2(7) | Дата создания назначения |
| MedicineId | INT | Идентификатор лекарства |
| ConsultationId | INT | Идентификатор консультации |

Таблица «Consultations» предназначена для хранения информации о врачебных приемах. Атрибуты данной таблицы представлены в таблице 1.10

Таблица 1.10 – Атрибуты сущности «Consultations»

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Характеристика |
| Id | INT | Идентификатор консультации |
| Location | NVARCHAR(100) | Место проведения консультации |
| StartTime | DATETIME2(7) | Время начала консультации |
| EndTime | DATETIME2(7) | Время окончания консультации |
| DoctorId | INT | Идентификатор врача |
| PatientId | INT | Идентификатор пациента |

Таблица «Symptoms» предназначена для хранения симптомов. Атрибуты данной таблицы представлены в таблице 1.11

Таблица 1.11 – Атрибуты сущности «Symptoms»

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Характеристика |
| Id | INT | Идентификатор симптома |
| Name | NVARCHAR(100) | Наименование симптома |

Таблица «ConsultationSymptoms» предназначена для хранения выявленных симптомов на приеме. Атрибуты данной таблицы представлены в таблице 1.12

Таблица 1.12 – Атрибуты сущности «ConsultationSymptoms»

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Характеристика |
| Id | INT | Идентификатор записи симптома |
| SymptomId | INT | Идентификатор симптома |
| ConsultationId | INT | Идентификатор консультации |

Таблица «Roles» предназначена для хранения ролей пользователей. Атрибуты данной таблицы представлены в таблице 1.13

Таблица 1.13 – Атрибуты сущности «Roles»

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Характеристика |
| Id | INT | Идентификатор роли |
| Name | NVARCHAR(20) | Наименование роли |
| CreatedAt | DATETIME2(7) | Дата создания роли |

Таблица «Users» предназначена для хранения информации о пользователях. Атрибуты данной таблицы представлены в таблице 1.14

Таблица 1.14 – Атрибуты сущности «Users»

|  |  |  |
| --- | --- | --- |
| Атрибут | Тип данных | Характеристика |
| Id | INT | Идентификатор пользователя |
| Name | NVARCHAR(20) | Имя пользователя |
| Password | NCHAR(60) | Пароль (хэшированный) |
| Email | NVARCHAR(100) | Электронная почта |
| Phone | NVARCHAR(20) | Телефон пользователя |
| CreatedAt | DATETIME2(7) | Дата создания записи |
| UpdatedAt | DATETIME2(7) | Дата обновления записи |
| RoleId | INT | Идентификатор роли |

## 

## 1.3 Создание информационно-логической модели предметной области в каноническом виде

Инфологическая модель данных — это своего рода макет будущей базы данных, который представляет информацию о данных, используя различные инструменты, такие как текстовые описания, диаграммы, таблицы и графики. Она служит для описания информации и ее организации в базе данных, обеспечивая понимание структуры данных как для разработчиков, так и для конечных пользователей.

Концептуальная модель данных отражает сущности и связи между ними в базе данных. Сущности представляют объекты или события, которые моделируются в системе и имеют определенные характеристики, выраженные через атрибуты. Эти атрибуты могут быть использованы для однозначной идентификации каждого экземпляра сущности. Концептуальная модель фокусируется на ключевых аспектах данных и их взаимосвязях, предоставляя абстрактное представление информации.

Методология IDEF1X используется для моделирования реляционных баз данных. Она предлагает специальный синтаксис для создания концептуальных схем баз данных. В IDEF1X сущности описываются как наборы объектов с общими атрибутами. Связи между сущностями отражают отношения между ними и могут быть разных типов, например, один-ко-многим или многие-ко-многим.

Преимущества IDEF1X включают в себя строгие стандарты моделирования, что помогает уменьшить неоднозначность в интерпретации структуры базы данных. Этот подход способствует единообразному пониманию данных и облегчает коммуникацию между разработчиками и пользователями.

Связи в IDEF1X представляют собой ссылки, соединения и ассоциации между сущностями. Связи – глаголы, которые показывают, как относятся сущности между собой. Связи могут быть нескольких видов: один ко одному, один ко многим, многие ко многим. Чаще используется связь вида один ко многим. Такие связи отображаются в виде линии между двумя сущностями с точкой на одном конце и глагольной фразой, отображаемой над линией. Отношение (связи) многие ко многим обычно используются на начальной стадии разработки диаграммы и отображаются в IDEF1X в виде сплошной линии с точками на обоих концах.

Основным преимуществом IDEF1X, по сравнению с другими многочисленными методами разработки реляционных баз данных является жесткая и строгая стандартизация моделирования. Установленные стандарты позволяют избежать различной трактовки построения модели.

На основе полученных сущностей и их атрибутов построим инфологическую модель базы данных «ПОЛИКЛИНИКА». На рисунке 1.1 изображена инфологическая модель базы данных «ПОЛИКЛИНИКА» методом IDEF1X.

