Министерство Образования Российской Федерации

Национальный исследовательский ядерный университет «МИФИ»

Кафедра «Кафедра компьютерных медецинских систем»

# Практическая работа студента

на тему:

# «Система ввода/вывода данных базы знаний СППР по онкологии предстательной железы»

**Выполнили:**
Студенты группы А10-23
Иванов Е.М
Рейнат Е.А
Калиничев С.В
Щукин.Я.И.

**Руководитель:**
Профессор НИЯУ МИФИ,
Доктор технических наук
Никитаев В.Г

# Содержание

# Глоссарий

**Электронно-вычислительная машина, ЭВМ** — комплекс технических средств, где основные элементы (логические, запоминающие, индикационные и др.) выполнены на электронных элементах, предназначенных для автоматической обработки информации в процессе решения вычислительных и информационных задач.

**Информационные технологии, ИТ** — широкий класс дисциплин и областей деятельности, относящихся к технологиям создания, сохранения, управления и обработки данных, в том числе с применением вычислительной техники(см ЭВМ).

**Стратегия** — план действий в условиях неопределенности. Это набор правил, согласно которым предпринимаемые действия должны зависеть от обстоятельств, включая естественные события и действия других людей..

**Здравоохранение** — отрасль деятельности государства, целью которой является организация и обеспечение доступного медицинского обслуживания населения, сохранение и повышение его уровня здоровья.

**Цифровые технологии** (англ. Digital technology) основаны на представлении сигналов дискретными полосами аналоговых уровней, а не в виде непрерывного спектра. Все уровни в пределах полосы представляют собой одинаковое состояние сигнала.

**Сигнал –** символ (знак, код), созданный и переданный в пространство (по каналу связи) одной системой, либо возникший в процессе взаимодействия нескольких систем. Смысл и значение сигнала проявляются в процессе дешифровки его второй (принимающей) системой.

**Дискретность** (от лат. discretus — разделённый, прерывистый) — свойство, противопоставляемое непрерывности, прерывность. Дискретность — всеобщее свойство материи, под дискретностью понимают:

1. Нечто, изменяющееся между несколькими различными стабильными состояниями, например механические часы, которые передвигают минутную стрелку дискретно (скачкообразно) на 1/60 часть окружности
2. Нечто, состоящее из отдельных частей, прерывистость, дробность. Например, дискретный спектр, дискретные структуры, дискретные сообщения.

**Аналоговый сигнал** – сигнал данных, у которого каждый из представляющих параметров описывается функцией времени и непрерывным множеством возможных значений.

**Автоматизация** – одно из направлений научно-технического прогресса, использующее саморегулирующие технические средства и математические методы с целью освобождения человека от участия в процессах получения, преобразования, передачи и использования энергии, материалов, изделий или информации, либо существенного уменьшения степени этого участия или трудоёмкости выполняемых операций.

**Онкология**(от др.-греч. ὄγκος — 'вздутие', 'припухлость' и λόγος — 'учение') — раздел медицины, изучающий доброкачественные и злокачественные опухоли, их этиологию и патогенез, механизмы и закономерности возникновения и развития, методы их профилактики, диагностики и лечения (в том числе хирургического, лучевого, химиотерапевтического, гормонального, иммунотерапевтического, фотодинамического)..

**Онкологические заболевания** – опухолевые заболевания, которые изучает онкология.

**Инновационные разработки** – разработки, содержащие технико-экономические, правовые и организационные обоснования конечной инновационной деятельности.

**Инновация, нововведение** (англ. innovation) — это внедрённое новшество, обеспечивающее качественный рост эффективности процессов или продукции, востребованное рынком. Является конечным результатом интеллектуальной деятельности человека, его фантазии, творческого процесса, открытий, изобретений и рационализации. Примером инновации является выведение на рынок продукции (товаров и услуг) с новыми потребительскими свойствами или качественным повышением эффективности производственных систем.

**Высокотехнологичная медицина** – это медицина с применением высоких медицинских технологий для лечения сложных заболеваний.

**Сегментация**— это процесс разделения цифрового изображения на несколько сегментов (множество пикселей, также называемых суперпикселями). Цель сегментации заключается в упрощении и/или изменении представления изображения, чтобы его было проще и легче анализировать. Сегментация изображений обычно используется для того, чтобы выделить объекты и границы (линии,

кривые, и т. д.) на изображениях. Более точно, сегментация изображений — это процесс присвоения таких меток каждому пикселю изображения, что пиксели с одинаковыми метками имеют общие визуальные характеристики.

**Модель** – логическое или математическое описание компонентов и функций, отображающих существенные свойства моделируемого объекта или процесса.

**Насыщенность** — это интенсивность определённого тона, то есть степень визуального отличия хроматического цвета от равного по светлоте ахроматического (серого) цвета.

**Светлота** – характеристика ощущения, согласно которой предмет кажется пропускающим или диффузно отражающим более или менее значительную долю падающего света.

**Цветовая модель** – математическая модель описания цветов на цветовом пространстве в виде последовательности конечного ряда чисел.

**Концептуальная модель**— принципиальная основа экономико-математической модели, предназначенной для реализации различными математическими и техническими средствами и, следовательно, для непосредственного решения задачи.

**Насыщенность** — это интенсивность определённого тона, то есть степень визуального отличия хроматического цвета от равного по светлоте ахроматического (серого) цвета.

**Светлота** – характеристика ощущения, согласно которой предмет кажется пропускающим или диффузно отражающим более или менее значительную долю падающего света.

**Цветовая модель** – математическая модель описания цветов на цветовом пространстве в виде последовательности конечного ряда чисел.

**Цветовой диалог** – графическое окно с различным стиллистическим оформлением(от ввода шеснадцатиричного числа до клика в круглую область цветов и настройкой интенсивности выбранного цвета) представляющее собой палитру цветов различных цветовых моделей.

**Кластер** – объединение нескольких однородных элементов, которое может рассматриваться как самостоятельная единица, обладающая определенными свойствами.

**Явление** — совокупность процессов материально-информационного преобразования, обусловленных общими причинами.

**Гистология** — раздел биологии, изучающий строение тканей живых организмов.

**Клетка** — элементарная единица строения и жизнедеятельности всех организмов, обладающая собственным обменом веществ и способная к самостоятельному существованию,самовоспроизведению и развитию.

**Алгоритм** – это последовательность арифметических, логических и прочих операций, необходимых для выполнения на ЭВМ.

**Алгоритмический язык** – формальный язык, используемый для записи, реализации и изучения алгоритмов. Всякий язык программирования является алгоритмическим языком, но не всякий алгоритмический язык пригоден для использования в качестве языка программирования.

**Бейсик** (BASIC) – семейство высокоуровневых языков программирования.

**Библиотека** (англ. library) – сборник подпрограмм или объектов, используемых для разработки программного обеспечения.

**Блок-схема** – графическое представление алгоритма, изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий.

**Ветвление** – алгоритм может пойти по одной из двух возможных ветвей. Происходит выбор одного из путей работы алгоритма.

**Вложенные циклы** (цикл в цикле) – структура, которая позволяет внутри тела цикла повторять некоторую последовательность операторов, т.е. организовать внутренний цикл.

**Высокоуровневый язык программирования** – язык программирования, разработанный для быстроты и удобства использования программистом. Основная черта высокоуровневых языков – абстракция, то есть введение смысловых инструкций, кратко описывающих такие структуры данных и операции над ними, описания которых на машинном коде очень длинны и сложны для понимания.

**Данные** – часть программы, совокупность значений определённых ячеек памяти, преобразование которых осуществляет код.

**Декларативное программирование** – описание логики алгоритма, но не управления. Программа является теорией, а вычисления представляют собой вывод в этой теории.

**Императивное программирование** – парадигма программирования, которая описывает процесс

вычисления в виде инструкций, изменяющих состояние программы. Императивная программа очень похожа на приказы, то есть это последовательность команд, которые должен выполнить компьютер.
**Индексный массив** – именованный набор однотипных переменных, расположенных в памяти непосредственно друг за другом, доступ к которым осуществляется по индексу.

**Инкапсуляция** – свойство языка программирования, позволяющее объединить и защитить данные и код в объекте и скрыть реализацию объекта от пользователя.
**Инструкция** (оператор) – наименьшая автономная часть языка программирования; команда. Программа обычно представляет собой последовательность инструкций.
**Интегрированная среда программирования** – система программных средств, используемая программистами для разработки программного обеспечения.
**Интерпретатор** (англ. interpreter – переводчик) – это программа, которая переводит каждую команду и выполняет программу построчно, что позволяет сразу редактировать и исправлять ошибки.
**Итерационные циклы** – циклы, число повторений тела которых заранее неизвестно. Выход из итерационного цикла осуществляется по выполнению некоторого условия.
**Исполнительная часть программы** – часть программы, содержащая инструкции (операторы) компьютеру на выполнение.
**Исходный код** – написанный человеком текст компьютерной программы на каком-либо языке программирования.
**Класс** – разновидность абстрактного типа данных в объектно-ориентированном программировании, характеризуемый способом своего построения.
**Кобол** – язык программирования, предназначенный для создания коммерческих приложений. Отличительной особенностью языка является возможность эффективной работы с большими массивами данных.
**Комментарии** – пояснения к исходному тексту программы, находящиеся непосредственно внутри комментируемого кода. Синтаксис комментариев определяется языком программирования.
**Компилятор** (англ. compiler – составитель) читает всю программу целиком, делает её перевод и создаёт законченный вариант программы на машинном языке, который затем и выполняется.
**Линейный алгоритм** – последовательное выполнение операций. В этом алгоритме не предусмотрены проверки условий или повторений, т.е. циклы.
**Логический тип** данных – примитивный тип данных, которые могут принимать два возможных значения, иногда называемых правдой иложью.
**Логическое программирование** – парадигма программирования, основанная на автоматическом доказательстве теорем. Логическое программирование основано на теории и аппарате математической логики с использованием математических принципов резолюций.
**Машинный код** – система команд конкретной вычислительной машины, которая интерпретируется непосредственно микропроцессором или микропрограммами данной вычислительной машины.
**Модуль** представляет собой функционально законченный фрагмент программы, оформленный в виде отдельного файла с исходным кодом или поименованной непрерывной его части, предназначенный для использования в других программах. Модули позволяют разбивать сложные задачи на более мелкие в соответствии с принципом модульности.
**Модульность** – принцип, согласно которому программное средство разделяется на отдельные именованные сущности, называемые модулями.
**Наследование** – один из четырёх важнейших механизмов объектно-ориентированного программирования, позволяющий описать новый класс на основе уже существующего, при этом свойства и функциональность родительского класса заимствуются новым классом.
**Низкоуровневый язык программирования** – язык программирования, близкий к программированию непосредственно в машинных кодах используемого реального или виртуального процессора. Для обозначения машинных команд обычно применяется мнемоническое обозначение. Это позволяет запоминать команды не в виде последовательности двоичных нулей и единиц, а в виде осмысленных сокращений слов человеческого языка.
**Объект** – некоторая сущность в виртуальном пространстве, обладающая определённым состоянием и поведением, имеет заданные значения свойств и операций над ними.
**Объектно-ориентированное** или объектное **программирование** – парадигма программирования, в которой основными концепциями являются понятия объектов и классов.

