

Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра информационных технологий автоматизированных систем

Расчетно-пояснительная записка к курсовому проекту
по курсу «Базы и банки данных»
на тему:
«База данных по пациентам поликлиники»

Руководитель проекта:

Выполнил:

Студент группы 890650
Лукашевич М. А.

Минск 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
1 Постановочная часть	6
1.1. Описание и анализ предметной области	6
1.2. Постановка задачи проектирования.....	7
1.3. Обоснование выбора модели данных	7
2 Теоретическая часть	10
2.1. Проектирование структуры базы данных.....	10
2.2. Формирование информационной структуры	14
2.3. Оптимизация (нормализация) информационной структуры.....	16
3 Расчётная часть	20
3.1. Выбор инструментальной платформы и языка программирования для реализации базы данных.	20
ЗАКЛЮЧЕНИЕ	24
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	26
Приложение А	27
Приложение Б.....	27

ВВЕДЕНИЕ

Система управления базами данных (СУБД) - комплекс программных и лингвистических средств общего или специального назначения, реализующий поддержку создания баз данных, централизованного управления и организации доступа к ним различных пользователей в условиях принятой технологии обработки данных.

На сегодняшний день существует множество всевозможных СУБД, которые отличаются между собой различными алгоритмами обработки данных.

Банк данных - автоматизированная информационная система централизованного хранения и коллективного использования данных. В состав банка данных входят одна или несколько баз данных, справочник баз данных, СУБД, а также библиотеки запросов и прикладных программ.

База данных - совокупность связанных данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования, независимая от прикладных программ. База данных является информационной моделью предметной области. Обращение к базам данных осуществляется с помощью системы управления базами данных (СУБД).

В современном информационном обществе повышается значение информации как товара. Это является следствием общего роста информационных потребностей и выражением развития отрасли информационных услуг.

В современных информационных технологиях неотъемлемую роль играют базы данных (БД). Данные в БД логически структурированы с целью обеспечения возможности их эффективного поиска и обработки в вычислительной системе. Также, базы данных позволяют организовать целостность данных. Таким образом, все данные из предметной области можно хранить в строго структурированном виде.

В контексте данной курсовой работы рассматривается база данных материально-ответственного лица.

Сегодня многие предприятия обладают огромной материальной базой, которая требует непрерывного обслуживания и мониторинга. Это задача материально-ответственного лица.

Специалист данного профиля обеспечивает бесперебойное снабжение предприятий и организаций необходимым сырьем, оборудованием, запасными частями, комплектующими. Он заказывает, принимает, организует хранение, выдачу рабочим группам материалов, комплектующих. Ведет их учет, определяет возможность замены одних материалов другими. Рассчитывает потребность в материалах, оборудовании и приборах по объему производства и сроков сдачи производственных работ, заказывает и контролирует получение, отслеживает расход материалов и корректирует планы заказов. Ведет документацию по заказам, приему, расходованию материалов, управляет складским технологическим оборудованием, следит за техникой безопасности.

SQL (Structured Query Language — язык структурированных запросов) — универсальный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных базах данных, основывается на реляционной алгебре.

Целью данной работы является изучение теории и практическое использование ее в создании базы данных, работе с СУБД, применении языка запросов SQL, написании пользовательского приложения для манипуляции с базой данных, что, несомненно, является актуальной задачей на сегодняшний день.

1 ПОСТАНОВОЧНАЯ ЧАСТЬ

1.1. Описание и анализ предметной области

Информационная система представляет собой любую систему обработки информации.

Информационная система в более узком смысле – совокупность аппаратно-программных средств, задействованных для решения некоторой прикладной задачи.

Информационные системы предназначены для хранения, обработки, поиска, распространения, передачи и предоставления информации.

В контексте данной работы будем рассматривать информационную систему регистратора поликлиники.

Такие информационные системы позволяют решать следующие основные задачи:

- Вести учёт пациентов;
- Вести учёт карт пациентов;
- Вести учёт записей;

Рассмотрим более подробно некоторые таблицы базы данных, созданной для решения данных задач.

Таблица «DEPARTMENT» содержит информацию об отделах предприятия, в которых могут работать доктора.

Таблица «DOCTOR» содержит информацию докторам.

Таблица «PATIENT» содержит информацию о пациентах.

Таблица «APPOINTMENT» содержит информацию о записях к докторам.

Таблица «PATIENT_CARD» содержит информацию о карточках пациентов

Таблица «DIAGNOSIS» содержит информацию о возможных диагнозах.

Таким образом, проектируемая база данных будет всю необходимую информацию содержать.

1.2. Постановка задачи проектирования

Целью курсового проектирования является построение базы данных по пациентам поликлиники

Главная задача моделируемой системы – сохранение в базе данных всех необходимых сведений о работниках, подразделениях, должностях, инвентаре и ценностях в предприятии; осуществление поиска по заданному критерию; предоставление данных в удобном для пользователя виде.

Технические средства. Применяемая СУБД: MySQL 5.1.14. Платформа и технологии: Kotlin, Ktor. Сервер приложений: Netty. Требования к целостности данных: данные в базе данных в любой момент времени должны быть правильными и непротиворечивыми. Требования к безопасности: доступ к системе должен быть осуществлён после аутентификации пользователя.

БД содержит информацию:

- сведения о поликлинике;
- сведения о врачах;
- сведения о пациентах;
- сведения об отделениях поликлиники;
- сведения о карточках пациентов;

Кроме того, программа должна иметь дружелюбный и приятный интерфейс.

1.3. Обоснование выбора модели данных

Ядром любой базы данных является модель данных. Модель данных представляет собой множество структур данных, ограничений целостности и операций манипулирования данными. С помощью модели данных могут быть представлены объекты предметной области и взаимосвязи между ними.

Модель данных — совокупность структур данных и операций их обработки.