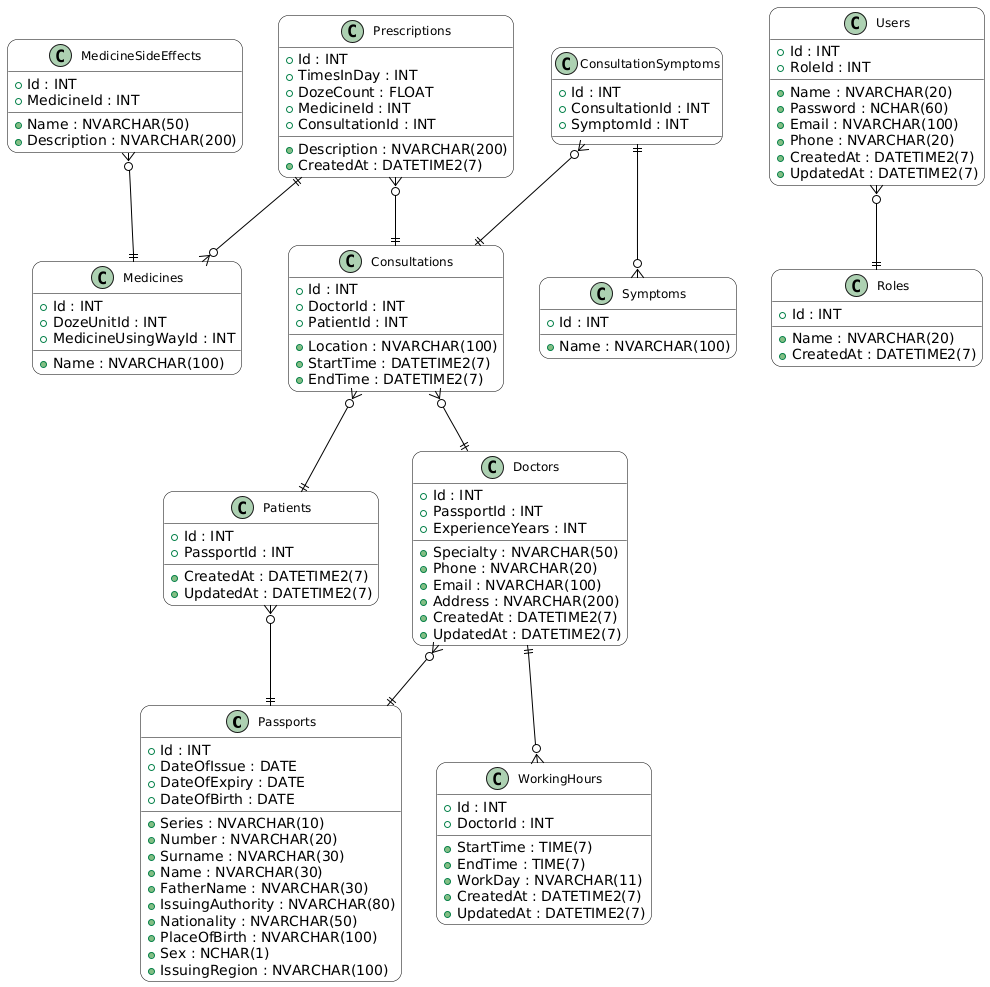


Рисунок 1.1 – Инфологическая модель базы данных «ПОЛИКЛИНИКА»

## 1.4 Создание даталогической модели реляционной базы данных

Даталогическая модель данных — это представление логической структуры базы данных в рамках конкретной системы управления базами данных (СУБД). Она отображает связи между элементами базы данных, учитывая особенности и возможности конкретной СУБД.

Для создания даталогической модели часто используют инфологическую модель, которая представляет собой описание структуры данных на уровне предметной области.

Проектирование даталогической модели для реляционной базы данных включает разделение всей информации на отдельные файлы или таблицы. В каждом файле определяется набор полей, которые отображаются как атрибуты в реляционной модели. Файлы соответствуют таблицам, а поля – атрибутам этих таблиц в базе данных.

На основе инфологической модели базы данных построена даталогическая модель базы данных «ПОЛИКЛИНИКА». Даталогическая модель базы данных «ПОЛИКЛИНИКА» представлена в приложении А.

## 1.5 Разработка данных контрольного примера

Под текстовым (контрольным) примером понимается совокупность исходных (текстовых) данных, для которых осуществляется полная апробация алгоритма. К исходным данным предъявляются следующие требования:

- небольшой объем, чтобы решение было как можно более быстрым;

- полнота, т.е. обязательно проверки всех предусмотренных разветвлений вычислительного процесса и обработки нестандартных ситуаций;

- реалистичной по форме и содержанию информации о той предметной области, для которой предназначена зада.

Таким образом, решение на основе контрольного примера является о определенным эталоном для проверки корректности алгоритма, реализованного в программном продукте. Если верный результат получается для контрольного примера, то алгоритм работает правильно и его можно применять для любой другой вариативности исходных данных.

Для каждой таблицы создадим CSV-файл, в котором будут храниться данные для последующей вставки базы данных.

CSV-файл – это обычный текстовый файл, в котором хранятся таблицы и данные электронных таблиц. Содержимое часто представляет собой таблицу текста, цифр или дат. CSV-файл можно легко импортировать и экспортировать с помощью программ, хранящих данные в таблицах.

Контрольные данные для сущности «Товар» представлены в таблице 1.12.

Таблица 1.12 – Контрольные данные примера для сущности «Товар»

|  |  |  |  |
| --- | --- | --- | --- |
| Код товара | Наименование | Количество | Единицы измерения |
| 1 | Футболка | 1000 | Штуки |
| 2 | Кофе | 3000 | Килограммы |
| 3 | Молоко | 4500 | Литры |
| 4 | Ручка | 6000 | Штуки |
| 5 | Мыло | 3000 | Штуки |
| 6 | Карандаш | 6000 | Штуки |
| 7 | Ластик | 6000 | Штуки |
| 8 | Свитер | 891 | Штуки |
| 9 | Ботинки | 400 | Штуки |
| 10 | Чай | 4000 | Килограммы |

Контрольные данные для сущности «Товары в магазинах» представлены в таблице 1.13.

Таблица 1.13 – Контрольные данные примера для сущности «Товары в магазинах»

|  |  |  |  |
| --- | --- | --- | --- |
| Код продукта в магазине | Код продукта | Код магазина | Количество |
| 1 | 1 | 1 | 50 |
| 2 | 2 | 1 | 20 |
| 3 | 3 | 1 | 30 |
| 4 | 4 | 1 | 100 |
| 5 | 5 | 1 | 70 |
| 6 | 2 | 3 | 145 |
| 7 | 6 | 4 | 150 |
| 8 | 1 | 10 | 200 |
| 9 | 10 | 3 | 120 |
| 10 | 3 | 4 | 110 |

Контрольные данные для сущности «Магазины» представлены в таблице 1.14.

Таблица 1.14 – Контрольные данные примера для сущности «Магазины»

|  |  |
| --- | --- |
| Код магазина | Наименование |
| 1 | Магнит |

Окончание таблицы 1.14

|  |  |
| --- | --- |
| Код магазина | Наименование |
| 2 | Дикси |
| 3 | Чижик |
| 4 | Бристоль |
| 5 | Пятерочка |
| 6 | Ашан |
| 7 | FIXPRICE |
| 8 | Макси |
| 9 | Лента |
| 10 | Чижик |

Контрольные данные для сущности «Заказы магазинов на год» представлены в таблице 1.15.

Таблица 1.15 – Контрольные данные примера для сущности «Заказы магазинов на год»

|  |  |  |  |
| --- | --- | --- | --- |
| Код заказа | Код магазина | Код продукта | Количество |
| 1 | 1 | 1 | 100 |
| 2 | 2 | 3 | 200 |
| 3 | 3 | 2 | 150 |
| 4 | 4 | 4 | 300 |
| 5 | 5 | 5 | 120 |
| 6 | 6 | 10 | 345 |
| 7 | 7 | 5 | 342 |
| 8 | 8 | 2 | 321 |
| 9 | 9 | 4 | 123 |
| 10 | 10 | 10 | 687 |

# 2 Программные разработки

## 2.1 Создание базы данных

Создание базы данных в SQL Server осуществляется с помощью команды CREATE DATABASE, где указывается уникальное имя для новой базы данных, соответствующее правилам идентификаторов SQL Server. Создание базы происходит в системной базе master, которая содержит необходимые системные таблицы для работы самой системы баз данных.