**Оператор ветвления** (условная инструкция, условный оператор) – оператор, конструкция языка программирования, обеспечивающая выполнение определённой команды только при условии истинности некоторого логического выражения, либо выполнение одной из нескольких команд в зависимости от значения некоторого выражения.
**Описательная часть программы** – часть программы, которая используется для описания переменных, констант, пользовательских типов, меток.
**Отладчик** (англ. debugger) – модуль среды разработки или отдельное приложение, предназначенное для поиска ошибок в программе. Отладчик позволяет выполнять пошаговую трассировку, отслеживать, устанавливать или изменять значение переменных в процессе выполнения программы, устанавливать и удалять контрольные точки или условия остановки и т.д.
**Парадигма программирования** – совокупность идей и понятий, определяющая стиль написания программ.
**Паскаль** – широко распространённый язык структурного программирования. В этом языке внедрена строгая проверка типов, что позволило выявлять многие ошибки на этапе компиляции. Также впервые оператор безусловного перехода перестал играть основополагающую роль при управлении порядком выполнения операторов.
**Переменная** – поименованная, либо адресуемая иным способом область памяти, имя или адрес которой можно использовать для осуществления доступа к данным, находящимся в переменной.
**Подпрограмма** – поименованная или иным образом идентифицированная часть компьютерной программы, содержащая описание определённого набора действий. Подпрограмма может быть многократно вызвана из разных частей программы. В языках программирования для оформления и использования подпрограмм существуют специальные синтаксические средства.
**Полиморфизм** – возможность объектов с одинаковой спецификацией иметь различную реализацию.
**Присваивание** – механизм в программировании, позволяющий динамически изменять связи объектов данных с их значениями.
**Пролог** (фр. Programmation en Logique) – язык и система логического программирования, основанные на языке предикатов математической логики дизъюнктов Хорна.
**Программа** – запись алгоритма на языке программирования, приводящая к конечному результату за конечное число шагов.
**Программирование** – процесс создания компьютерных программ.
**Процедура** – любая подпрограмма, которая не является функцией.
**Процедурный язык программирования** предоставляет возможность программисту определять каждый шаг в процессе решения задачи. Особенность таких языков программирования состоит в том, что задачи разбиваются на шаги и решаются шаг за шагом. Используя процедурный язык, программист определяет языковые конструкции для выполнения последовательности алгоритмических шагов.
**Семантика** – система правил определения поведения отдельных языковых конструкций. Семантика определяет смысловое значение предложений алгоритмического языка.
**Си** (англ. C) – стандартизированный процедурный язык программирования, разработанный в начале 1970-х годов сотрудниками Bell Labs Кеном Томпсоном и Денисом Ритчи как развитие языка Би.
**Синтаксис** – сторона языка программирования, которая описывает структуру программ как наборов символов. Синтаксису языка противопоставляется его семантика.
**Скриптовый язык** (язык сценариев) – язык программирования, разработанный для записи «сценариев», последовательностей операций, которые пользователь может выполнять на компьютере. Сценарии обычно интерпретируются, а не компилируются.
**Следование** – тип вычислительного процесса, при котором действия выполняются строго в том порядке, в котором записаны.
**Структура данных** – программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных. Для добавления, поиска, изменения и удаления данных структура данных предоставляет некоторый набор функций, составляющих интерфейс структуры данных.
**Структурное программирование** – методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков.
**Тип переменной** определяет объём оперативной памяти, выделяемой под хранение переменной.
**Транслятор** – это программа, которая преобразует исходную программу (написанную на одном из языков высокого уровня) в программу, состоящую из машинных команд.

**Фортран** – первый язык программирования высокого уровня. Ключевой идеей, отличающей новый язык от ассемблера, была концепция подпрограмм. Впервые программист смог по-настоящему абстрагироваться от особенностей машинной архитектуры.
**Функциональное программирование** объединяет разные подходы к определению процессов вычисления на основе достаточно строгих абстрактных понятий и методов символьной обработки данных.
**Функция** – поименованная часть программы, которая может вызываться из других частей программы столько раз, сколько необходимо. Функция обязательно возвращает значение.
**Цикл с параметром** – цикл, в котором тело цикла выполняется для всех значений некоторой переменной (параметра цикла) в заданном диапазоне.
**Цикл с постусловием** – цикл, в котором тело цикла выполняется до тех пор, пока не выполнится условие.
**Цикл с предусловием** – цикл, в котором тело цикла выполняется до тех пор, пока условие выполняется.
**Циклический алгоритм** предусматривает многократное повторение одной или нескольких операций в зависимости от условия задачи.
**Язык ассемблера** – язык программирования низкого уровня, мнемонические команды которого соответствуют инструкциям процессора вычислительной системы.
**Языки программирования** – формальные языки, специально созданные для общения человека с вычислительной машиной.

**Однослойные эпителии –** все клетки располагаются на базальной мембране, причем ядра клеток ***однорядных*** эпителиев находятся на одном уровне, а ядра клеток ***многорядного*** эпителия находятся на разных уровнях, что создает эффект многорядности (и ложное впечатление многослойности).

1. ***Однослойный плоский эпителий*** образован уплощенными клетками полигональной формы с утолщением в области расположения дисковидного ядра. На свободной поверхности клетки имеются единичные микроворсинки. Примером такого типа является эпителий (мезотелий), покрывающий легкое (висцеральная плевра) и эпителий выстилающий изнутри грудную полость (париетальная плевра), а также париетальный и висцеральный листки брюшины, околосердечная сумка.

2. ***Однослойный кубический эпителий*** образован клетками, содержащими ядро сферической формы. Такой эпителий встречается в фолликулах щитовидной железы, в мелких протоках поджелудочной железы и желчных протоках, в почечных канальцах.

3. ***Однослойный призматический (цилиндрический) эпителий*** (рис.1)образован клетками с резко выраженной *полярностью*. Ядро эллипсовидной формы лежит вдоль длинной оси клетки и смещено к их базальной части, хорошо развитые органеллы неравномерно распределены по цитоплазме. На апикальной поверхности находятся *микроворсинки, щеточная каемка*. Этот вид эпителия характерен для среднего отдела пищеварительного канала и выстилает внутреннюю поверхность тонкой и толстой кишки, желудка, желчного пузыря, ряда крупных протоков поджелудочной железы и желчных протоков печени. Для этого вида эпителия характерны функции *секреции и (или) всасывания*.

В эпителии тонкой кишки встречаются два основных типа дифференцированных клеток – *призматические каемчатые,* обеспечивающие пристеночное пищеварение, и *бокаловидные,* вырабатывающие слизь. Такое неодинаковое строение и функции клеток в однослойном эпителии называется ***горизонтальной*****анизоморфностью.**

4. ***Многорядный реснитчатый (мерцательный) эпителий воздухоносных путей*** образован клетками нескольких типов: 1) низкие вставочные (базальные), 2) высокие вставочные (промежуточные), 3) реснитчатые (мерцательные), 4) бокаловидные. Низкие вставочные клетки являются камбиальными, своим широким основанием они прилежат к базальной мембране, а узкой апикальной частью не доходят до просвета. Бокаловидные клетки вырабатывают слизь, которая покрывает поверхность эпителия, перемещаясь по ней благодаря биению ресничек мерцательных клеток. Апикальные части этих клеток граничат с просветом органа.

**Многослойные эпителии**– эпителии, в которых лишь клетки, образующие базальный слой, располагаются на базальной мембране. Клетки, входящие в состав остальных слоев, утрачивают с ней связь. Многослойным эпителиям свойственна **вертикальная анизоморфность** – неодинаковые

морфологические свойства клеток различных слоев эпителиального пласта. В основу классификации многослойных эпителиев положена форма клеток поверхностного слоя.

Поддержание целостности многослойных эпителиев обеспечивается регенерацией. Эпителиоциты непрерывно делятся в самом глубоком базальном слое за счет стволовых клеток, далее идет смещение в вышележащие слои. После дифференцировки происходит дегенерация и слущивание клеток с поверхности пласта. Процессы ***пролиферации*** и ***дифференцировки*** эпителиальных клеток регулируются рядом биологически активных веществ, часть которых выделяется клетками подлежащей соединительной ткани. Наиболее важным из них являются цитокины, в частности эпидермальный фактор роста; оказывают влияние гормоны, медиаторы и другие факторы.Дифференцировка эпителиоцитов сопровождатся изменением экспрессии синтезируемых ими цитокератинов, образующих промежуточные филаменты.

**Многослойные плоские эпителии**в зависимости от наличия или отсутствия рогового слоя подразделяются на ***ороговевающие и неороговевающие.***

*1.* ***Многослойный плоский ороговевающий эпителий*** *(рис.3) образует наружный слой кожи - эпидермис, и покрывает некоторые участки слизистой оболочки полости рта. Он состоит из пяти слоев:*

***Базальный слой*** (1) образован клетками кубической или призматической формы, лежащими на базальной мембране. Они способны к митотическому делению, поэтому за счет них происходит смена вышележащих слоев эпителия.

**Шиповатый слой** (2) образован крупными клетками неправильной формы. В глубоких слоях могут встречаться делящиеся клетки. В базальном и шиповатом слоях хорошо развиты тонофибриллы (пучки тонофиламентов), а между клетками десмосомальные, плотные, щелевидные контакты.

**Зернистый слой** (3) состоит из уплощенных клеток, в цитоплазме которых содержатся зерна кератогиалина – фибриллярного белка, который в процессе ороговенияпревращается в элеидинкератин.

**Блестящий слой** (4) выражен только в эпителии толстой кожи, покрывающей ладони и подошвы. Он представляет собой зону перехода от живых клеток зернистого слоя к чешуйкам рогового слоя, не обладающим признаками живых клеток. На гистологических препаратах он имеет вид узкой оксифильной гомогенной полоски и состоит из уплощенных клеток. В блестящем слое завершаются процессы ***ороговения***, который заключается в превращении живых эпителиальных клеток в роговые чешуйки – механически прочные и химически устойчивые постклеточные структуры, образующие в совокупности ***роговой слой*** эпителия, который выполняет защитные функции. Хотя собственно формирование роговых чешуек происходит в наружных отделах зернистого слоя или в блестящем слое, синтез веществ, обеспечивающих ороговение, осуществляется уже в шиповатом слое.