СУБД основывается на использовании иерархической, сетевой или реляционной модели, на комбинации этих моделей или на некотором их подмножестве.

Принципы реляционной модели были сформулированы в 1969-1970 годах Э.Ф. Коддом.

Эти модели характеризуются простотой структуры данных, удобным для пользователя табличным представлением и возможностью использования формального аппарата алгебры отношений и реляционного исчисления для обработки данных.

Реляционная модель ориентирована на организацию данных в виде двумерных таблиц. Каждая реляционная таблица представляет собой двумерный массив и обладает следующими свойствами:

- каждый элемент таблицы есть один элемент данных;
- все столбцы в таблице однородные, т.е. все элементы в столбце имеют одинаковый тип (числовой, символьный и т.д.) и длину;
- каждый столбец имеет уникальное имя;
- одинаковые строки в таблице отсутствуют;
- порядок следования строк и столбцов может быть произвольным;

Отношения представлены в виде таблиц, строки которых соответствуют кортежам или записям, а столбцы — атрибутам отношений, доменам, полям.

Таким образом, реляционная модель данных предоставляет все удобства для работы с данными.

На рисунке 1.1 можно ознакомиться с концептуальной (сущностной) моделью нашей базы данных.

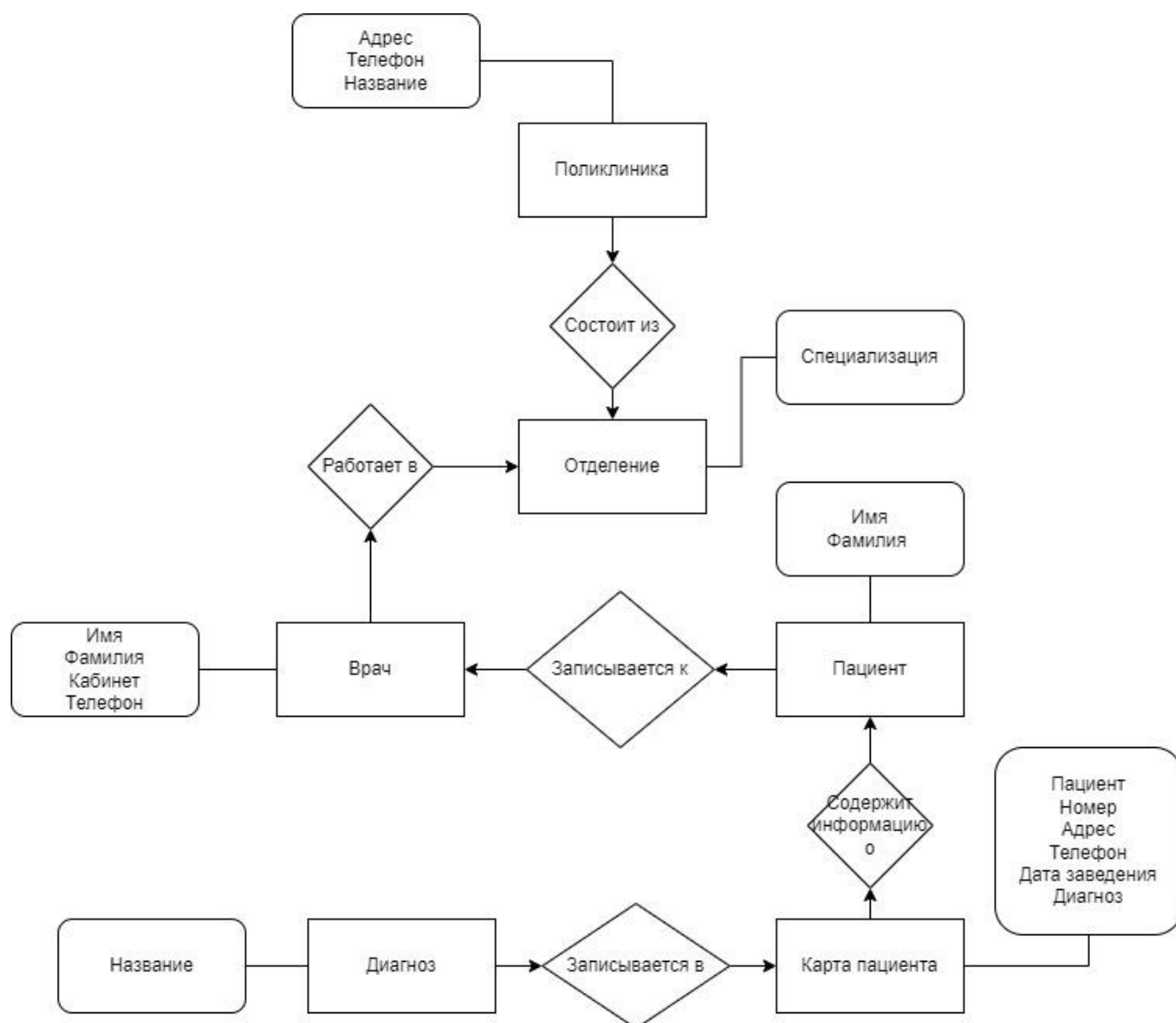


Рисунок 1.1 – Концептуальная модель базы данных.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1. Проектирование структуры базы данных

База данных представляет собой 6 таблиц.

Рассмотрим сущности, из которых состоит БД с указанием всех индексов.

Сущность «DEPARTMENT» содержит в себе информацию о отделениях.

Описание приведено в таблице 2.1. Вид таблицы представлен на рисунке 2.1.

Таблица 1.1 – Поля сущности «DEPARTMENT»

Имя поля	Тип данных	Обязательное поле
ID	INT	Да
SPECIALIZATION	VARCHAR (250)	Да
MANAGER	INT	Да
TELEPHONE	VARCHAR (20)	Да

На рисунке 2.1 представлена таблица «DEPARTMENT».