Для создания таблиц используется инструкция CREATE TABLE в языке T-SQL. Эти инструкции могут быть встроены в процедуру.

Целостность данных обеспечивается с помощью ограничений целостности. Ограничение NOT NULL позволяет определить, можно ли хранить в столбце NULL значения или нет. PRIMARY KEY задает первичный ключ таблицы, а FOREIGN KEY устанавливает связь между данными в разных таблицах, обеспечивая ссылочную целостность.

Ограничение FOREIGN KEY предотвращает наличие записей в таблице, которые ссылаются на несуществующие записи в другой таблице. Оно гарантирует корректность данных, предотвращая некорректные ссылки между записями.

UNIQUE - это ограничение, которое гарантирует уникальность значений в столбце или комбинации столбцов.

Таблицы на языке Transact-SQL создаются при помощи инструкции CREATE TABLE. Листинг всех таблиц представлен в приложении Б.

На рисунке 2.1.1 изображена таблица «Товар» в Microsoft SQL Server



Рисунок 2.1- Таблица «Товар»

Аналогичные действия проводятся для создания всех таблиц БД «Снабжение магазинов»

На рисунке 2.2 изображена диаграмма базы данных БД «Снабжение магазинов».

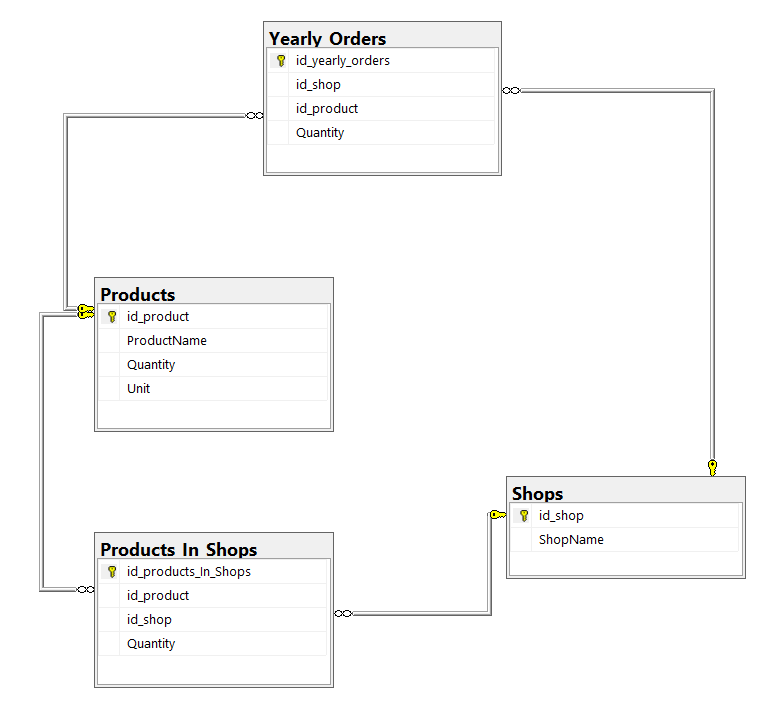


Рисунок 2.2 – Диаграмма базы данных «Снабжение магазинов»

## 2.2 Заполнение данными

Созданные таблицы необходимо заполнить данными из CSV-файлов, которые были созданы ранее.

Чтобы импортировать данные из CSV-файла, создадим временные таблицы для каждой сущности и сохраним в них данные. Во временных таблицах данные будут сохранены в виде строк, а после этого перенесены в основные таблицы и приведены к нужному нам типу данных.

На рисунке 2.3 изображена таблица «Товар» в CSV-файле.

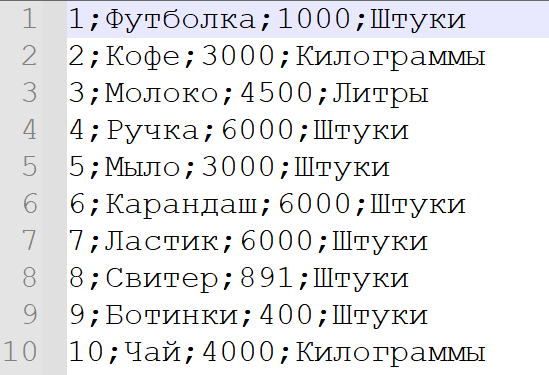


Рисунок 2.3 – Таблица «Товар» в CSV-файле

Для создания временных таблиц используется системная база данных – tempdb, в которой хранятся все временные объекты. База данных tempdb используется всеми базами данных системы. Содержимое этой базы данных удаляется при каждом перезапуске системы.

Инструкция BULK INSERT выполняет импорт файла данных в таблицу или представление базы данных в формате, указанном пользователем. В предложении WITH для данной инструкции можно задавать множество опций.

После вставки данных во временную таблицу заполним ими основную таблицу при помощи команды INSERT INTO. В тех случаях, когда необходимо выполнить преобразование от типов с высшим приоритетом к типа с низшим приоритетом, то надо выполнять явное приведение типов. Функция CAST преобразует выражение одного типа к другому.

На рисунке 2.4 изображена заполненная таблица «Товар» в Microsoft SQL Server

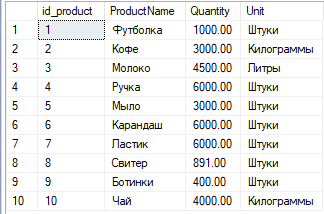


Рисунок 2.4 – Заполненная таблица «Товар»

Аналогичные действия проводятся для всех таблиц БД «Снабжение магазинов»

Листинг импорта данных из CSV-файла в базу данных «Снабжение магазинов» представлен в Приложении В.

## 2.3 Разработка триггеров

Триггер – это механизм, который вызывается, когда в указанной таблице происходит определенное действие. Каждый триггер имеет следующие основные составляющие:

- имя;

- действие;

- исполнение.

Триггер создается с помощью инструкции CREATE TRIGGER.

Можно задать тип триггера с помощью двух дополнительных параметров: AFTER и INSTEAD OF. Триггеры типа AFTER вызываются после выполнения действия, запускающего триггер, а триггеры типа INSTEAD OF выполняются вместо действия, запускающего триггера.

Параметры INSERT, UPDATE и DELETE задают действия триггера. Под действием триггера имеются ввиду инструкция Transact-SQL, которая запускает триггер. Допускается любая комбинациях этих трех инструкций.

Триггер типа AFTER можно использовать для выполнения следующих операций:

- создание журнала аудита действий в таблицах базы данных;

- реализация бизнес-правил;

- принудительное обеспечение ссылочной целостности.

Листинг создания триггеров для базы данных «Снабжение магазинов» представлен в приложении Г.