***Роговой слой*** (5)наиболее поверхностный и имеет максимальную толщину в эпидермисе кожи в области ладоней и подошв. Он образован плоскими ***роговыми чешуйками*** с резко утолщенной плазмолеммой. Клетки не содержат ядра и органелл и заполнены сетью из толстых пучков кератиновых филаментов, погруженных в плотный матрикс. Роговые чешуйки в течение определенного времени сохраняют связи друг с другом и удерживаются в составе пластов благодаря частично сохраненным десмосомам, а также взаимному проникновению бороздок и гребешков, образующих ряды на поверхности соседних чешуек. В наружных частях рогового слоя десмосомы разрушаются, и роговые чешуйки слущиваются с поверхности эпителия.

2. ***Многослойный плоский неороговевающий эпителий*** покрывает поверхность роговицы глаза, слизистой оболочки полости рта, пищевода, влагалища. Он образован тремя слоями:

1) ***Базальный слой*** аналогичен по строению и функции соответствующему слою ороговевающего эпителия.

2) ***Шиповатый слой*** образован крупными полигональными клетками, которые по мере приближения к поверхностному слою уплощаются. Их цитоплазма заполняется многочисленными тонофиламентами, которые располагаются диффузно. В более наружных клетках этого слоя накапливается кератогиалин в виде мелких округлых гранул.

3) ***Поверхностный слой*** нерезко отделен от шиповатого. Содержание органелл снижено по сравнению с таковым в клетках шиповатого слоя, плазмолемма утолщена, ядро с плохо различимыми гранулами хроматина (пикнотическое). При десквамации клетки этого слоя постоянно удаляются с поверхности эпителия.

3. ***Переходный эпителий –*** особый вид многослойного эпителия, который выстилает большую часть мочевыводящих путей. Он образован тремя слоями:

1) ***Базальный слой*** образован мелкими клетками, имеющими на срезе треугольную форму и своим широким основанием прилежат к базальной мембране.

2) ***Промежуточный слой*** состоит из удлиненных клеток, более узкой частью направленных к базальному слою и черепицеобразно накладывающихся друг на друга.

3) ***Поверхностный слой*** образован крупными одноядерными полиплоидными или двуядерными клетками, которые в наибольшей степени изменяют свою форму при растяжении эпителия (от округлой до плоской). Этому способствует формирование в апикальной части цитоплазмы этих клеток в состоянии покоя многочисленных инвагинаций плазмолеммы и особых дисковидных пузырьков – резервов плазмолеммы, которые встраиваются в нее по мере растяжения органа и клеток.

**Регенерация покровных эпителиев.** Покровный эпителий, занимая пограничное положение, постоянно испытывает влияние внешней среды, поэтому эпителиальные клетки быстро изнашиваются и погибают. Восстановление эпителия – ***физиологическая регенерация*** – происходит путем митотического деления клеток. В однослойном эпителии большинство клеток способны к делению, а в многослойном такой способностью обладают только клетки базального и частично шиповатого слоев. Высокая способность эпителия к физиологической регенерации служит основой для быстрого восстановления его в патологических условиях -***репаративная регенерация.***

**Ацинусы** - мелкие пузырьки с выходными отверстиями котроые переходят в длинные извитые выводные протоки. Выводные протоки ацинусов открываются отверстиями на задней поверхности простатической части мочеиспусательного канала а области семенного холмика. Анатомически предстательная железа состоит из 30-50 ацинусов окруженных мышечными волокнами, кровеносными и лимфатическими сосудами. Ацинусы предстательной железы покрыты специальными клетками - железистым эпителием, который и вырабатывает секрет.

**Гломеруляции** — это типичные округлые подслизистые геморрагические образования, их можно заметить в нескольких различных квадрантах мочевого пузыря

Инфильтративный рост характеризуется распространением опухолевых элементов в направлении наименьшего сопротивления и врастанием в окружающие ткани, разрушая их.

**Периневральная** или интраневральная **инвазия** Распространение опухолевых элементов за пределы предстательной железы, т. е. обнаружение их в жировой клетчатке

# Введение

Развитие электронно-вычислительных машин и информационных технологий обеспечило создание и широкое использование систем во многих областях.

Ключевая стратегия усовершенствования здравоохранения России – это переориентирование на современные цифровые технологии автоматизированной фиксации, хранения и обработки медицинских изображений различных органов человека. В связи с отмечанием увеличения количества онкологических заболеваний, лечение которых результативно исключительно на начальных стадиях и ростом совокупности диагностической информации, выраженной в форме графических изображений. Вследствие чего к областям важнейшего направления, где крайне необходимы инновационные разработки, причисляется также и высокотехнологичная медицина: специальное оборудование, уникальные приборы, новые методы диагностирования и лечения заболеваний. Одно из стратегических направлений развития – это увеличение профилактической нацеленности здравоохранения, что создает необходимость создания современных диагностических комплексов с высокой пропускной способностью для массовой диспансеризации и выявления разного рода патологий на самых ранних стадиях. Оперативное и достоверное диагностирование недостижимо без использования компьютерных средств изучения и алгоритмов переобработки первичной медицинской информации. Зачастую снимки, полученные при первоначальной диагностике, необходимо непремерно обработать на ЭВМ, так как точное диагностирование визуальными методами чересчур затруднено. Немаловажный недостатком отечественной диагностики является недостаточное алгоритмическое и программное оснащение средств ее поддержки.

Сегодня значительная часть специализированных систем только предоставляет изображение или последовательность изображений, не только не осуществляя их компьютерный анализ, который мог бы помочь провести распознавание, но даже и не сохраняя уникальные клинические сведения. Таким образом, продуктивность эксплуатации автоматизированных диагностических методов максимально зависит от квалификации исследователя и требует от него визуально осуществлять всю надобную в диагностике сегментацию.

# Актуальность темы

В онкологической медецине задачей наивысшего приоретета является распознавание заболеваний на самых ранних стадиях, поскольку существуют формы заболеваний при которых смерть происходит за считанные сутки.

Большинство видов онкологий, выявленного на первой стадии, излечимо практически на 100%. Один из основных показателей радикальности лечения рака – пятилетняя выживаемость. Больные, пережившие этот срок, имеют такую же продолжительность жизни, как их сверстники.

Прогноз при раке простаты на начальных стадиях после проведения радикальной простатэктомии: пятилетняя выживаемость в 74-84% случаев, десятилетняя – в 55-56% случаев.

После радиотерапии пятилетняя выживаемость составляет 72-80%, а десятилетняя – 48%. У пациентов, находящихся на гормонотерапии после орхиэктомии пятилетняя выживаемость составляет не более 55%.

При ранней диагностики онкологических заболеваний уже давно использует компьютерная автоматизированная система сегментации изображения как «оружие первой линии». Онкоклиники успешно и широко применяет весть известный высокоинформативный аналитический апарат при подозрении на рак различных органов или в профилактических целях, что дает возможность выявить опухоль на самой ранней стадии ее развития, когда, порой, простая визуализационная диагностика не в силах с уверенностью дать точный ответ.

Компьютерная автоматизированная система сегментации изображения(КАССИ) – это компьютерная система, которая присваивает пикселям изображения метки так, что области, обладающие сходными в определенном смысле свойствами, имеют одинаковые метки. Также КАССИ порой называют поиск проекции объекта на изображении. Автоматизированные системы позволяют облегчить визуальную диффиренцацию полезной информации из первоисточников, проанализировать ее, а также выявить существующие модели для решения определенных задач..

## Цель работы

Разработать систему по вводу/выводу данных базы знаний СППР по онкологии предстательной железы.

## Задачи работы

- Исследовать предметную область, т.е. изучить существующие алгоритмы и их классификацию
- Реализовать  алгоритм аутентификации
- Реализовать алгоритм заполнения полей СУБД представляющей собой интерфейс для базы знаний
- Реализовать алгоритм запроса полей
- Заключение по анализу алгоритма аутентификации
- Заключение по анализу алгоритмов заполнения
- Заключение по анализу алгоритмов запроса
- Реализовать алгоритмы сегментации
- Алгоритм визуального отображения
- Вывод

# Исходные данные

К исходным данным работы относятся:

- 1000 изображений гистологических снимков предстательной железы
- 78 морфологических признаков для анализа микрофотографий, которые относятся к секреторному эпителию, ацинусам и без относительного заболевания.
- теоретические основы о микрофотографиях простаты
- теоретический материал разработки кроссплатформенного ПО

# Глава 1. Теоретическая часть

## 1.1 Анализ предметной области

### 1.1.1 Изучение подобных систем по онкологии предстательной железы

Рак предстательной железы у мужчин является наиболее распространённой злокачественной опухолью. На что уделяют большое внимание врачи. Причины развития заболевания окончательно не изучены. Известно, что некоторые факторы повышает возможность развития рака простаты. Как правило, начальные стадии рака предстательной железы протекают бессимптомно. Первые признаки появляются уже в далеко зашедших стадиях заболевания. Основными методами диагностики рака предстательной железы являются:

1. Пальцевое исследование прямой кишки является наиболее доступным методом диагностики рака простаты. Во время этого исследования врач ощупывает предстательную железу через стенку прямой кишки. С помощью такого исследования врач определяет размеры и плотность предстательной железы, а также обнаруживает наличие опухоли простаты.
2. Определение в крови уровня простат-специфического антигена (ПСА) является распространенным методом диагностики данного заболевания. ПСА представляет собой особое вещество, уровень которого повышается при раке простаты, доброкачественной гиперплазии предстательной железы (заболевание, которое характеризуется разрастанием тканей простаты, увеличением размеров органа и сдавлением мочеиспускательного канала), а также при простатитах (воспаление простаты). Таким образом, повышенный уровень ПСА еще не является гарантией наличия рака. Определение уровня ПСА является также скрининговым методом диагностики рака простаты – это означает, что тест проводят всем мужчинам, у которых имеется повышенный риск развития заболевания. Таким образом, этот метод позволяет выявить рак простаты на ранних стадиях, когда еще не имеется симптомов заболевания.
3. УЗИ простаты широко применяется в диагностике. Существует 2 основных способа проведения УЗИ: через переднюю брюшную стенку (трансабдоминальное УЗИ) и через прямую кишку (трансректальное УЗИ). Трансректальное УЗИ (или ТРУЗИ) является более эффективным методом диагностики, так как позволяет выявить даже небольшие опухоли в простате.
4. Биопсия – это наиболее эффективный метод диагностики, который позволяет установить диагноз. Биопсия представляет собой получение участка опухоли для дальнейшего изучения под микроскопом. Существует несколько способов осуществления биопсии при раке предстательной железы: через стенку прямой кишки, через кожу промежности, либо через уретру (мочеиспускательный канал). Как правило, биопсия производится под контролем УЗИ-аппарата.
5. Если в результате исследований был установлен рак простаты, назначаются дополнительные анализы, которые помогают выяснить степень распространения (стадию рака простаты). К таким исследованиям относят: рентгенографию костей, сцинтиграфию (определение степени накопления специального вещества в костях, которое позволяет выявить наличие метастазов), компьютерную томографию и др.