ID	SPECIALIZATION	MANAGER	TELEPHONE
1	Отделение общей врачебной практики	2	375(17)1233211
2	Женская консультация	1	375(17)1233212
3	Доврачебный кабинет и отделение профилактики №1	3	375(17)1233213
4	Доврачебный кабинет и отделение профилактики №2	4	375(17)1233214
5	Доврачебный кабинет и отделение профилактики №3	5	375(17)1233215
6	Травма-хирургическое отделение	6	375(17)1233216

Рисунок 2.1 – Содержимое таблицы «DEPARTMENT».

Сущность «DOCTOR» содержит в себе информацию о врачах. Описание приведено в таблице 2.2. Вид таблицы представлен на рисунке 2.2.

Таблица 2.2 – Поля сущности «DOCTOR»

Имя поля	Тип данных	Обязательное поле
ID	INT	Да
NAME	VARCHAR (80)	Да
SURNAME	VARCHAR (80)	Да
CABINET	INT	Да
TELEPHONE	VARCHAR (20)	Да

На рисунке 2.2 представлена таблица «DOCTOR».

DOCTOR_ID	NAME	SURNAME	CABINET	TELEPHONE
1	Василиса	Кулькина	201	375(17)1233211
2	Владимир	Моль	301	375(17)1233212
3	Анна	Шоль	211	375(17)1233213
4	Людмила	Пуль	101	375(17)1233214
5	Анастасия	Василь	402	375(17)1233215
6	Максим	Лукашевич	245	375(17)1233216

Рисунок 2.2 – Содержимое таблицы «DOCTOR».

Сущность «PACIENT» содержит в себе информацию о пациентах. Описание приведено в таблице 2.3. Вид таблицы представлен на рисунке 2.3.

Таблица 2.3 – Поля сущности «PACIENT»

Имя поля	Тип данных	Обязательное поле
ID	INT	Да
FIRST_NAME	VARCHAR (80)	Да
LAST_NAME	VARCHAR (80)	Да

На рисунке 2.3 представлена таблица «PACIENT».

PACIENT_ID	NAME	SURNAME
1	Владислав	Свой
2	Лютый	Жуль
3	Жанна	Пьер
4	Игорь	Владиновский
5	Вероника	Централь
6	Александр	Северный

Рисунок 2.3 – Содержимое таблицы «PACIENT».

Сущность «PATIENT_CARD» содержит в себе информацию о карточках пациентов. Описание приведено в таблице 2.4. Вид таблицы представлен на рисунке 2.4.

Таблица 2.4 – Поля сущности «PATIENT_CARD»

Имя поля	Тип данных	Обязательное поле
ID	INT	Да
PACIENT_ID	INT	Да
NUMBER	VARCHAR (80)	Да
ADDRESS	VARCHAR (80)	Да
TELEPHONE	VARCHAR (20)	Да
DATE_OF_ESTABLISHMENT	VARCHAR (80)	Да
DIAGNOSIS_ID	INT	Да

На рисунке 2.4 представлена таблица «PATIENT_CARD».

PACIENT_ID	NUMBER	ADDRESS	TELEPHONE	DATE_OF_ESTABLISHMENT	DIAGNOSIS_ID
Василиса	11	ул.Владыко д59 кв256	375(17)1233211	01.02.2021	1
Владимир	222	ул.Владыко д59 кв256	375(17)1233212	01.02.2021	3
Анна	321	ул.Владыко д11 кв256	375(17)1233213	01.02.2021	1
Людмила	1111	ул.Володко д9 кв256	375(17)1233214	01.02.2021	NONE
Анастасия	541	ул.Ясная д191 кв11	375(17)1233215	01.02.2021	NONE
Максим	99	ул.Прекраса д5 кв56	375(17)1233216	01.02.2021	4

Рисунок 2.4 – Содержимое таблицы «PATIENT_CARD».

Сущность «DIAGNOSIS» содержит в себе информацию о диагнозах. Описание приведено в таблице 2.5. Вид таблицы представлен на рисунке 2.5.

Таблица 2.5 – Поля сущности «DIAGNOSIS»

Имя поля	Тип данных	Обязательное поле
ID	INT	Да
NAME	VARCHAR (80)	Да

На рисунке 2.5 представлена таблица «DIAGNOSIS».

DIAGNOSIS_ID	NAME
1	Болезнь Крона
2	Алкоголизм
3	Аденоиды
4	Болезнь Альцгеймера
5	Геморрой
6	Гепатит

Рисунок 2.5 – Содержимое таблицы «DIAGNOSIS».

Сущность «APPOINTMENT» содержит в себе информацию о записях. Описание приведено в таблице 2.5. Вид таблицы представлен на рисунке 2.5.

Таблица 2.6 – Поля сущности «APPOINTMENT»

Имя поля	Тип данных	Обязательное поле
ID	INT	Да
DOCTOR_ID	INT	Да
PATIENT_ID	INT	Да
DATE	VARCHAR (80)	Да
TIME	VARCHAR (80)	Да

На рисунке 2.6 представлена таблица «APPOINTMENT».

ID	DOCTOR_ID	PATIENT_ID	DATE	TIME
1	2	5	28.12.2022	10:30:00
2	3	6	28.12.2022	11:30:00
3	4	7		12:30:00

Рисунок 2.6 – Содержимое таблицы «APPOINTMENT».

Все сущности связаны между собой по ключевым полям. Таблица «PATIENT_CARD» связана с таблицей «PATIENT» по ключевому полю «PATIENT_ID», связь между таблицами – один к одному. Связь между таблицами «DOCTOR» и «DEPARTMENT» – много к одному – осуществляется по полю «DEPARTMENT_ID».

Сущность «DIAGNOSIS» связана с сущностью «PATIENT_CARD» по ключевому полю «DIAGNOSIS_ID», связь между таблицами – много к одному.