На рисунке 2.3.1 представлена ошибка о неправильно заполненных данных в таблице «Заказы на год»

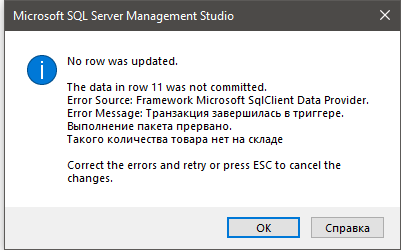


Рисунок 2.5 – Ошибка заполнения поля Количество товара в таблице «Заказы на год»

## 2.4 Разработка представлений

Представление – это виртуальная таблица, содержимое которой определяется запросом. Как и таблица, представление состоит из ряда именованных столбцов и строк данных. Пока представление не будет проиндексировано, оно не существует в базе данных как хранимая совокупность значений. Строки и столбцы данных извлекаются из таблиц, указанных в определяющем представление запросе и динамически создаваемых при обращениях к представлению.

Представление выполняет функцию фильтра базовых таблиц, на которые оно ссылается. Определяющий представление запрос может быть инициирован в одной или нескольких таблицах или в других представлениях текущей или других баз данных. Кроме того, для определения представлений с данными из нескольких разнородных источников можно использовать распределенные запросы. Это полезно, например, если нужно объединить структурированные подобным образом данные, относящиеся к разным серверам, каждый из которых хранит данные конкретного отдела организации.

Представление обычно используется для направления, упрощения и настройки восприятия каждым пользователем информации базы данных. Представления могут использоваться как механизмы безопасности, давая возможность пользователям обращаться к данным через представления, но не предоставляя им разрешений на непосредственный доступ к базовым таблицам, лежащих на основе представлений. Представления могут использоваться для обеспечения интерфейса обратной совместимости, моделирующего таблицу, которая существует, но схема которой изменилась. Представления могут также использоваться при прямом и обратном копировании данных в SQL Server для повышения производительности и секционирования данных.

Общий синтаксис для создания представлений:

CREATE VIEW view\_name [(column\_list)]

[WITH {ENCRYPTION SCHEMABINDING | VIEW\_METADATA}]

AS select\_statement

[WITH CHECK OPTION]

Для БД снабжения магазинов необходимо создать несколько представлений. Важно учитывать, какие товары находятся в каких магазинах и в каком количестве.

С помощью инструкции CREATE VIEW нужно создать представление ProductsInShopsView. Далее представлен код для создания данного представления:

USE WholesaleBase

GO

--Представление на отображение товаров в магазинах

CREATE VIEW ProductsInShopsView AS

SELECT PS.id\_products\_In\_Shops, S.ShopName, P.ProductName, PS.Quantity AS QuantityInShop, P.Unit

FROM Products\_In\_Shops PS

INNER JOIN Products P ON PS.id\_product = P.id\_product

INNER JOIN Shops S ON PS.id\_shop = S.id\_shop;

После создания представления нужно обратиться к нему инструкцией SELECT:

USE WholesaleBase

SELECT \* FROM ProductsInShopsView

После этого в окне результатов появится таблица представления, это изображено на рисунке 2.6.

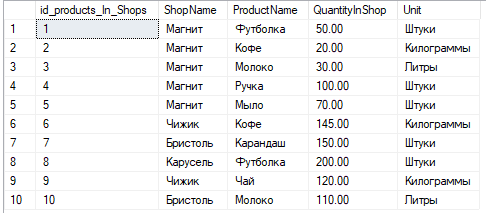


Рисунок 2.6 – представление для вывода определенных товаров в каждом магазине в определенном количестве

Так же важно учитывать какие магазины заказали определенные товары ив каком количестве. Для этого создадим представление YearlyOrdersWithProductDetailsView. Далее представлен код для создания данного представления:

USE WholesaleBase

GO

--Представление заказы на год с деталями о товарах

CREATE VIEW YearlyOrdersWithProductDetailsView AS

SELECT YO.id\_yearly\_orders, S.ShopName, P.ProductName, YO.Quantity AS OrderedQuantity, P.Unit

FROM Yearly\_Orders YO

INNER JOIN Products P ON YO.id\_product = P.id\_product

INNER JOIN Shops S ON YO.id\_shop = S.id\_shop;

После создания представления нужно обратиться к нему инструкцией SELECT:

USE WholesaleBase

SELECT \* FROM YearlyOrdersWithProductDetailsView

Далее в окне результатов появится таблица представления, это показано на рисунке 2.7.



Рисунок 2.7 – представление для вывода данных о заказах магазинов

## 2.5 Разработка хранимых процедур

Хранимая процедура в SQL Server представляет собой группу из одного или нескольких операторов Transact-SQL или ссылку на метод CLR Microsoft .NET Framework. Процедуры аналогичны конструкциям в других языках программирования, поскольку обеспечивают следующее:

- обрабатывают входные параметры и возвращают вызывающей программе значения в виде выходных параметров;

- содержат программные инструкции, которые выполняют операции в базе данных, включая вызов других процедур;

- возвращают значение состояния вызывающей программе, таким образом передавая сведения об успешном или неуспешном завершении (и причины последнего).

Преимущества использования хранимых процедур:

- снижение сетевого трафика между клиентами и сервером Команды в процедуре выполняются как один пакет кода. Это позволяет существенно сократить сетевой трафик между сервером и клиентом, поскольку по сети отправляется только вызов на выполнение процедуры. Без инкапсуляции кода, предоставляемой процедурой, по сети бы пришлось пересылать все отдельные строки кода;

- большая безопасность. Многие пользователи и клиентские программы могут выполнять операции с базовыми объектами базы данных посредством процедур, даже если у них нет прямых разрешений на доступ к базовым объектам. Процедура проверяет, какие из процессов и действий могут выполняться, и защищает базовые объекты базы данных. Это устраняет необходимость предоставлять разрешения на уровне индивидуальных объектов и упрощает формирование уровней безопасности;

- при вызове процедуры через сеть виден только вызов на выполнение процедуры. Таким образом, злоумышленники не могут просматривать имена объектов таблиц и баз данных, внедрять собственные инструкции Transsact-SQL или выполнять поиск критически важных данных;

- повторное использование кода. Если какой-то код многократно используется в операции базы данных, то отличным решением будет произвести его инкапсуляцию в процедуры. Это устранит необходимость излишнего копирование того же кода, снизит уровень несогласованности когда и позволит осуществлять доступ к коду любым пользователем или приложением, имеющим необходимые разрешения.