#### *Популяционное исследование рака почки и рака предстательной железы в некоторых пациентах*

(A population-based study of renal cell carcinoma and prostate cancer in the same patients)

DANIEL A. BAROCAS, FARHANG RABBANI', DOUGLAS S. SCHERR and E. DARRACOTT VAUGHAN Jr Departments of Urology, James Buchanan Brody Foundation. New York-Presbyterian Hospital - Weill Medical College of Cornell University and "Memorial Sloan-Kettering Cancer Center, New York, NY, USA Accepted for publication 5 August 2005

Аннотация

Для исследования заболеваемости раком простаты у мужчин с почечно-клеточного рака (ПКР) и заболеваемости RCC у мужчин с раком предстательной железы.

Методы

Мы оценили базу данных программы наблюдения, эпидемиологии и конечных результатов Национального института рака с 1973 по 1996, чтобы вычислить частоту РСС у мужчин с раком простаты и заболеваемость раком простаты у мужчин с ПКР. Стандартизированный коэффициент заболеваемости (SIR, наблюдается / ожидаемое) был вычисления для каждого из сценария интерес, а также для РСС и рака простаты у мужчин с другими общими злокачественных образований в легких / раком толстой кишки бронхов / ректального рака неходжкинская лимфома были выбраны для сценарий управления, потому что это наиболее распространенный, не урологические рака среди меня в США

Результаты

Был выше заболеваемость RC у мужчин с раком предстательной железы (SIR 1,25, P <0,01) RC заболеваемости был также выше у мужчин с каждым из других злокачественных опухолей. Заболеваемость раком простаты был выше у мужчин с ПКР (SIR 1,42, P <0,001), но не был значительно повышен для любого из сценариев управления.

## 1.1.2 Общие сведения о СУБД

С самого начала развития вычислительной техники образовались два основных направления ее использования. Первое направление - применение вычислительной техники для выполнения численных расчетов, которые слишком долго или вообще невозможно производить вручную. Становление этого направления способствовало интенсификации методов численного решения сложных математических задач, развитию класса языков программирования, ориентированных на удобную запись численных алгоритмов, становлению обратной связи с разработчиками новых архитектур ЭВМ.

Второе направление - это использование средств вычислительной техники в автоматических или автоматизированных информационных системах. Обычно объемы информации, с которыми приходится иметь дело таким системам, достаточно велики, а сама информация имеет достаточно сложную структуру. Одними из естественных требований к таким системам являются средняя быстрота выполнения операций и сохранность информации.

### Файловые системы

Файловые системы явились первой попыткой компьютеризировать известные всем ручные картотеки. Подобная картотека (или подшивка документов) в некоторой организации могла содержать всю внешнюю и внутреннюю документацию, связанную с каким-либо проектом, продуктом, задачей, клиентом или сотрудником. Так как обычно таких папок бывает очень много, то для поиска какой-либо информации, нам необходимо просмотреть всю картотеку от начала и до конца. Более изощренный подход предусматривает использование в такой системе некоторого алгоритма индексирования, позволяющего ускорить поиск нужных сведений. Например, можно использовать разделители или отдельные папки для различных логически связанных типов объектов.

Ручные картотеки позволяют успешно справляться с поставленными задачами, если количество хранимых объектов, которые нужно только хранить и извлекать, невелико. Однако они совершенно не подходят для тех случаев, где нужно выполнить перекрестные связи или выполнить обработку сведений. Файловые системы были разработаны в ответ на потребность в получении более эффективных способов доступа к данным. Однако, вместо организации централизованного хранилища всех данных предприятия, был использован децентрализованный подход, при котором сотрудники каждого отдела работают со своими собственными данными и хранят их в своем отделе.

Совершенно очевидно, что большое количество данных в отделах дублируется, что весьма характерно для любых файловых систем. Это сопровождается неэкономным расходованием ресурсов, поскольку на ввод избыточных данных необходимо затрачивать время и деньги. Более того, для их хранения необходимо дополнительное место во внешней памяти, что связано с увеличением накладных расходов. И хотя во многих случаях дублирования можно избежать за счет совместного использования файлов, такой подход не всегда реализуется, из-за невозможности одновременного к ним обращения. Еще более важен тот факт, что дублирование данных может привести к нарушению их целостности. Иначе говоря, данные в разных отделах могут стать противоречивыми. Например, некий сотрудник получает повышение по службы с соответствующим увеличением заработной платы. Если это изменение будет зафиксировано только в информации отдела кадров, оставшись не проведенным в файлах расчетного сектора, то данный сотрудник будет ошибочно получать прежнюю заработную плату. И даже если сотрудники расчетного сектора вовремя внесут необходимые изменения, все равно существует вероятность неправильного их ввода.

Кроме того, физическая структура и способ хранения записей файлов данных жестко зафиксированы в коде программ приложений. Это значит, что изменить существующую структуру данных достаточно сложно. Например, увеличение в файле длины какого-то поля на один символ кажется совершенно незначительным изменением его структуры, но для воплощения этого изменения потребуется, как минимум, создать программу преобразования файла в новый формат. Помимо этого, все обращающиеся к этому файлу программы должны быть изменены с целью соответствия новой структуре файла. Причем таких программ может быть очень много. Следовательно, программист должен, прежде всего, выявить их все, а затем проверить и внести необходимые изменения Данная особенность файловых систем называется зависимостью от программ и данных.

Одним словом, файловые системы обычно обеспечивают хранение слабо структурированной информации (например, текстовых данных: документов, текстов программ и т.д.), оставляя дальнейшую структуризацию прикладным программам. В некоторых случаях это даже хорошо, так как при разработке любой новой прикладной системы опираясь на простые, стандартные и сравнительно дешевые средства файловой системы можно реализовать те структуры хранения, которые наиболее естественно соответствуют специфике данной прикладной области.

## Потребности информационных систем

Однако ситуация коренным образом отличается для упоминавшихся выше информационных систем. Эти системы главным образом ориентированы на хранение, выбор и модификацию постоянно существующей информации. Структура информации зачастую очень сложна, и хотя структуры данных различны в разных информационных системах, между ними часто бывает много общего. На начальном этапе использования вычислительной техники для управления информацией проблемы структуризации данных решались индивидуально в каждой информационной системе. Производились необходимые надстройки над файловыми системами (библиотеки программ), подобно тому, как это делается в компиляторах, редакторах и т.д.

Но поскольку информационные системы требуют сложных структур данных, эти индивидуальные дополнительные средства управления данными являлись существенной частью информационных систем и практически повторялись от одной системы к другой. Стремление выделить и обобщить общую часть информационных систем, ответственную за управление сложно структурированными данными, и явилось, судя по всему, первой побудительной причиной создания СУБД. Очень скоро

стало понятно, что невозможно обойтись общей библиотекой программ, реализующей над стандартной базовой файловой системой более сложные методы хранения данных, например, хранение информации в нескольких файлах.Фактически, если информационная система поддерживает согласованное хранение информации в нескольких файлах, можно говорить о том, что она поддерживает базу данных. Если же некоторая вспомогательная система управления данными позволяет работать с несколькими файлами, обеспечивая их согласованность, можно назвать ее системой управления базами данных. Уже только требование поддержания согласованности данных в нескольких файлах не позволяет обойтись библиотекой функций: такая система должна иметь некоторые собственные данные (метаданные) и даже знания, определяющие целостность данных.

Но это еще не все, что обычно требуют от СУБД. Как уже указывалось выше, файловые системы имеют жесткий и весьма ограниченный набор запросов, "зашитый" в управляющую программу. Современные СУБД способны реализовывать произвольно сформулированные запросы на близком пользователю языке. Такие языки называются *языками запросов к базам данных*. На данный момент самым распространенным языком запросов является SQL.

Далее, представьте себе, что в нашей первоначальной реализации информационной системы, основанной на использовании библиотек расширенных методов доступа к файлам, обрабатывается операция добавления информации сразу в несколько файлов. Следуя требованиям согласованного изменения файлов, информационная система вставила новую запись в первый файл и собиралась модифицировать запись другого, но именно в этот момент произошло аварийное выключение питания. Очевидно, что после перезапуска системы ее база данных будет находиться в рассогласованном состоянии. Потребуется выяснить это и привести информацию в согласованное состояние. Настоящие СУБД берут такую работу на себя. Прикладная система не обязана заботиться о корректности состояния базы данных.

Наконец, представим себе, что мы хотим обеспечить параллельную (например, многотерминальную) работу с базой данных сотрудников. Если опираться только на использование файлов, то для обеспечения корректности на все время модификации любого из двух файлов доступ других пользователей к этому файлу будет блокирован. Настоящие СУБД обеспечивают гораздо более тонкую синхронизацию параллельного доступа к данным.

Таким образом, СУБД решают множество проблем, которые затруднительно или вообще невозможно решить при использовании файловых систем. При этом существуют приложения, для которых вполне достаточно файлов; приложения, для которых необходимо решать, какой уровень работы с данными во внешней памяти для них требуется; и приложения, для которых, безусловно, нужны базы данных.

## Функции СУБД. Типовая организация СУБД.

Как было показано ранее, традиционных возможностей файловых систем оказывается недостаточно для построения даже простых информационных систем. Мы выявили несколько потребностей, которые не покрываются возможностями систем управления файлами: поддержание логически согласованного набора файлов; обеспечение языка манипулирования данными; восстановление информации после разного рода сбоев; реально параллельная работа нескольких пользователей. Можно считать, что если прикладная информационная система опирается на некоторую систему управления данными, обладающую этими свойствами, то эта система управления данными является *системой управления базами данных (СУБД)*.

## Основные функции СУБД

Более точно, к числу функций СУБД принято относить следующие:

**1. Непосредственное управление данными во внешней памяти.** Эта функция включает обеспечение необходимых структур внешней памяти как для хранения данных, непосредственно входящих в БД, так и для служебных целей, например, для уменьшения времени доступа к данным в некоторых случаях (обычно для этого используются индексы). В некоторых реализациях СУБД активно используются возможности существующих файловых систем, в других работа производится вплоть до уровня устройств внешней памяти. Но подчеркнем, что в развитых СУБД пользователи в любом случае не обязаны знать, использует ли СУБД файловую систему, и если использует, то как организованы файлы. В частности, СУБД поддерживает собственную систему именования объектов БД.

**2. Управление буферами оперативной памяти**. СУБД обычно работают с БД значительного размера; по крайней мере, этот размер обычно существенно больше доступного объема оперативной памяти. Понятно, что если при обращении к любому элементу данных будет производиться обмен с внешней памятью, то вся система будет работать со скоростью устройства внешней памяти. Практически единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти. При этом, даже если операционная система производит общесистемную буферизацию (как в случае ОС UNIX), этого недостаточно для целей СУБД, которая располагает гораздо большей информацией о полезности буферизации той или иной части БД. Поэтому в развитых СУБД поддерживается собственный набор буферов оперативной памяти с собственной дисциплиной замены буферов.