Таблица «APPOINTMENT» связана с таблицей «DOCTOR» по ключевому полю «DOCTOR_ID», связь между таблицами – один ко многим и с таблицей «PATIENT» по ключевому полю «PATIENT_ID» – один ко многим.

Листинг программы приведен в приложении А.

2.2. Формирование информационной структуры

На практике при построении логической модели реляционной базы данных особое значение для решения задачи формирования отношений базы данных имеет понятие функциональной зависимости (ФЗ). Установление ФЗ и получение наилучшего с точки зрения минимальности представления множества ФЗ позволят построить наиболее оптимальный вариант базы данных, обеспечивающий надежность хранения и обработки данных на основе методов эквивалентных преобразований схем отношений реляционной базы данных.

Процесс решения такой задачи называется нормализацией отношений информационной модели предметной области и заключается в превращении ее объектов в логические таблицы базы данных.

Рассмотрим сущность «DOCTOR». Известно, что разные доктора могут работать в одном и том же отделении. Чтобы при вводе данных о новом работнике всякий раз не дублировать данные, которые характеризуют эту ситуацию, создается вспомогательная сущность – «DEPARTMENT».

Определим первичные ключи таблиц, определим атрибуты, по которым будем строить индексы, а также продумаем, какие хранимые процедуры следует написать. Прделаем эти действия для оставшихся таблиц.

2.3. Оптимизация (нормализация) информационной структуры

Нормализация отношений информационной модели предметной области является механизмом создания логической модели реляционной базы данных.

Заметим, что с математической точки зрения задача построения, как информационной модели предметной области, так и логической модели реляционной базы данных является результатом решения следующих комбинаторных задач:

- группировка атрибутов в отношении предметной области;
- распределение атрибутов по отношениям базы данных;

Такие задачи имеют решения, допускающие большое число вариантов, и приводят к проблеме выбора рационального варианта из множества альтернативных вариантов схем отношений. Выбор наиболее рационального варианта обусловлен соблюдением различного рода соглашений и требований.

Перечень наиболее важных требований приведен ниже.

- Первичные ключи отношений должны быть минимальными (требование минимальности первичных ключей);
- Число отношений базы данных должно по возможности давать наименьшую избыточность данных (требование надежности данных);
- Число отношений базы данных не должно приводить к потере производительности системы (требование производительности системы);
- Данные не должны быть противоречивыми, т.е. при выполнении операций включения, удаления и обновления, данных их потенциальная противоречивость должна быть сведена к минимуму (требования непротиворечивости данных);
- Схема отношений базы данных должна быть устойчивой, способной адаптироваться к изменениям при ее расширении дополнительными атрибутами (требование гибкости структуры базы данных);
- Разброс времени реакции на различные запросы к базе данных не должен быть большим (требование производительности системы);

– Данные должны правильно отражать состояние предметной области базы данных в каждый конкретный момент времени (требование актуальности данных);

Таблица находится в первой нормальной форме (1НФ), если все содержащиеся в ней атрибуты имеют атомарные (одиночные) значения. Для приведения к 1НФ строки и/или столбцы, содержащие неатомарные атрибуты, разбиваются на несколько строк и/или столбцов.

В нашей базе данных туристического портала атрибуты во всех таблицах имеют атомарные значения, значит, все таблицы находятся в первой нормальной форме.

После приведения таблиц к первой нормальной форме могут оставаться аномалии добавления, удаления, обновления, а также возможна избыточность данных.

Приведение таблиц ко второй нормальной форме поможет решить некоторые из этих проблем.

Таблица находится во второй нормальной форме (2НФ), если она находится в 1НФ, и все ее не ключевые атрибуты находятся в полной функциональной зависимости от ключа.

Многие таблицы нашей базы данных имеют простой ключ. Таким образом, можно сказать, что эти таблицы автоматически можно считать приведённым ко второй нормальной форме, так как все ее не ключевые атрибуты находятся в полной функциональной зависимости от ключа.

Таблица приводится к 2НФ в следующем порядке:

– выделяются все атрибуты, находящиеся в полной функциональной зависимости от ключа. Эти атрибуты, вместе с ключом, выделяются в отдельную таблицу;

– выделяются атрибуты, которые находятся в функциональной зависимости от одной из частей ключа. Эти атрибуты вместе с той частью ключа, от которой они зависят, выделяются в отдельную таблицу. Такое действие

выполняется для всех неполных функциональных зависимостей, имеющих в таблице.

Ко второй нормальной форме нам требуется привести таблицу с агентствами. Для этого выделим все атрибуты, находящиеся в зависимости от ключей "Офисы турфирм" и "Отзывы турфирм". Выделяем эти атрибуты в отдельную таблицу.

Базы данных, приведенные к 2НФ, также могут иметь существенные недостатки, такие как избыточность данных, аномалия добавления, аномалия удаления и аномалия обновления. Эти недостатки будут устранены при приведении БД к третьей нормальной форме.

Основная причина недостатков БД во второй нормальной форме – наличие транзитивных зависимостей. Пусть атрибут В находится в функциональной зависимости от атрибута или группы атрибутов А ($A \rightarrow B$), а атрибут С – в функциональной зависимости от атрибута В ($B \rightarrow C$), причем А - ключ, В и С – не ключевые атрибуты. Такая зависимость ($A \rightarrow B \rightarrow C$) называется транзитивной. Таблица находится в третьей нормальной форме (3НФ), если она находится в 2НФ (а значит, и в 1НФ) и не содержит транзитивных зависимостей.

Чтобы привести к 3НФ таблицу, содержащую транзитивную зависимость $A \rightarrow B \rightarrow C$, требуется выделить атрибуты В и С в отдельную таблицу, где В становится ключевым атрибутом. Другими словами, для приведения к 3НФ таблица разбивается на две: в одну из них помещаются все атрибуты, кроме атрибута С, в другую – атрибуты В и С.