- более легкое обслуживание. Если клиентские приложения вызывают процедуры, а операции без данных остаются лишь на уровне данных, то для внесения изменений в основную базу данных будет достаточно обновить только процедуры. Уровень приложения остается незатронутым изменениями в схемах баз данных, связях или процессах;

- повышение производительности. По умолчанию компиляция процедуры и создание плана выполнения, используемого для последующих выполнений, производится при ее первом запуске. Поскольку обработчику запросов не нужно создавать новый план, обычно обработка процедур занимает меньше времени.

Хранимые процедуры создаются посредством инструкции CREATE PROCEDURE, которая имеет следующий синтаксис:

CREATE PROC[EDURE] [schema\_name.]proc\_name

[({@param} type1 [VARYING] [=default] [OUTPUT])] {, …}

[WITH {RECOMPILE | ENCRYPTION | EXECUTE AS ‘user\_name’}]

[FOR REPLICATION]

AS batch | EXTERNAL NAME method\_name

Жизненный цикл хранимой процедуры состоит из двух этапов: ее создания и ее выполнения. Каждая процедура создается один раз, а выполняется многократно. Хранимая процедура выполняется посредством инструкции EXECUTE пользователем, который является владельцем процедуры или обладает правом EXECUTE для доступа к этой процедуре (см. главу 12). Инструкция EXECUTE имеет следующий синтаксис:

[[EXEC[UTE]] [@return\_status =] {proc\_name

l@proc\_name\_var}

{[[@parameter1=]value[[@parameter1=]@variable[OUTPUT]][DEFAULT}… [WITH RECOMPILE]

Для удобства использования базы данных нужны хранимые процедуры для поиска информации в базе, вставку данных в таблицы и создания архива данных.

С помощью инструкции CREATE PROCEDURE нужно создать хранимую процедуру GetShopAndQuantityByProduct, которая выводит информацию по конкретному товару. Далее представлен код создания данной процедуры:

USE WholesaleBase

GO

--Хранимая процедура для вывода магазинов и количества товара, в которых есть определенный товар

CREATE PROCEDURE GetShopAndQuantityByProduct

@ProductName NVARCHAR(100)

AS

BEGIN

SELECT s.ShopName, pis.Quantity AS QuantityInShop

FROM Shops s

INNER JOIN Products\_In\_Shops pis ON s.id\_shop = pis.id\_shop

INNER JOIN Products p ON pis.id\_product = p.id\_product

WHERE p.ProductName = @ProductName;

END;

С помощью инструкции CREATE PROCEDURE нужно создать хранимую процедуру LoadDataFromCSV, которая вставляет данные в таблицы из CSV файлов. Листинг создания данной процедуры представлен в приложении Д.

Также необходимо создать хранимую процедуру по созданию архива данных. С помощью инструкции CREATE PROCEDURE нужно создать хранимую процедуру Arch\_Table\_Del, которая создаст таблицы, в которые будут сохраняться удаленные данные. Далее представлен код создания данной процедуры:

USE WholesaleBase

--Создание хранимой процедуры для создания архивных таблиц для товаров и заказов на год

GO

CREATE PROCEDURE Arch\_Table\_Del

AS

BEGIN

IF OBJECT\_ID('Arch\_Products\_Del') is null

CREATE TABLE Arch\_Products\_Del (

id\_product INT,

ProductName NVARCHAR(100),

Quantity DECIMAL(18, 2),

Unit NVARCHAR(50),

DeletedDateTime DATETIME

)

IF OBJECT\_ID('Arch\_Yearly\_Orders\_Del') is null

CREATE TABLE Arch\_Yearly\_Orders\_Del (

id\_yearly\_orders INT,

id\_shop INT,

id\_product INT,

Quantity DECIMAL(18, 2),

DeletedDateTime DATETIME);

END;

После удачного создания хранимых процедур они появятся в одноименном пункте MS SQL (см. рисунок 2.8)

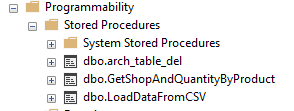


Рисунок 2.8 – созданные хранимые процедуры в MS SQL

В результате программных разработок была создана БД снабжения магазинов, ее таблицы. Таблицы заполнены корректными данными. Также разработаны триггера, представления и хранимые процедуры.

# Заключение

В ходе выполнения курсовой работы была разработана база данных «Снабжение магазинов» на языке Transact-SQL.

При этом решены следующие задачи:

* изучены синтаксис языка Transact-SQL и СУБД Microsoft SQL Server;
* проанализирована предметная область базы данных и ее атрибуты;
* выполнено построение информационно-логической и даталогической модели базы данных;
* разработаны входные данные для таблиц БД;
* выполнено создание и заполнение БД;
* созданы триггеры, необходимые для БД;
* созданы представления для БД;
* созданы хранимые процедуры для БД;

Для реализации основной цели курсовой работы была использована СУБД Miscoroft SQL Server.

Полученный программный продукт готов к эксплуатации и может быть использован различными оптовыми базами для учета, хранения и вывода различной информации.

В настоящее время для торговли оптовые базы просто необходимы для координации потоков товаров в магазины и поэтому всегда существует необходимость создания новых элементов и операций, поэтому в базук данных могут быть внесены изменения.

# Список используемых источников

* 1. Кореньков В.В. Технологии баз данных. Проектирование реляционных баз данных – Москва : Курс, 2022. – 128 с.
  2. Уолтер Шилдс. SQL: быстрое погружение.
  3. Алан Бьюли. Изучаем SQL, 2007.
  4. Иванов А. Д. Основы проектирования реляционных баз данных.
  5. Петров С. Л. Основы SQL: Руководство по реляционным базам данных.
  6. Джо Селко. SQL для умников: Advanced SQL Programming.
  7. Учебник с теорией и задачами по изучению SQL Academy [Электронный ресурс] URL: <https://sql-academy.org/ru>
  8. Смирнов И. В. Реляционные базы данных: теория и практика.
  9. Сидорова Е. И. СУБД и язык SQL в проектировании информационных систем.
  10. Козлов А. Д. Основы работы с базами данных: практическое руководство.