**3**. **Управление транзакциями** Транзакция - это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется, и СУБД фиксирует изменения БД, произведенные этой транзакцией, во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД. Понятие транзакции необходимо для поддержания логической целостности БД. Например, в случае информационной системы отдела кадров при приеме на работу нового сотрудника необходимо новую информацию как в файл учета сотрудников, так и в файл отдела, в который этого сотрудника приняли. Единственным способом не нарушить целостность БД в этом случае, это при выполнении операции приема на работу нового сотрудника является объединение элементарных операций над файлами СОТРУДНИКИ и ОТДЕЛЫ в одну транзакцию.

**4. Журналирование**. Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя. Обычно рассматриваются два возможных вида аппаратных сбоев: так называемые мягкие сбои, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания), и жесткие сбои, характеризуемые потерей информации на носителях внешней памяти. Примерами программных сбоев могут быть: аварийное завершение работы СУБД (по причине ошибки в программе или в результате некоторого аппаратного сбоя) или аварийное завершение пользовательской программы, в результате чего некоторая транзакция остается незавершенной. Первую ситуацию можно

рассматривать как особый вид мягкого аппаратного сбоя; при возникновении последней требуется ликвидировать последствия только одной транзакции.

Понятно, что в любом случае для восстановления БД нужно располагать некоторой дополнительной информацией. Другими словами, поддержание надежности хранения данных в БД требует избыточности хранения данных, причем та часть данных, которая используется для восстановления, должна храниться особо надежно. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД.

Журнал - это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью, в которую поступают записи обо всех изменениях основной части БД. В разных СУБД изменения БД журналируются на разных уровнях: иногда запись в журнале соответствует некоторой логической операции изменения БД (например, операции удаления строки из таблицы реляционной БД), иногда - минимальной внутренней операции модификации страницы внешней памяти; в некоторых системах одновременно используются оба подхода.

Во всех случаях придерживаются стратегии "упреждающей" записи в журнал (так называемого протокола Write Ahead Log - WAL). Грубо говоря, эта стратегия заключается в том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем измененный объект попадет во внешнюю память основной части БД. Известно, что если в СУБД корректно соблюдается протокол WAL, то с помощью журнала можно решить все проблемы восстановления БД после любого сбоя.

Самая простая ситуация восстановления - индивидуальный откат транзакции. Строго говоря, для этого не требуется общесистемный журнал изменений БД. Достаточно для каждой транзакции поддерживать локальный журнал операций модификации БД, выполненных в этой транзакции, и производить откат транзакции путем выполнения обратных операций, следуя от конца локального журнала.

**5. Поддержка языков БД**. Для работы с базами данных используются специальные языки, в целом называемые языками баз данных. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков. Чаще всего выделялись два языка - язык определения схемы БД (SDL - Schema Definition Language) и язык манипулирования данными (DML - Data Manipulation Language). SDL служил главным образом для определения логической структуры БД, т.е. той структуры БД, какой она представляется пользователям. DML содержал набор операторов манипулирования данными, т.е. операторов, позволяющих заносить данные в БД, удалять, модифицировать или выбирать существующие данные.

В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных. Стандартным языком наиболее распространенных в настоящее время реляционных СУБД является язык SQL (Structured Query Language). Перечислим основные функции реляционной СУБД, поддерживаемые на "языковом" уровне, т.е. функции, поддерживаемые при реализации интерфейса SQL (если читатель не знаком с основами реляционной модели данных, следует сначала ознакомиться с ней в главе 3.5 и лишь после этого рассматривать основы языка SQL).

Прежде всего, язык SQL сочетает средства SDL и DML, т.е. позволяет определять схему реляционной БД и манипулировать данными. При этом именование объектов БД (для реляционной БД - именование таблиц и их столбцов) поддерживается на языковом уровне в том смысле, что компилятор языка SQL производит преобразование имен объектов в их внутренние идентификаторы на основании специально поддерживаемых служебных таблиц-каталогов. Внутренняя часть СУБД (ядро) вообще не работает с именами таблиц и их столбцов.

Язык SQL содержит специальные средства определения ограничений целостности БД. Опять же, ограничения целостности хранятся в специальных таблицах-каталогах, и обеспечение контроля целостности БД производится на языковом уровне, т.е. при компиляции операторов модификации БД компилятор SQL на основании имеющихся в БД ограничений целостности генерирует соответствующий программный код.

Специальные операторы языка SQL позволяют определять так называемые представления БД, фактически являющиеся хранимыми в БД запросами (результатом любого запроса к реляционной БД является таблица) с именованными столбцами. Для пользователя представление является такой же таблицей, как любая базовая таблица, хранимая в БД, но с помощью представлений можно ограничить или наоборот расширить видимость БД для конкретного пользователя. Поддержание представлений производится также на языковом уровне.

Наконец, авторизация доступа к объектам БД производится также на основе специального набора операторов SQL. Идея состоит в том, что для выполнения тех или иных операторов SQL пользователь должен обладать различными полномочиями, или, иначе говоря, правами. Пользователь, создавший таблицу БД, обладает полным набором прав для работы с этой таблицей. В число этих полномочий входит право на передачу всех или части полномочий другим пользователям, включая право на передачу полномочий. Права пользователей описываются в специальных таблицах-каталогах, контроль полномочий поддерживается на языковом уровне.

Базы данных в ходу уже довольно давно. Особенно популярны они стали благодаря системам управления, реализующим проверенную временем - реляционную модель данных. В этой статье попробуем выделить основные различия между разными системами управления базами данных (СУБД). Рассмотрим функциональные отличия и особенности, как они работают и какую СУБД лучше выбрать исходя из потребностей разработки.

Базы данных - это специально разработанное хранилище для различных типов данных. Каждая база данных, имеет определённую модель (реляционная, документно-ориентированная), которая обеспечивает удобный доступ к данным. Системы управления базами данных (СУБД) - специальные приложения (или библиотеки) для управления базами данных различных размеров и форм.

**Реляционная система управления базами данных (РСУБД).**

СУБД должна обеспечивать реляционную модель работы с данными. Сама модель подразумевает определенный тип связи между сущностями из разных таблиц. Чтобы хранить и работать с данными, такой тип СУБД должен иметь определенную структуру (таблицы). В таблицах каждый столбец может содержать данные разного типа. Каждая запись состоит из множества атрибутов (столбцов) и имеет уникальный ключ, хранящейся в той же таблице - все эти данные взаимосвязаны между собой, как описано в реляционной модели.

**Типы данных и отношений между ними.**

Отношения в базах данных можно рассматривать как математическое множество, содержащее в себе число атрибутов, которые суммарно представляют собой базу данных и информацию, хранящуюся в ней (фраза для тех, кто понимает, что такое математическое множество). При создании структуры таблицы каждое поле записи должно иметь заранее описанный тип (например: строка, целочисленное значение и т.д.). Все СУБД имеют в своем составе различные типы данных, которые не всегда взаимозаменяемы. При работе с СУБД всегда приходится сталкиваться с подобными ограничениями.

**Популярные и основные реляционные базы данных.**

В этой статье мы с вами рассмотрим три основных свобоно распространяемых СУБД.

SQLite - очень мощная встраиваемая система управления .

MySQL - самая популярная и распространённая СУБД.

PostgreSQL - наиболее продвинутая СУБД.

**SQLite.**

Легко встраиваемая в приложения база данных. Так как это система базируется на файлах, то она предоставляет довольно широкий набор инструментов для работы с ней, по сравнению с сетевыми СУБД. При работе с этой СУБД обращения происходят напрямую к файлам (в эти файлах хранятся данные), вместо портов и сокетов в сетевых СУБД. Именно поэтому SQLite очень быстрая, а также мощная благодаря технологиям обслуживающих библиотек.

## Типы данных SQLite.

NULL - значение NULL.

INTEGER - знаковое целочисленное значение, использует 1, 2, 3, 4, 6, или 8 байт в зависимости от порядка числа.

REAL - число с плавающей точкой, занимает 8 байт для хранения числа в формате IEEE.

TEXT - текстовая строка, при хранении используются кодировки UTF-8, UTF-16BE или UTF-16LE.

BLOB - тип данных BLOB, массив двоичных данных (предназначенный, в первую очередь, для хранения изображений, аудио и видео).

## Преимущества SQLite.

Файловая структура - вся база данных состоит из одного файла, поэтому её очень легко переносить на разные машины.

Используемые стандарты - хотя может показаться, что эта СУБД примитивная, но она использует SQL. Некоторые особенности опущенны (RIGHT OUTER JOIN или FOR EACH STATEMENT), но основные все-таки поддерживаются.

Отличная при разработке и тестировании - в процессе разработки приложений часто появляется необходимость масштабирования. SQLite предлагает всё что необходимо для этих целей, так как состоит всего из одного файла и библиотеки написанной на языке C.

## Недостатки SQLite.

Отсутствие системы пользователей - более крупные СУБД включают в свой состав системы управления правами доступа пользователей. Обычно применения этой функции не так критично, так как эта СУБД используется в небольших приложениях.

Отсутствие возможности увеличения производительности - опять, исходя из проектирования, довольно сложно выжать что-то более производительное из этой СУБД.

## Когда использовать SQLite.

Встроенные приложения - если вам важна возможность легкого переноса приложения и не важна масштабируемость. Например однопользовательские приложения, мобильные приложения или приложения элементарной аналитики.

Прямой доступ к диску - при необходимости напрямую обращаться к диску вы можете выиграть при переходе на эту СУБД в функционале и простоте использования SQL языка.

Тестирование - использование дополнительных процессов при тестировании функционала, очень замедляет приложение.

## Когда отказаться от SQLite.

Многопользовательские приложения - если вам необходимо обеспечить доступ к данным для нескольких пользователей, да и к тому же различать их по правам доступа, то, наверное, полноценная СУБД (например: MySQL) будет более логичным выбором.

Запись больших объемов данных - одно из ограничений SQLite это операции записи. Разрешен только один процесс записи в промежуток времени, что сильно ограничивает производительность.

**MySQL.**

MySQL - это самая распространенная полноценная серверная СУБД. MySQL очень функциональная, свободно распространяемая СУБД, которая успешно работает с различными сайтами и веб приложениями. Обучиться использованию этой СУБД довольно просто, так как на просторах интернета вы легко найдете большее количество информации.

Несмотря на то, что в ней не реализован весь SQL функционал, MySQL предлагает довольно много инструментов для разработки приложений. Так как это серверная СУБД, приложения для доступа к данным, в отличии от SQLite работают со службами MySQL.

## Типы данных MySQL.

TINYINT - очень малые целочисленные значения.

SMALLINT - малые целочисленные значения.

MEDIUMINT - средние целочисленные значения.

INT или INTEGER - стандартные целочисленные значения.

BIGINT - большие целочисленные значения.

FLOAT - маленькие значения с плавающей точкой (точность до одного значения после точки). Всегда знаковые значения.

DOUBLE, BOUBLE PRECISION, REAL - Стандартные значения с плавающей точкой. Всегда знаковые.