Таким образом, вся база данных киностудии приведена к 3НФ. Сопоставив все полученные таблицы и, таблицы с исходными данными легко убедиться, что в процессе нормализации никакие данные не потеряны.

На рисунке 2.8 приведена модель базы данных после приведения таблиц к третьей нормальной форме.

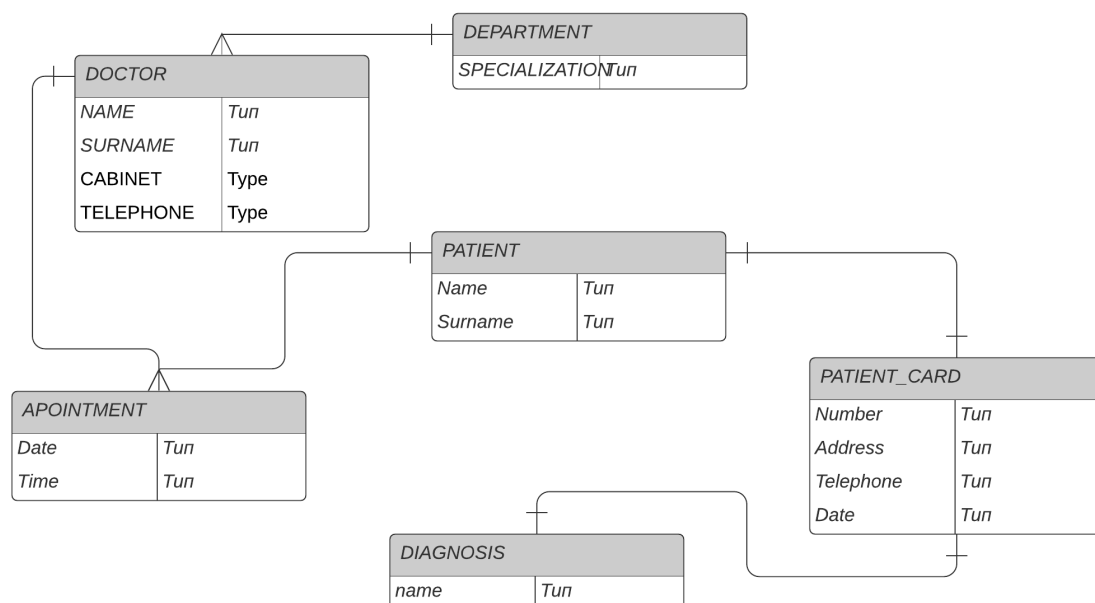


Рисунок 2.8 – ER диаграмма

В теории баз данных говорится о том, что схема базы данных должна быть полностью нормализована. При работе с полностью нормализованными базами данных необходимо применять весьма сложные SQL-запросы, что приводит к обратному эффекту – замедлению работы базы данных. Поэтому иногда для упрощения запросов даже прибегают к обратной процедуре – денормализации.

3 РАСЧЁТНАЯ ЧАСТЬ

3.1. Выбор инструментальной платформы и языка программирования для реализации базы данных.

При выборе платформы для реализации базы данных следует обратить внимание на следующие детали. База данных пациентов поликлиники будет реализована как Kotlin приложение, поэтому предполагается, что она рассчитана на одновременное использование несколькими пользователями. В этом плане PostgreSQL в качестве платформы для реализации базы данных нас полностью устраивает.

С помощью PostgreSQL можно быстро создавать деловые приложения для различных сфер деятельности человека. В то же время СУБД PostgreSQL имеет архитектурные ограничения (например, максимальный размер базы данных не более двух гигабайт), которые не позволяют использовать этот инструмент для управления большими промышленными распределенными базами данных. Для таких целей применяются СУБД промышленного уровня, такие как Oracle, IBM DB2, Microsoft SQL Server, Sybase и ряд других. Но так как наша база данных не будет настолько велика, то и по данному критерию СУБД MySQL нам подходит.

PostgreSQL – это функционально полная реляционная СУБД. В ней предусмотрены все необходимые средства для определения и обработки данных, а также для управления ими при работе с большими объектами информации. Области применения PostgreSQL обозначены достаточно ясно. PostgreSQL является решением для малых и средних приложений. Входит в состав стеков технологий WAMP, LAMP и в портативные сборки серверов Денвер, ХАМРР. Обычно PostgreSQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать PostgreSQL в автономные программы.

Гибкость СУБД PostgreSQL обеспечивается поддержкой большого количества типов таблиц. Поддерживающие транзакции на уровне отдельных записей.

Postgresql имеет API для языков Delphi, C, C++, Эйфель, Java, Лисп, Perl, PHP, Python, Ruby, Smalltalk и Tcl, библиотеки для языков платформы .NET, а также обеспечивает поддержку для ODBC посредством ODBC-драйвера.

База данных будет иметь интерфейс, реализованный как Kotlin приложение, написанное с помощью фреймворка Ktor. Теперь немного теории и доводов в пользу выбора технологий.

Kotlin — объектно-ориентированный язык программирования, разработанный компанией JetBrains. Приложения Kotlin обычно компилируются в специальный java байт-код или в другой код языков, поэтому они могут работать на любой виртуальной Java-машине (JVM) независимо от компьютерной архитектуры, а также исполняться как другой высокоуровневый код с помощью LVVM компилятора.

Достоинство подобного способа выполнения программ — в полной независимости байт-кода от операционной системы и оборудования, что позволяет выполнять Kotlin-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности благодаря тому, что исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером) вызывают немедленное прерывание.