# Приложение А

Даталогическая модель базы данных:

Таблица Passports (хранение паспортных данных)

Первичный ключ: Id

Поля:

Id (INT) – Идентификатор паспорта

Series (NVARCHAR(10)) – Серия паспорта

Number (NVARCHAR(20)) – Номер паспорта

Surname (NVARCHAR(30)) – Фамилия

Name (NVARCHAR(30)) – Имя

FatherName (NVARCHAR(30)) – Отчество

IssuingAuthority (NVARCHAR(80)) – Орган, выдавший паспорт

DateOfIssue (DATE) – Дата выдачи паспорта

DateOfExpiry (DATE) – Дата окончания действия паспорта

Nationality (NVARCHAR(50)) – Национальность

PlaceOfBirth (NVARCHAR(100)) – Место рождения

Sex (NCHAR(1)) – Пол

IssuingRegion (NVARCHAR(100)) – Регион выдачи

DateOfBirth (DATE) – Дата рождения

Таблица Patients (данные о пациентах)

Первичный ключ: Id

Внешний ключ: PassportId → Passports.Id (NOT NULL, CASCADE UPDATE, RESTRICT DELETE)

Поля:

Id (INT) – Идентификатор пациента

PassportId (INT) – Идентификатор паспорта

CreatedAt (DATETIME2(7)) – Дата создания записи

UpdatedAt (DATETIME2(7)) – Дата обновления записи

Таблица Doctors (данные о врачах)

Первичный ключ: Id

Внешний ключ: PassportId → Passports.Id (NOT NULL, CASCADE UPDATE, RESTRICT DELETE)

Поля:

Id (INT) – Идентификатор врача

Specialty (NVARCHAR(50)) – Специализация

Phone (NVARCHAR(20)) – Телефон врача

Email (NVARCHAR(100)) – Электронная почта

ExperienceYears (INT) – Количество лет опыта

Address (NVARCHAR(200)) – Адрес

CreatedAt (DATETIME2(7)) – Дата создания записи

UpdatedAt (DATETIME2(7)) – Дата обновления записи

PassportId (INT) – Идентификатор паспорта

Таблица WorkingHours (рабочее время врачей)

Первичный ключ: Id

Внешний ключ: DoctorId → Doctors.Id (NOT NULL, CASCADE UPDATE, RESTRICT DELETE)

Поля:

Id (INT) – Идентификатор записи

StartTime (TIME(7)) – Время начала работы

EndTime (TIME(7)) – Время окончания работы

CreatedAt (DATETIME2(7)) – Дата создания записи

UpdatedAt (DATETIME2(7)) – Дата обновления записи

DoctorId (INT) – Идентификатор врача

WorkDay (NVARCHAR(11)) – Рабочий день

Таблица DozeUnits (единицы дозировки)

Первичный ключ: Id

Поля:

Id (INT) – Идентификатор дозировки

Name (NVARCHAR(20)) – Наименование дозировки

Таблица MedicineUsingWays (способы применения лекарств)

Первичный ключ: Id

Поля:

Id (INT) – Идентификатор способа применения

Name (NVARCHAR(20)) – Наименование способа применения

Таблица Medicines (данные о лекарствах)

Первичный ключ: Id

Внешние ключи:

DozeUnitId → DozeUnits.Id (NOT NULL, CASCADE UPDATE, RESTRICT DELETE)

MedicineUsingWayId → MedicineUsingWays.Id (NOT NULL, CASCADE UPDATE, RESTRICT DELETE)

Поля:

Id (INT) – Идентификатор лекарства

Name (NVARCHAR(100)) – Наименование лекарства

DozeUnitId (INT) – Идентификатор единицы дозировки

MedicineUsingWayId (INT) – Идентификатор способа применения

Таблица Prescriptions (выписанные рецепты)

Первичный ключ: Id

Внешние ключи:

MedicineId → Medicines.Id (NOT NULL, CASCADE UPDATE, RESTRICT DELETE)

ConsultationId → Consultations.Id (NOT NULL, CASCADE UPDATE, RESTRICT DELETE)

Поля:

Id (INT) – Идентификатор назначения

Description (NVARCHAR(200)) – Описание назначения

TimesInDay (INT) – Частота применения в день

DozeCount (FLOAT) – Количество дозировки

CreatedAt (DATETIME2(7)) – Дата создания назначения

Таблица Consultations (врачебные приемы)

Первичный ключ: Id

Внешние ключи:

DoctorId → Doctors.Id (NOT NULL, CASCADE UPDATE, RESTRICT DELETE)

PatientId → Patients.Id (NOT NULL, CASCADE UPDATE, RESTRICT DELETE)

Поля:

Id (INT) – Идентификатор консультации

Location (NVARCHAR(100)) – Место проведения консультации

StartTime (DATETIME2(7)) – Время начала консультации

EndTime (DATETIME2(7)) – Время окончания консультации

Таблица Symptoms (симптомы)

Первичный ключ: Id

Поля:

Id (INT) – Идентификатор симптома

Name (NVARCHAR(100)) – Наименование симптома

Таблица ConsultationSymptoms (выявленные симптомы)

Первичный ключ: Id

Внешние ключи:

SymptomId → Symptoms.Id (NOT NULL, CASCADE UPDATE, RESTRICT DELETE)

ConsultationId → Consultations.Id (NOT NULL, CASCADE UPDATE, RESTRICT DELETE)

Поля:

Id (INT) – Идентификатор записи

SymptomId (INT) – Идентификатор симптома

ConsultationId (INT) – Идентификатор консультации

Таблица Roles (роли пользователей)

Первичный ключ: Id

Поля:

Id (INT) – Идентификатор роли

Name (NVARCHAR(20)) – Наименование роли

CreatedAt (DATETIME2(7)) – Дата создания роли

Таблица Users (пользователи)

Первичный ключ: Id

Внешний ключ: RoleId → Roles.Id (NOT NULL, CASCADE UPDATE, RESTRICT DELETE)

Поля:

Id (INT) – Идентификатор пользователя

Name (NVARCHAR(20)) – Имя пользователя

Password (NCHAR(60)) – Пароль (хэшированный)

Email (NVARCHAR(100)) – Электронная почта

Phone (NVARCHAR(20)) – Телефон пользователя

CreatedAt (DATETIME2(7)) – Дата создания записи

UpdatedAt (DATETIME2(7)) – Дата обновления записи

RoleId (INT) – Идентификатор роли

Копировать код

# Приложение Б

Листинг создания таблиц для базы данных:

USE WholesaleBase

--Создаем таблицу Товары

CREATE TABLE Products (

id\_product INTEGER IDENTITY (1, 1) CONSTRAINT PK\_Product PRIMARY KEY,

ProductName NVARCHAR(100) UNIQUE NOT NULL,

Quantity DECIMAL (18, 2),

Unit NVARCHAR(50) NOT NULL -- Единица измерения (штуки, литры, килограммы и т.д.)