DECIMAL, NUMERIC - распакованное значение с плавающей точкой, всегда знаковое.

DATE – дата.

DATETIME - дата и время в одном значении.

TIMESTAMP - временная отметка timestamp.

TIME – время.

YEAR - год, 2 или 4 числа (4 - по-умолчанию).

CHAR - строковое значение фиксированной длины, справа всегда добавляются пробелы до указанной длины при сортировке.

VARCHAR - строковое значение переменной длины.

TINYBLOB, TINYTEXT - значение типа BLOB или TEXT, 255 (2^8 - 1) символов - максимальныя длина

BLOB, TEXT - значение типа BLOB или TEXT, 65535 (2^16 - 1) символов - максимальныя длина.

MEDIUMBLOB, MEDIUMTEXT - значение типа BLOB или TEXT, 16777215 (2^24 - 1) символов - максимальныя длина.

LONGBLOB, LONGTEXT - значение типа BLOB или TEXT, 4294967296 (2^32 - 1) символов - максимальныя длина.

ENUM – перечисление.

SET – множество.

## Преимущества MySQL.

Простота в работе - установить MySQL довольно просто. Дополнительные приложения, например GUI, позволяет довольно легко работать с БД.

Богатый функционал - MySQL поддерживает большинство функционала SQL.

Безопасность - большое количество функций обеспечивающих безопасность, которые поддерживается по умолчанию.

Масштабируемость - MySQL легко работает с большими объемами данных и легко масштабируется.

Скорость - упрощение некоторых стандартов позволяет MySQL значительно увеличить производительность.

## Недостатки MySQL.

Известные ограничения - по задумке в MySQL заложены некоторые ограничения функционала, которые иногда необходимы в особо требовательных приложениях.

Проблемы с надежностью - из-за некоторых способов обработки данных MySQL (связи, транзакции, аудиты) иногда уступает другим СУБД по надежности.

Медленная разработка - Хотя MySQL технически открытое ПО,  существуют жалобы на процесс разработки.  Стоит заметить, что существуют другие довольно успешные СУБД созданные на базе MySQL, например MariaDB.

## Когда следует использовать MySQL.

Распределённые операции - если функционала SQLite не хватает, то стоит рассмотреть MySQL.  Так как эта СУБД сочетает в себе продвинутый функционал и свободный доступ к исходному коду.

Высокий уровень безопасности - система безопасности MySQL включает в себе простые и в то же время достойные способы защиты доступа к данным.

Веб сайты и веб приложения - большинство сайтов и онлайн приложений спокойно работают с MySQL несмотря на некоторые ограничения. Будучи легкой в настройке и масштабируемой системой - MySQL проверена временем.

Индивидуальные решения - если вы работаете с каким либо специфическим проектом, MySQL легко сможет вам помочь благодаря широким возможностям в настройке и функционалом.

## Когда лучше отказаться от MySQL.

Соответствие стандартам -Так как MySQL не ставит для себя целью - полностью соответствввать стандртам SQL, то эта СУБД не полностью поддерживает SQL.. Если в будущем вы планируете перейти на подобную систему, то MySQL - не лучший выбор.

Многопоточность - хотя некоторые движки БД довольно легко выполняют параллельное чтение, параллельные операции чтения-записи могут создать проблемы.

Недостаток функционала - некоторые движки MySQL, например, не поддерживают полнотекстовый поиск.

**PostgreSQL.**

PostgreSQL является самым профессиональным из всех трех рассмотренных нами СУБД. Она свободно распространяемая и максимально соответствует стандартам SQL. PostgreSQL или Postgres стараются полностью применять ANSI/ISO SQL стандарты своевременно с выходом новых версий. От других СУБД PostgreSQL отличается поддержкой востребованного объектно-ориентированного и/или реляционного подхода к базам данных. Например, полная поддержка надежных транзакций, т.е. атомарность, последовательность, изоляционность, прочность (Atomicity, Consistency, Isolation, Durability (ACID). Благодаря мощным технологиям Postgre очень производительна. Параллельность достигнута не за счет блокировки операций чтения, а благодаря реализации управления многовариантным параллелизмом (MVCC), что также обеспечивает соответствие ACID. PostgreSQL очень легко расширять своими процедурами, которые называются хранимые процедуры. Эти функции упрощают использование постоянно повторяемых операций. Хотя PostgreSQL и не может похвастаться большой популярностью в отличии от MySQL, существует довольно большое число приложений облегчающих работу с PostgreSQL, несмотря на всю мощность функционала. Сейчас довольно легко установить эту СУБД используя стандартные менеджеры пакетов операционных систем.

## Типы данных PostgreSQL.

bigint - знаковое 8-ми битное целочисленное значение.

bigserial - автоматически инкрементируемое 8-ми битное целочисленное значение.

bit[(n)] - строка постоянной длины.

bit varying [(n)] - строка переменной длины.

boolean - булево значение (true/false).

box - прямоугольник на плоскости.

bytea - бинарные данные (массив байтов).

character varying [(n)] - строковое значение переменной длины.

character [(n)] - строковое значение постоянной длины.

cidr - IPv4/IPv6 сетевой адрес.

circle - круг на плоскости.

date - календарная дата (год, месяц, день).

double precision - число с плавающей точкой двойной точности (8 байт).

inet - IPv4/IPv6 адрес хоста.

integer - знаковое 4-ех байтовое целочисленное значение.

interval [fields][(p)] - отрезок времени.

line - бесконечная прямая на плоскости.

lseg - отрезок на плоскости.

macaddr - MAC адрес.

mony - валютное значение.

numeric [(p, s)] - точное численное значение с выбранной точностью.

path - геометрическая кривая на плоскости.

point - геометрическая точка на плоскости.

polygony - многоугольник на плоскости.

real - число с плавающей точкой одинарной точности (4 байта).

smallint - знаковое целочисленное значение (4 байта).

serial - автоматические инкрементируемое целочисленное значение (4 байта).

text - строковое значение переменной длины.

time [(p)] [without time zone] - время суток (без часового пояса).

time [(p)] with time zone - время суток (включая часовой пояс).

timestamp [(p)] [without time zone] - дата и время (без часового пояса).

timestamp [(p)] with time zone - дата и время (с часовым поясом).

tsquery - текстовый поисковый запрос.

tsvector - документ текстового поиска.

txid_snapshot - пользовательский снимок транзакции с ID.

uuid - универсальный уникальный идентификатор.

xml - XML данные.

## Достоинства PostgreSQL.

Открытое ПО соответствующее стандарту SQL - PostgreSQL - бесплатное ПО с открытым исходным кодом.  Эта СУБД является очень мощной системой.

Большое сообщество - существует довольно большое сообщество в котором вы запросто найдёте ответы на свои вопросы.

Большое количество дополнений - несмотря на огромное количество встроенных функций, существует очень много дополнений, позволяющих разрабатывать данные для этой СУБД и управлять ими.

Расширения - существует возможность расширения функционала за счет сохранения своих процедур.

Объектность - PostrgreSQL это не только реляционная СУБД, но также и объектно-ориентированная с поддержкой наследования и много другого.

## Недостатки PostgreSQL.

Производительность - при простых операциях чтения PostgreSQL может значительно замедлить сервер и быть медленнее своих конкурентов, таких как MySQL.

Популярность - по своей природе, популярностью эта СУБД похвастаться не может, хотя и присутствует довольно большое сообщество.

Хостинг - в силу выше перечисленных факторов иногда довольно сложно найти хостинг с поддержкой этой СУБД.

## Когда использовать PostgreSQL.

Целостность данных - когда надежность и целостность данных - ваши требования, PostgreSQL будет, пожалуй, лучшим выбором.

Сложные пользовательские процедуры - если вам необходимо использовать пользовательские процедуры, то PostgreSQL имеет встроенную поддержку для них.

Интеграция - если в будущем вы планируете переход на платные СУБД, например Oracle, то сделать это с PostgreSQL будет довольно просто по сравнению с другими бесплатными СУБД.

Сложная структура данных - по сравнению с другими открытими СУБД PostgreSQL предоставляет больше возможностей для создания сложных структур данных без необходимости жертовать какими либо аспектами.

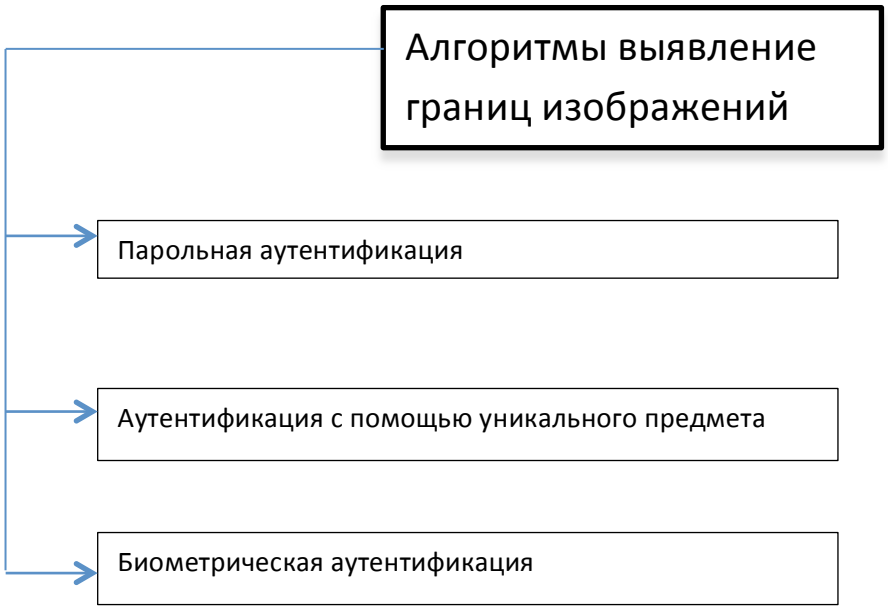
Когда не следует использовать PostgreSQL.

Скорость - если быстрое чтение для вас единственный фактор, то стоит присмотреться к другим СУБД.

Простая настройка - если вам не нужна целостность данных, соответствие ACID или сложные структуры данных, то настройка PostgreSQL может изрядно потрепать вам нервы.

Репликация - если вы не готовы потратить время и энергию на то, что мог бы с легкостью сделать MySQL, то наверное проще было бы на нем и остаться.