Часто к недостаткам концепции виртуальной машины относят то, что исполнение байт-кода виртуальной машиной может снижать производительность программ и алгоритмов, реализованных на языке Kotlin. Данное утверждение было справедливо для первых версий виртуальной машины Java, однако в последнее время оно практически потеряло актуальность. Этому способствовал ряд усовершенствований:

- применение технологии трансляции байт-кода в машинный код непосредственно во время работы программы (JIT-технология) с возможностью сохранения версий класса в машинном коде;

- широкое использование платформенно-ориентированного кода (native-код) в стандартных библиотеках;
- аппаратные средства, обеспечивающие ускоренную обработку байт-кода (например, технология Jazelle, поддерживаемая некоторыми процессорами фирмы ARM);

Ktor это платформа для простого создания подключенных приложений — веб-приложений, HTTP-сервисов, мобильных и браузерных приложений. Современные подключенные приложения должны быть асинхронными, чтобы предоставить пользователям наилучшие возможности, а сопрограммы Kotlin предоставляют потрясающие возможности для этого простым и понятным способом.

Хотя это еще не совсем так, цель Ktor состоит в том, чтобы предоставить сквозную много платформенную среду приложений для подключенных приложений. В настоящее время поддерживаются клиентские и серверные сценарии JVM, а также клиенты JavaScript, iOS и Android, и мы работаем над внедрением серверных средств в собственные среды, а клиентских средств — в другие собственные цели.

Для реализации сервера выбран движок Netty, а для реализации postgresSQL HikariCP.

HikariCP - это быстрый и легкий пул соединений. По сравнению с другими пулами соединений HikariCP имеет некоторое улучшение производительности. Ниже приводится сравнение его производительности с некоторыми другими пулами соединений:

Что касается причин, по которым пул соединений HikariCP работает быстрее, чем другие пулы соединений, существует несколько аспектов:

По сравнению с обычными пулами соединений ArrayList используется для хранения объектов Statement в ConnectionProxy. HikariCP использует FatList для хранения объектов. Разница в том, что каждый раз, когда ArrayList выполняет метод get (Index), область действия List должна быть Проверка не требуется, и

FastList не требуется. Пока может быть обеспечена достоверность диапазона, накладные расходы на проверку диапазона могут быть опущены.

Кроме того, в коде Java многие операции над подключением закрываются непосредственно после использования, ранее они проходили от начала до конца, чтобы закрыть соответствующее соединение, в то время как HikariCP сканирует коллекцию подключений с конца. С точки зрения производительности, начиная с хвоста лучше.

Внутренне для хранения используется коллекция без блокировки, и операция переключения в ней оптимизирована.

ЗАКЛЮЧЕНИЕ

Разрабатываемая база данных в контексте данной работы в перспективе развития может использоваться как open-source пример сервера на ktor.

В данной работе разработана база данных, реализующая базу данных пациентов поликлиники.

В процессе выполнения курсовой работы были закреплены знания, полученные при изучении дисциплины «Банки и базы данных». Были изучены такие пункты:

- анализ предметной области;
- построение концептуальной модели базы данных;
- организация базы данных;
- разработка прикладной программы;
- наполнение и сопровождение базы данных;

В процессе организации БД проведен до необходимого уровня абстракций анализ предметной области, построена реляционная модель БД, произведена нормализация реляционной БД. Оформляя пояснительную записку, были ознакомлены с государственными стандартами СТП 01-2010. Реализация проекта была выполнена на современных программных платформах. В качестве технологии доступа к данным была использована объектно-реляционная модель, которая позволяет просто и лаконично осуществлять запросы к базе данных. Была освоена и закреплена работа с такими прикладными программами:

- KotlinJs;
- KotlinJvm;
- Kotlin Html dsl;
- Kotlin gradle dsl
- Ktor;
- Postgres;
- Google Chrome Developer Tools;

- Konsole;
- Gimp;

Предполагается дальнейшее развитие и улучшение созданной БД.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Spring Roo - Reference Documentation / Бен Алекс, Стефан Шмидт [и др.]; Springsource community. – Boston, 2011. -193 с.
2. Spring Framework - Reference Documentation / Род Джонсон, Юрген Хоуэллер [и др.]; Springsource community. – Boston, 2011. -791 с.
3. Segerstad D. Be calm while making your database / Daniel Segerstad; под ред. Johannes Hedberg. –Oslo, 2003. -1488 с.
4. Кенобьев О.В. Изучаем Hibernate ORM за 24 часа / О.В. Кенобьев; под ред. А. Небоходова. –Спб, 2007. -501 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Примеры листингов программы

Таблица Записей

```
package me.x99.database

import org.jetbrains.exposed.dao.id.IntIdTable
import org.jetbrains.exposed.sql.Column

object AppointmentTable : IntIdTable() {
    val doctorId: Column<Int> =
integer("doctorId").references(DoctorTable.id)
    val patientId: Column<Int> =
integer("patientId").references(PatientTable.id)
    val date = varchar("date", 100)
    val time = varchar("time", 100)
}
```

Роjo класс для записей

```
package me.x99.model

data class Appointment(
    val id: Int,
    val doctor: Doctor?,
    val patient: Patient?,
    val date: String,
    val time: String
){
    constructor() : this(-999, null, null, "", "")
}
```