);

--Создаем таблицу Магазины

CREATE TABLE Shops (

id\_shop INTEGER IDENTITY (1, 1) CONSTRAINT PK\_Store PRIMARY KEY,

ShopName NVARCHAR(100) NOT NULL

);

--Создаем таблицу Продукты в магазинах

CREATE TABLE Products\_In\_Shops (

id\_products\_In\_Shops INTEGER IDENTITY (1, 1) CONSTRAINT PK\_Products\_In\_Shops PRIMARY KEY,

id\_product INT,

id\_shop INT,

Quantity DECIMAL(18, 2), -- Количество товара в магазине

CONSTRAINT FK\_Products\_In\_Shops\_Products FOREIGN KEY (id\_product) REFERENCES Products(id\_product)

ON DELETE NO ACTION

ON UPDATE CASCADE,

CONSTRAINT FR\_Products\_In\_Shops\_Shops FOREIGN KEY (id\_shop) REFERENCES Shops(id\_shop)

ON DELETE NO ACTION

ON UPDATE CASCADE

);

--Создаем таблицу Заказы на год

CREATE TABLE Yearly\_Orders (

id\_yearly\_orders INTEGER IDENTITY (1, 1) CONSTRAINT PK\_Yearly\_Orders PRIMARY KEY,

id\_shop INT,

id\_product INT,

Quantity DECIMAL(18, 2), -- Количество товара, запрошенное магазином

CONSTRAINT FK\_Yearly\_Orders\_Products FOREIGN KEY (id\_product) REFERENCES Products(id\_product)

ON DELETE NO ACTION

ON UPDATE CASCADE,

CONSTRAINT FK\_Yearly\_Orders\_Shops FOREIGN KEY (id\_shop) REFERENCES Shops(id\_shop)

ON DELETE NO ACTION

ON UPDATE CASCADE

);

# Приложение В

Листинг для импорта данных из CSV файлов в базу данных:

USE WholesaleBase;

GO

--Products

SET IDENTITY\_INSERT dbo.Products ON;

IF (OBJECT\_ID('tempdb.dbo.#csv\_tab\_products') IS NOT NULL)

DROP TABLE #csv\_tab\_products;

CREATE TABLE #csv\_tab\_products (

pole1 NVARCHAR(max),

pole2 NVARCHAR(max),

pole3 NVARCHAR(max),

pole4 NVARCHAR(max)

);

BULK INSERT #csv\_tab\_products

FROM 'C:\Program Files (x86)\Microsoft SQL Server\csv\Products.csv'

WITH (

FIELDTERMINATOR = ';',

ROWTERMINATOR = '\n'

);

INSERT INTO dbo.Products (id\_product, ProductName, Unit)

SELECT

CAST(pole1 AS INT),

CAST(pole2 AS NVARCHAR(25)),

CAST(pole3 AS DECIMAL(18, 2)),

CAST(pole4 AS NVARCHAR(20))

FROM #csv\_tab\_products;

SET IDENTITY\_INSERT dbo.Products OFF;

--Shops

SET IDENTITY\_INSERT dbo.Shops ON;

IF (OBJECT\_ID('tempdb.dbo.#csv\_tab\_shops') IS NOT NULL)

DROP TABLE #csv\_tab\_shops;

CREATE TABLE #csv\_tab\_shops (

pole1 NVARCHAR(max),

pole2 NVARCHAR(max)

);

BULK INSERT #csv\_tab\_shops

FROM 'C:\Program Files (x86)\Microsoft SQL Server\csv\Shops.csv'

WITH (

FIELDTERMINATOR = ';',

ROWTERMINATOR = '\n'

);

INSERT INTO dbo.Shops (id\_shop, ShopName)

SELECT

CAST(pole1 AS INT),

CAST(pole2 AS NVARCHAR)

FROM #csv\_tab\_shops;

SET IDENTITY\_INSERT dbo.Shops OFF;

--Products\_In\_Stores

SET IDENTITY\_INSERT dbo.Products\_In\_Shops ON;

IF (OBJECT\_ID('tempdb.dbo.#csv\_tab\_products\_in\_shops') IS NOT NULL)

DROP TABLE #csv\_tab\_products\_in\_shops;

CREATE TABLE #csv\_tab\_products\_in\_shops (

pole1 NVARCHAR(max),

pole2 NVARCHAR(max),

pole3 NVARCHAR(max),

pole4 NVARCHAR(max)

);

BULK INSERT #csv\_tab\_products\_in\_shops

FROM 'C:\Program Files (x86)\Microsoft SQL Server\csv\Products\_In\_Shops.csv'

WITH (

FIELDTERMINATOR = ';',

ROWTERMINATOR = '\n'

);

INSERT INTO dbo.Products\_In\_Shops (id\_products\_In\_Shops, id\_product, id\_shop, Quantity)

SELECT

CAST(pole1 AS INT),

CAST(pole2 AS INT),

CAST(pole3 AS INT),

CAST(pole4 AS DECIMAL)

FROM #csv\_tab\_products\_in\_shops;

SET IDENTITY\_INSERT dbo.Products\_in\_shops OFF;

--Yearly\_Orders

SET IDENTITY\_INSERT dbo.Yearly\_Orders ON;

IF (OBJECT\_ID('tempdb.dbo.#csv\_tab\_yearly\_orders') IS NOT NULL)

DROP TABLE #csv\_tab\_yearly\_orders;

CREATE TABLE #csv\_tab\_yearly\_orders (

pole1 NVARCHAR(max),

pole2 NVARCHAR(max),

pole3 NVARCHAR(max),

pole4 NVARCHAR(max)

);

BULK INSERT #csv\_tab\_yearly\_orders

FROM 'C:\Program Files (x86)\Microsoft SQL Server\csv\Yearly\_Orders.csv'

WITH (

FIELDTERMINATOR = ';',

ROWTERMINATOR = '\n'

);

INSERT INTO dbo.Yearly\_Orders(id\_yearly\_orders, id\_shop, id\_product, Quantity)

SELECT

CAST(pole1 AS INT),

CAST(pole2 AS INT),

CAST(pole3 AS INT),

CAST(pole4 AS DECIMAL)

FROM #csv\_tab\_yearly\_orders;

SET IDENTITY\_INSERT dbo.Yearly\_Orders OFF;

# Приложение Г

Листинг создания триггеров:

USE WholesaleBase

GO

-- Триггер на уменьшение товара на складе при добавлении товара в годовой заказ магазина

CREATE TRIGGER ReduceStockOnOrder

ON Yearly\_Orders

AFTER INSERT, UPDATE

AS

BEGIN

DECLARE @id\_product INT, @quantity DECIMAL(18, 2);

SELECT @id\_product = inserted.id\_product, @quantity = inserted.Quantity

FROM inserted;

DECLARE @current\_stock DECIMAL(18, 2);

SELECT @current\_stock = p.Quantity

FROM Products p

WHERE p.id\_product = @id\_product;

IF (@quantity > @current\_stock)

BEGIN

ROLLBACK TRANSACTION;

PRINT 'Такого количества товара нет на складе';

END

ELSE

BEGIN

UPDATE Products

SET Quantity = Quantity - @quantity

WHERE id\_product = @id\_product;

END

END;