### 1.1.3 Классификация алгоритмов аутентификации




**Парольная аутентификация**

В настоящее время парольная аутентификация является наиболее распространенной, прежде всего, благодаря своему единственному достоинству – простоте использования.
Однако, парольная аутентификация имеет множество недостатков:

1. В отличие от случайно формируемых криптографических ключей (которые, например, может содержать уникальный предмет, используемый для аутентификации), пароли пользователя бывает возможно подобрать из-за достаточно небрежного отношения большинства пользователей к формированию пароля. Часто встречаются случаи выбора пользователями легко предугадываемых паролей, например:
   - пароль эквивалентен идентификатору (имени) пользователя (или имени пользователя, записанному в обратном порядке, или легко формируется из имени пользователя и т.д.);
   - паролем является слово или фраза какого-либо языка; такие пароли могут быть подобраны за ограниченное время путем «словарной атаки» - перебора всех слов согласно словарю, содержащему все слова и общеупотребительные фразы используемого языка;
   - достаточно часто пользователи применяют короткие пароли, которые взламываются методом «грубой силы», т.е. простым перебором всех возможных вариантов.
2. Существуют и свободно доступны различные утилиты подбора паролей, в том числе, специализированные для конкретных широко распространенных программных средств. Например, утилита подбора пароля для документа Microsoft Word 2000 (Word Password Recovery Key), предназначенная для восстановления доступа к документу, если его владелец забыл пароль. Несмотря на данное полезное назначение, ничто не мешает использовать эту и подобные ей утилиты для взлома чужих паролей
3. Пароль может быть получен путем применения насилия к его владельцу.
4. Пароль может быть подсмотрен или перехвачен при вводе.

**Аутентификация с помощью уникального предмета**

В большинстве случаев аутентификация с помощью уникального предмета обеспечивает более серьезную защиту, чем парольная аутентификация.
Предметы, используемые для аутентификации, можно условно разделить на следующие две группы:

1. «Пассивные» предметы, которые содержат аутентификационную информацию (например, некий случайно генерируемый пароль) и передают ее в модуль аутентификации по требованию. При этом, аутентификационная информация может храниться в предмете как в открытом (примеры: магнитные карты, смарт-карты с открытой памятью, электронные таблетки Touch Memory), так и в защищенном виде (смарт-карты с защищенной памятью, USB-токены). В последнем случае требуется ввод PIN-кода для доступа к хранящимся данным, что автоматически превращает предмет в средство двухфакторной аутентификации.
2. «Активные» предметы, которые обладают достаточными вычислительными ресурсами и способны активно участвовать в процессе аутентификации (примеры: микропроцессорные смарт-карты и USB-токены). Эта возможность особенно интересна при удаленной аутентификации пользователя, поскольку на основе таких предметов можно обеспечить *строгую* аутентификацию. Под этим термином скрывается такой вид аутентификации, при котором секретная информация, позволяющая проверить подлинность пользователя, не передается в открытом виде.

Аутентификация с помощью уникальных предметов обладает и рядом недостатков:

1. Предмет может быть похищен или отнят у пользователя.
2. В большинстве случаев требуется специальное оборудование для работы с предметами.
3. Теоретически возможно изготовление копии или эмулятора предмета.

**Биометрическая аутентификация**

Биометрическая аутентификация основана на уникальности ряда характеристик человека. Наиболее часто для аутентификации используются следующие характеристики:

1. Отпечатки пальцев.
2. Узор радужной оболочки глаза и структура сетчатки глаза.
3. Черты лица.

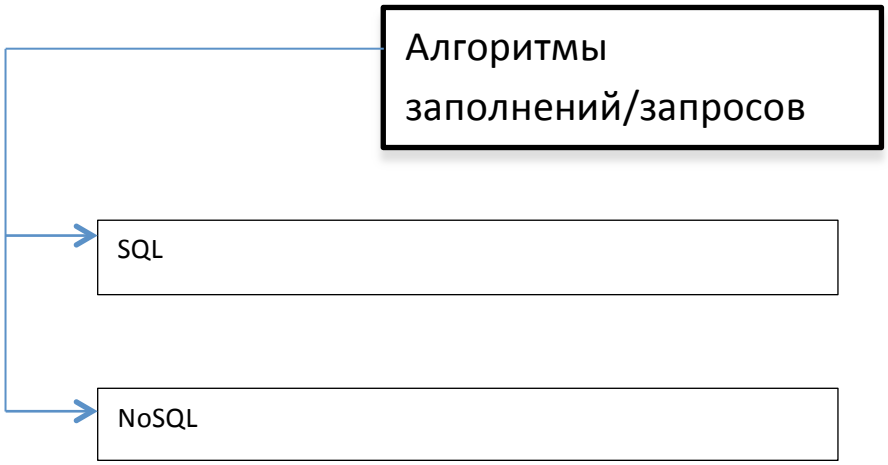
4. Форма кисти руки.
5. Параметры голоса.
6. Схема кровеносных сосудов лица.
7. Форма и способ подписи.

В процессе биометрической аутентификации эталонный и предъявленный пользователем образцы сравнивают с некоторой погрешностью, которая определяется и устанавливается заранее. Погрешность подбирается для установления оптимального соотношения двух основных характеристик используемого средства биометрической аутентификации:

1. FAR (False Accept Rate) – коэффициент ложного принятия (т.е. некто успешно прошел аутентификацию под именем легального пользователя).
2. FRR (False Reject Rate) – коэффициент ложного отказа (т.е. легальный пользователь системы не прошел аутентификацию).

Обе величины измеряются в процентах и должны быть минимальны. Следует отметить, что величины являются обратнозависимыми, поэтому аутентифицирующий модуль при использовании биометрической аутентификации настраивается индивидуально – в зависимости от используемой биометрической характеристики и требований к качеству защиты ищется некая «золотая середина» между данными коэффициентами. Серьезное средство биометрической аутентификации должно позволять настроить коэффициент FAR до величин порядка 0,01 – 0,001 % при коэффициенте FRR до 3 – 5%.
В зависимости от используемой биометрической характеристики, средства биометрической аутентификации имеют различные достоинства и недостатки. Например, использование отпечатков пальцев наиболее привычно и удобно для пользователей, но, теоретически, возможно создание «искусственного пальца», успешно проходящего аутентификацию.
Общий же недостаток биометрической аутентификации – необходимость в оборудовании для считывания биометрических характеристик, которое может быть достаточно дорогостоящим.

## 1.1.4 Классификация алгоритмов заполнений/запросов




### SQL

SQL или Structured Query Language (язык структурированных запросов) - язык программирования, предназначенный для управления данными в системах управления реляционными базами данных.

Рассмотрим ключевые моменты обращения к данным карт из SQL-запросов:

• Карта делиться на отдельные слои. В запросах SQL запрашиваются данные из одного или нескольких слоев карты.
• У каждого пространственного объекта слоя есть уникальный для этого слоя идентификатор *Sys*.
• Все поля данных БД слоя привязываются к пространственным объектам слоя по номерам *Sys*, причем названия таблиц БД слоя в запросах не учитываются.

В системе Zulu в результате выполнения запроса SQL может выводиться таблица с выборкой данных из слоев текущей карты и результатами вычисления выражений.

В данном руководстве приняты следующие соглашения по отображению элементов SQL:

• **SELECT** - ключевые слова языка.
• Название поля - названия полей баз данных слоев и названия слоев карт Zulu.
• Пример кода - примеры запросов SQL в тексте.
• <*Описание элементов*> - подстановочные поля, используемые в данном документе для удобства описания и к синтаксису SQL не относящиеся. На месте подстановочного поля в SQL запросе подставляются элементы описанные отдельно, либо описанные в самом подстановочном поле.
• [Элементы в квадратных скобках] - необязательные элементы выражения.
• A|B - вертикальной чертой отделяются возможные варианты ключевых слов или значений.

При составлении выражений SQL следует руководствоваться следующими правилами написания отдельных его элементов:

• Регистр символов ключевых слов SQL, названий полей и слоев не учитываются. Ключевое слово введенное прописными буквами, строчными и в смешанном регистре воспринимаются как одно ключевое слово. Например, ключевое слово **SELECT** может быть записано как **Select**, или **select**, а поле perimeter может быть записано так же, как и Perimeter, либо PERIMETER.
Тем не менее, для удобства восприятия, в данном документе ключевые слова языка SQL будут записываться прописными буквами.
• Длинная команда, состоящая из нескольких ключевых слов, может разбиваться на несколько строк.
• Команды запроса разделяются знаком «;» После последней команды знак «;»ставить необязательно.
• Названия полей и слоев в запросах, содержащие пробелы, должны заключаться в квадратные скобки, например название поля Номер источника должно быть записано как [Номер источника].
• Все строковые значения в запросах заключаются в одинарные или двойные кавычки. Если требуется использовать данные символы в самой строке, следует их дублировать, либо использовать кавычки отличающиеся от тех, что окружают данную строку (например, допускается использовать одинарные кавычки в строке окруженной двойными кавычками и двойные кавычки в строке окруженной одинарными кавычками).
Примеры задания строковых значений:
○ «Текстовая строка» - строковое значение в двойных кавычках;
○ 'Текстовая строка' - строковое значение в одинарных кавычках;
○ «Строка с текстом в кавычках - ""дублированных "" и 'нет'»- Использование кавычек в текстовой строке.
• При необходимости задания в запросе поля данных с указанием слоя, которому он принадлежит, название поля задается в формате:<*Название слоя*>.<*Название поля*>. Например, для задания поля Номер источника слоя Пример тепловой сети в запросе следует ввести строку [Пример тепловой сети].[Номер источника].
• В строках запроса значения могут не только вводиться непосредственно и получаться из полей БД, но вычисляться с помощью строковых и численных выражений. Такие выражения могут включать в себя различные константы, названия полей данных, функции и подзапросы, объединенные арифметическими или побитовыми операторами и скобками;

• В текст запроса могут добавляться комментарии, не влияющие на ход выполнения запроса. Комментарии могут использоваться для пояснения деталей запроса, либо для временного исключения отдельных команд из запроса.

Предусмотрены два вида комментариев:

- Комментарий строки. Такой комментарий начинается с сочетания символов «--», комментарием считаются все символы до конца строки после знаков начала комментария;
- Комментарий фрагмента. Такие комментарии начинаются сочетанием символов «/*», а заканчивается сочетанием «*/». Все символы между начальными и конечными символами считаются комментарием. Такие комментарии могут занимать несколько строк, либо часть строки, но не могут разрывать собой ключевые слова, названия функций.

### Выборка данных

Основным действием выполняемым с помощью SQL запросов в системе Zulu является выборка данных для их вывода в виде таблицы в области результатов запроса. Выборка данных производится с помощью ключевого слова **SELECT**, после которого задаются параметры выборки.

Строка команды выборки состоит из трех основных частей, в которых задается какие поля таблиц выводятся в итоговую таблицу, из каких слоев берутся данные и по каким условиям отбираются данные для итоговой таблицы.

При запросе данных из одного слоя карты, для использования в команде выборки полей данных достаточно указывать только их названия. Если же в выборке используются поля из нескольких слоев, то названия полей требуется указывать в формате *<Название слоя>.<Название поля>*.

Типовой запрос имеет следующий вид:

**SELECT** *<выводимые колонки>* [**FROM** *<список слоев>*] [**WHERE** *<условия отбора>* ]

где:

**SELECT** *<выводимые колонки>*

Часть команды выборки в которой задаются колонки выводимой таблицы данных. В области *<выводимые колонки>* через запятую перечисляются названия полей, значения которых выводятся в колонках таблицы, либо выражения, результаты расчета которых выводятся в таблице.