Класс репозитория записей

```
package me.x99.repo

import me.x99.database.AppointmentTable
import me.x99.database.DepartmentTable
import me.x99.database.DoctorTable
import me.x99.database.PatientTable
import me.x99.model.Appointment
import me.x99.model.Department
import me.x99.model.Doctor
import me.x99.model.Patient
import org.jetbrains.exposed.sql.*
import org.jetbrains.exposed.sql.SqlExpressionBuilder.eq
import org.jetbrains.exposed.sql.transactions.transaction

class AppointmentRepo {
```

```

suspend fun create(appointment: Appointment) {
    transaction {
        AppointmentTable.insert {
            it[doctorId] = appointment.doctor?.id ?: -1
            it[patientId] = appointment.patient?.id ?: -1
            it[date] = appointment.date
            it[time] = appointment.time
        }
    }
}

suspend fun getAll(): List<Appointment> {
    return transaction {
        AppointmentTable.selectAll().toDoctors()
    }
}

suspend fun get(id: Int): Appointment? {
    return transaction {
        AppointmentTable.select { AppointmentTable.id eq id }
            .toDoctors().firstOrNull()
    }
}

suspend fun delete(id: Int) {
    return transaction {
        AppointmentTable.deleteWhere {
            AppointmentTable.id eq id
        }
    }
}

private fun Query.toDoctors() = this.map {
    val doctor = DoctorTable.select(DoctorTable.id eq
it[AppointmentTable.doctorId]).map { doctorRow ->
        val department =
            DepartmentTable.select(DepartmentTable.id eq
doctorRow[DoctorTable.departmentId])
                .firstOrNull()?.toDepartment()
            doctorRow.toDoctors(department)
        }.firstOrNull()
    val patient: Patient =
        PatientTable.select(PatientTable.id eq
it[AppointmentTable.patientId]).first().toPatient()
    it.toAppointment(doctor, patient)
}

private fun ResultRow.toAppointment(doctor: Doctor?, patient:
Patient): Appointment {
    return Appointment(
        id = this[AppointmentTable.id].value,
        doctor = doctor,

```

```

        patient = patient,
        date = this[AppointmentTable.date],
        time = this[AppointmentTable.time]
    )
}

private fun ResultRow.toDoctors(department: Department?):
Doctor {
    return Doctor(
        id = this[DoctorTable.id].value,
        name = this[DoctorTable.name],
        surname = this[DoctorTable.surname],
        cabinet = this[DoctorTable.cabinet],
        telephone = this[DoctorTable.telephone],
        department = department
    )
}

private fun ResultRow.toPatient(): Patient {
    return Patient(
        id = this[PatientTable.id].value,
        name = this[PatientTable.name],
        surname = this[PatientTable.surname]
    )
}

private fun ResultRow.toDepartment(): Department {
    return Department(
        id = this[DepartmentTable.id].value,
        specialization = this[DepartmentTable.specialization],
        telephone = this[DepartmentTable.telephone]
    )
}
}

```

Класс обращение к базе данных через запрос

```

package me.x99.routing.data

import io.ktor.application.call
import io.ktor.http.*
import io.ktor.request.receive
import io.ktor.response.respond
import io.ktor.routing.*
import me.x99.model.Appointment

fun Route.appointmentRoute(appointmentRepo:
me.x99.repo.AppointmentRepo) {

    route("/appointment") {
        get("/{appointmentId}") {
            val appointmentId = call.parameters["appointmentId"]

```

```

        ?: throw IllegalArgumentException("Parameter
appointment Id not found")
        appointmentRepo.get(appointmentId.toInt())?.let {
appointment -> call.respond(appointment) }
    }

    get("/") {
        call.respond(appointmentRepo.getAll())
    }

    post("/create") {
        val receivedAppointment =
call.receive(Appointment::class)

call.respond(appointmentRepo.create(receivedAppointment))
    }

    delete("/{appointmentId}") {
        val id = call.parameters["id"]?.toInt() ?:
error("Invalid delete request")
        appointmentRepo.delete(id)
        call.respond(HttpStatusCode.OK)
    }
}
}

```

Класс фабрика для базы данных

```

package me.x99.database

import com.zaxxer.hikari.HikariConfig
import com.zaxxer.hikari.HikariDataSource
import me.x99.full_data.setData
import org.jetbrains.exposed.sql.Database
import org.jetbrains.exposed.sql.SchemaUtils
import org.jetbrains.exposed.sql.transactions.transaction

object DatabaseFactory {

    fun init() {
        Database.connect(hikari())
        transaction {
            SchemaUtils.create(AppointmentTable)
            SchemaUtils.create(DiagnosisTable)
            SchemaUtils.create(PatientTable)
            SchemaUtils.create(DepartmentTable)
            SchemaUtils.create(DoctorTable)
            SchemaUtils.create(PatientCardTable)

            DepartmentTable.setData()
            DoctorTable.setData()
            PatientTable.setData()
            DiagnosisTable.setData()
        }
    }
}

```

```

        PatientCardTable.setData()
        AppointmentTable.setData()
    }
}

/**
 * Look at hikari.properties and change accordingly
 * */
private fun hikari(): HikariDataSource {
    val config = HikariConfig()
    config.driverClassName = "org.h2.Driver"
    config.jdbcUrl = "jdbc:h2:mem:test"
    config.maximumPoolSize = 3
    config.isAutoCommit = false
    config.transactionIsolation =
"TRANSACTION_REPEATABLE_READ"
    config.validate()
    return HikariDataSource(config)
}
}

```

ПРИЛОЖЕНИЕ Б

(информационное)