--Триггер на обеспечение целостности при удалении магазина из таблицы Магазины

GO

CREATE TRIGGER DeleteFromShops

ON Shops

AFTER DELETE

AS

BEGIN

DELETE FROM Products\_In\_Shops

WHERE id\_shop IN (SELECT id\_shop FROM deleted);

DELETE FROM Yearly\_Orders

WHERE id\_shop IN (SELECT id\_shop FROM deleted);

END;

--Триггер на добавление товаров на склад при удалении заказа

GO

CREATE TRIGGER EnsureReferentialIntegrity

ON Yearly\_Orders

AFTER DELETE

AS

BEGIN

DECLARE @id\_product INT, @quantity DECIMAL(18, 2);

-- Получаем товары, удаленные из таблицы Yearly\_Orders

DECLARE order\_cursor CURSOR FOR

SELECT id\_product, Quantity

FROM deleted;

OPEN order\_cursor;

FETCH NEXT FROM order\_cursor INTO @id\_product, @quantity;

WHILE @@FETCH\_STATUS = 0

BEGIN

-- Обновляем количество товаров на складе (Products)

UPDATE Products

SET Quantity = Quantity + @quantity

WHERE id\_product = @id\_product;

FETCH NEXT FROM order\_cursor INTO @id\_product, @quantity;

END;

CLOSE order\_cursor;

DEALLOCATE order\_cursor;

END;

GO

--Триггер на добавление удаленных товаров в архив

CREATE TRIGGER ArchiveDeletedProducts

ON Products

AFTER DELETE

AS

BEGIN

INSERT INTO Arch\_Products\_Del (id\_product, ProductName, Quantity, Unit, DeletedDateTime)

SELECT id\_product, ProductName, Quantity, Unit, GETDATE()

FROM deleted;

END;

--Триггер на добавление выполненных заказов в архив

GO

CREATE TRIGGER ArchiveDeletedYearlyOrders

ON Yearly\_Orders

AFTER DELETE

AS

BEGIN

INSERT INTO Arch\_Yearly\_Orders\_Del (id\_yearly\_orders, id\_shop, id\_product, Quantity, DeletedDateTime)

SELECT id\_yearly\_orders, id\_shop, id\_product, Quantity, GETDATE()

FROM deleted;

END;

# Приложение Д

Листинг создания хранимой процедуры для импорта данных из CSV файлов в базу данных:

--Хранимая процедура для добавления данных из csv в таблицы

GO

CREATE PROCEDURE LoadDataFromCSV

AS

BEGIN

--Products

SET IDENTITY\_INSERT dbo.Products ON;

IF (OBJECT\_ID('tempdb.dbo.#csv\_tab\_products') IS NOT NULL)

DROP TABLE #csv\_tab\_products;

CREATE TABLE #csv\_tab\_products (

pole1 NVARCHAR(max),

pole2 NVARCHAR(max),

pole3 NVARCHAR(max),

pole4 NVARCHAR(max)

);

BULK INSERT #csv\_tab\_products

FROM 'C:\Program Files (x86)\Microsoft SQL Server\csv\Products.csv'

WITH (

FIELDTERMINATOR = ';',

ROWTERMINATOR = '\n'

);

INSERT INTO dbo.Products (id\_product, ProductName, Quantity, Unit)

SELECT

CAST(pole1 AS INT),

CAST(pole2 AS NVARCHAR(25)),

CAST(pole3 AS DECIMAL(18 ,2)),

CAST(pole4 AS NVARCHAR(20))

FROM #csv\_tab\_products;

SET IDENTITY\_INSERT dbo.Products OFF;

--Shops

SET IDENTITY\_INSERT dbo.Shops ON;

IF (OBJECT\_ID('tempdb.dbo.#csv\_tab\_shops') IS NOT NULL)

DROP TABLE #csv\_tab\_shops;

CREATE TABLE #csv\_tab\_shops (

pole1 NVARCHAR(max),

pole2 NVARCHAR(max)

);

BULK INSERT #csv\_tab\_shops

FROM 'C:\Program Files (x86)\Microsoft SQL Server\csv\Shops.csv'

WITH (

FIELDTERMINATOR = ';',

ROWTERMINATOR = '\n'

);

INSERT INTO dbo.Shops (id\_shop, ShopName)

SELECT

CAST(pole1 AS INT),

CAST(pole2 AS NVARCHAR)

FROM #csv\_tab\_shops;

SET IDENTITY\_INSERT dbo.Shops OFF;

--Products\_In\_Stores

SET IDENTITY\_INSERT dbo.Products\_In\_Shops ON;

IF (OBJECT\_ID('tempdb.dbo.#csv\_tab\_products\_in\_shops') IS NOT NULL)

DROP TABLE #csv\_tab\_products\_in\_shops;

CREATE TABLE #csv\_tab\_products\_in\_shops (

pole1 NVARCHAR(max),

pole2 NVARCHAR(max),

pole3 NVARCHAR(max),

pole4 NVARCHAR(max)

);

BULK INSERT #csv\_tab\_products\_in\_shops

FROM 'C:\Program Files (x86)\Microsoft SQL Server\csv\Products\_In\_Shops.csv'

WITH (

FIELDTERMINATOR = ';',

ROWTERMINATOR = '\n'

);

INSERT INTO dbo.Products\_In\_Shops (id\_products\_In\_Shops, id\_product, id\_shop, Quantity)

SELECT

CAST(pole1 AS INT),

CAST(pole2 AS INT),

CAST(pole3 AS INT),

CAST(pole4 AS DECIMAL)

FROM #csv\_tab\_products\_in\_shops;

SET IDENTITY\_INSERT dbo.Products\_in\_shops OFF;

--Yearly\_Orders

SET IDENTITY\_INSERT dbo.Yearly\_Orders ON;

IF (OBJECT\_ID('tempdb.dbo.#csv\_tab\_yearly\_orders') IS NOT NULL)

DROP TABLE #csv\_tab\_yearly\_orders;

CREATE TABLE #csv\_tab\_yearly\_orders (

pole1 NVARCHAR(max),

pole2 NVARCHAR(max),

pole3 NVARCHAR(max),

pole4 NVARCHAR(max)

);

BULK INSERT #csv\_tab\_yearly\_orders

FROM 'C:\Program Files (x86)\Microsoft SQL Server\csv\Yearly\_Orders.csv'

WITH (

FIELDTERMINATOR = ';',

ROWTERMINATOR = '\n'

);

INSERT INTO dbo.Yearly\_Orders(id\_yearly\_orders, id\_shop, id\_product, Quantity)

SELECT

CAST(pole1 AS INT),

CAST(pole2 AS INT),

CAST(pole3 AS INT),

CAST(pole4 AS DECIMAL)

FROM #csv\_tab\_yearly\_orders;

SET IDENTITY\_INSERT dbo.Yearly\_Orders OFF;

END;