Примеры работы программы

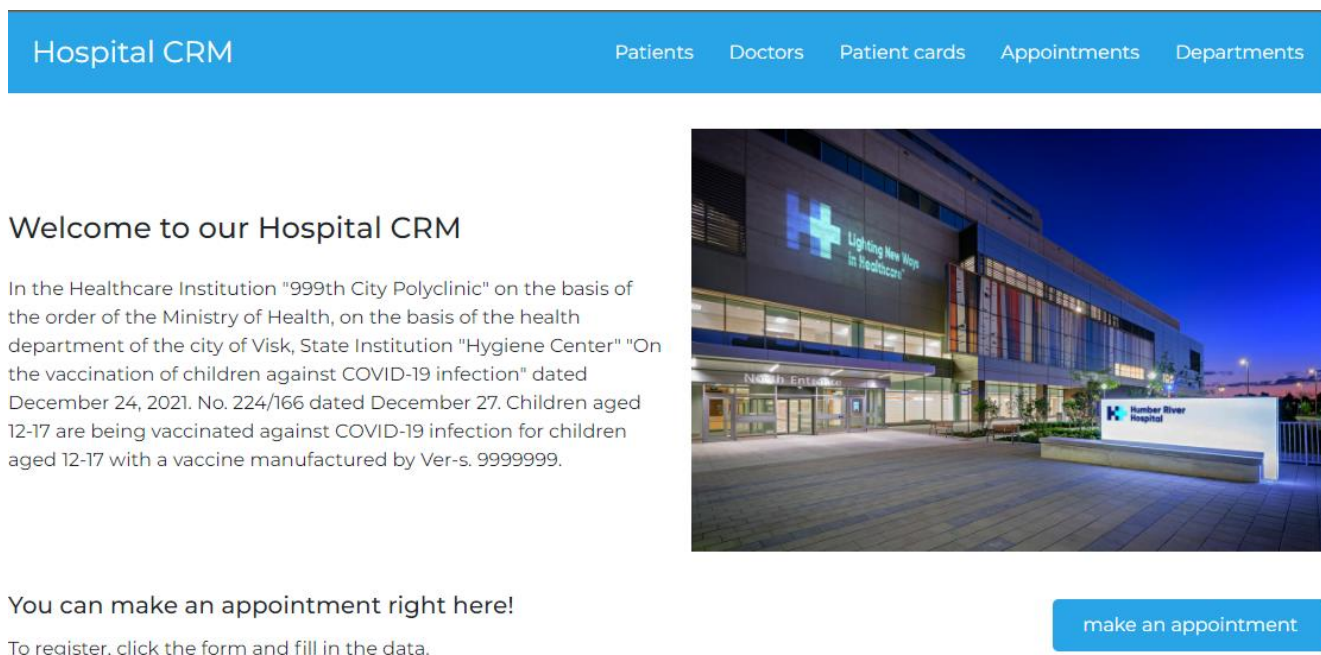


Рисунок Б.1 – Главная страница

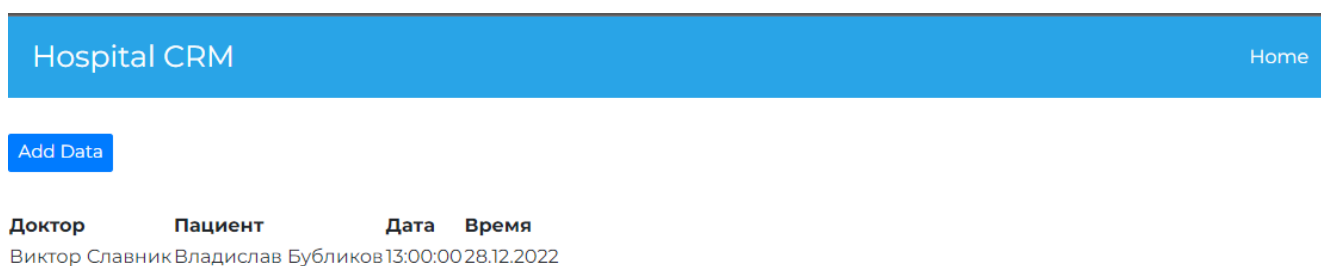
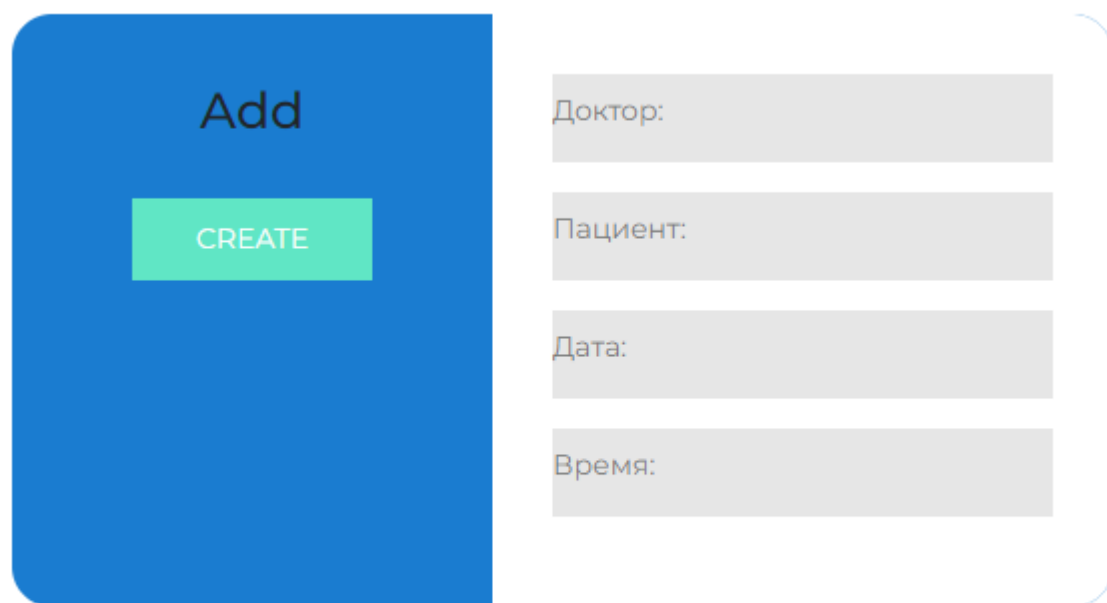


Рисунок Б.2 – Список записей



The image shows a user interface for adding a new record. On the left is a large blue rounded rectangle with the word "Add" in white text at the top and a green rounded rectangle with the word "CREATE" in white text below it. To the right of this is a white rounded rectangle containing four light gray input fields. The first field is labeled "Доктор:", the second "Пациент:", the third "Дата:", and the fourth "Время:". The entire form is enclosed in a light gray border with rounded corners.

Add

CREATE

Доктор:

Пациент:

Дата:

Время:

Рисунок Б.3 – Добавление записи