Для вывода в таблице значений всех полей данных из указанных в запросе слоев, задайте вместо списка полей символ « * ». В таком случае, значения полей в таблице будут выводиться в том же порядке, в котором они заданы в БД.

Для того чтобы в итоговой таблице выводились только отличающиеся друг от друга записи, задайте списком полей ключевое слово **DISTINCT**.

**FROM** *<список слоев>*

Часть строки выборки в которой через запятую перечисляются слои карты из которых запрашиваются данные.

Если все поля в запросе указаны в формате *<Название слоя>.<Название поля>*, часть запроса с ключевым словом **FROM** может быть опущена.

Если данные запрашиваются из более чем одного слоя карты, в итоговой таблице будет выведено декартово пересечение записей запрошенных слоев. Например, в случае запроса полей из двух слоев в итоговой таблице будет набор записей со всеми возможными комбинациями полей из записей первого

и второго слоя, т.е., например при запросе поля A из слоя содержащего 2 записи и запросе поля B из слоя также содержащего две записи, в итоговой таблице будет четыре записи со следующими данными: A1+B1, A1+B2, A2+B1, A2+B2.

**WHERE** *<условия отбора>*

Часть, в которой задаются условия, в соответствии с которыми отбираются записи данных в таблицу результатов.

В качестве условий могут использоваться операции сравнения, проверки равенства, вхождения значений полей в заданный диапазон, проверки относительного расположения элементов и т.д. Подробно синтаксис условий будет рассмотрен далее.

Если в таблице результатов требуется вывести все записи для указанных полей, эта часть запроса может быть опущена.

Также в команде выборки могут использоваться различные дополнительные команды, рассматриваемые в последующих подразделах.

**Примеры выборок**

**Простейшая выборка**

SELECT * FROM Кварталы

## NoSQL

### Основные черты

Традиционные СУБД ориентируются на требования ACID к транзакционной системе: атомарность, согласованность, изолированность (англ. *isolation*), надёжность (англ. *durability*), тогда как в NoSQL вместо ACID может рассматриваться набор свойств BASE:

- базовая доступность (англ. *basic availability*) — каждый запрос гарантированно завершается (успешно или безуспешно).
- гибкое состояние (англ. *soft state*) — состояние системы может изменяться со временем, даже без ввода новых данных, для достижения согласования данных.
- согласованность в конечном счёте (англ. *eventual consistency*) — данные могут быть некоторое время рассогласованы, но приходят к согласованию через некоторое время.

Термин «BASE» был предложен Эриком Брюером, автором теоремы CAP, согласно которой в распределённых вычислениях можно обеспечить только два из трёх свойств: согласованность данных, доступность или устойчивость к разделению.

Разумеется, системы на основе BASE не могут использоваться в любых приложениях: для функционирования биржевых и банковских систем использование транзакций является необходимостью. В то же время, свойства ACID, какими бы желанными они ни были, практически невозможно обеспечить в системах с многомиллионной веб-аудиторией, вроде amazon.com. Таким образом, проектировщики NoSQL-систем жертвуют согласованностью данных ради достижения двух других свойств из теоремы CAP. Некоторые СУБД, например, Riak, позволяют настраивать требуемые

характеристики доступности-согласованности даже для отдельных запросов путём задания количества узлов, необходимых для подтверждения успеха транзакции.

Решения NoSQL отличаются не только проектированием с учётом масштабирования. Другими характерными чертами NoSQL-решений являются:

- Применение различных типов хранилищ.
- Возможность разработки базы данных без задания схемы.
- Использование многопроцессорности.
- Линейная масштабируемость (добавление процессоров увеличивает производительность).
- Инновационность: «не только SQL» открывает много возможностей для хранения и обработки данных.
- Сокращение времени разработки
- Скорость: даже при небольшом количестве данных конечные пользователи могут оценить снижение времени отклика системы с сотен миллисекунд до миллисекунд

### 1.1.5 Язык программирования C++

C++ — компилируемый статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование, обеспечивает модульность, раздельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов, виртуальные функции. Стандартная библиотека включает, в том числе, общеупотребительные контейнеры и алгоритмы. C++ сочетает свойства как высокоуровневых, так инизкоуровневых языков. В сравнении с его предшественником — языком C, — наибольшее внимание уделено поддержкеобъектно-ориентированного и обобщённого программирования.

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр). Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ. Например, наплатформе x86 это GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder идругие. C++ оказал огромное влияние на другие языки программирования, в первую очередь на Java и C#.

Синтаксис C++ унаследован от языка C. Одним из принципов разработки было сохранение совместимости с C. Тем не менее, C++ не является в строгом смысле надмножеством C; множество программ, которые могут одинаково успешно транслироваться как компиляторами C, так и компиляторами C++, довольно велико, но не включает все возможные программы на C.

**История создания**

Язык возник в начале 1980-х годов, когда сотрудник фирмы Bell Labs Бьёрн Страуструп придумал ряд усовершенствований к языку C под собственные нужды. Когда в конце 1970-х годов Страуструп

начал работать в Bell Labs над задачами теории очередей (в приложении к моделированию телефонных вызовов), он обнаружил, что попытки применения, существующих в то время языков моделирования оказываются неэффективными, а применение высокоэффективных машинных языков слишком сложно из-за их ограниченной выразительности. Так, язык Симула имеет такие возможности, которые были бы очень полезны для разработки большого программного обеспечения, но работает слишком медленно, а язык BCPLдостаточно быстр, но слишком близок к языкам низкого уровня и не подходит для разработки большого программного обеспечения.

Вспомнив опыт своей диссертации, Страуструп решил дополнить язык C (преемник BCPL) возможностями, имеющимися в языке Симула. Язык C, будучи базовым языком системы UNIX, на которой работали компьютеры Bell, является быстрым, многофункциональным и переносимым. Страуструп добавил к нему возможность работы с классами и объектами. В результате практические задачи моделирования оказались доступными для решения как с точки зрения времени разработки (благодаря использованию Симула-подобных классов), так и с точки зрения времени вычислений (благодаря быстродействию C). В первую очередь в C были добавлены классы (с инкапсуляцией), наследование классов, строгая проверка типов, inline-функции и аргументы по умолчанию. Ранние версии языка, первоначально именовавшегося «C with classes» («Си с классами»), стали доступны с 1980 года.

Разрабатывая C с классами, Страуструп написал программу cfront — транслятор, перерабатывающий исходный код C с классами в исходный код простого C. Это позволило работать над новым языком и использовать его на практике, применяя уже имеющуюся в UNIX инфраструктуру для разработки на C. Новый язык, неожиданно для автора, приобрёл большую популярность среди коллег и вскоре Страуструп уже не мог лично поддерживать его, отвечая на тысячи вопросов.

При создании C++ Бьёрн Страуструп хотел получить универсальный язык со статическими типами данных, эффективностью и переносимостью языка C.

Непосредственно и всесторонне поддерживать множество стилей программирования, в том числе процедурное программирование, абстракцию данных, объектно-ориентированное программирование и обобщённое программирование.

Дать программисту свободу выбора, даже если это даст ему возможность выбирать неправильно.

Максимально сохранить совместимость с C, тем самым делая возможным лёгкий переход от программирования на C.

Избежать разночтений между C и C++: любая конструкция, допустимая в обоих языках, должна в каждом из них обозначать одно и то же и приводить к одному и тому же поведению программы.

Избегать особенностей, которые зависят от платформы или не являются универсальными.

«Не платить за то, что не используется» — никакое языковое средство не должно приводить к снижению производительности программ, не использующих его.

Не требовать слишком усложнённой среды программирования.

Выбор именно C в качестве базы для создания нового языка программирования объясняется тем, что язык C:

- является многоцелевым, лаконичным и относительно низкоуровневым языком;
- подходит для решения большинства системных задач;
- исполняется везде и на всём;
- стыкуется со средой программирования UNIX.

Несмотря на ряд известных недостатков языка C, Страуструп пошёл на его использование в качестве основы, так как «в C есть свои проблемы, но их имел бы и разработанный с нуля язык, а проблемы C нам известны». Кроме того, это позволило быстро получить прототип компилятора (cfront), который лишь выполнял трансляцию добавленных синтаксических элементов в оригинальный язык C.

По мере разработки C++ в него были включены другие средства, которые перекрывали возможности конструкций C, в связи с чем неоднократно поднимался вопрос об отказе от совместимости языков путём удаления устаревших конструкций. Тем не менее, совместимость была сохранена из следующих соображений:

- сохранение действующего кода, написанного изначально на C и прямо перенесённого в C++;
- исключение необходимости переучивания программистов, ранее изучавших C (им требуется только изучить новые средства C++);
- исключение путаницы между языками при их совместном использовании («если два языка используются совместно, их различия должны быть или минимальными, или настолько большими, чтобы языки было невозможно перепутать»).

К 1983 году в язык были добавлены новые возможности, такие как виртуальные функции, перегрузка функций и операторов, ссылки, константы, пользовательский контроль над управлением свободной памятью, улучшенная проверка типов и новый стиль комментариев. Получившийся язык уже перестал быть просто дополненной версией классического C и был переименован из C с классами в «C++». Его первый коммерческий выпуск состоялся в октябре 1985 года.

Имя языка, получившееся в итоге, происходит от оператора унарного постфиксного инкремента C ++ (увеличение значения переменной на единицу).

До начала официальной стандартизации язык развивался в основном силами Страуструпа в ответ на запросы программистского сообщества. Функцию стандартных описаний языка выполняли написанные Страуструпом печатные работы по C++ (описание языка, справочное руководство и так далее).

**История стандартов**

В 1985 году вышло первое издание «Языка программирования C++», обеспечивающее первое описание этого языка, что было чрезвычайно важно из-за отсутствия официального стандарта. В 1989 году состоялся выход C++ версии 2.0. Его новые возможности включали множественное наследование, абстрактные классы, статические функции-члены, функции-константы и защищённые члены. В 1990 году вышло «Комментированное справочное руководство по C++», положенное впоследствии в основу стандарта. Последние обновления включали шаблоны, исключения, пространства имён, новые способы приведения типов и булевский тип.

Стандартная библиотека C++ также развивалась вместе с ним. Первым добавлением к стандартной библиотеке C++ стали потоки ввода-вывода, обеспечивающие средства для замены традиционных функций C printf и scanf. Позднее самым значительным развитием стандартной библиотеки стало включение в неё Стандартной библиотеки шаблонов.

В 1998 году был опубликован стандарт языка ISO/IEC 14882:1998 (известный как C++98),[8] разработанный комитетом по стандартизации C++ (ISO/IEC JTC1/SC22/WG21 working group). Стандарт C++ не описывает способы именования объектов, некоторые детали обработки исключений и другие возможности, связанные с деталями реализации, что делает несовместимым

объектный код, созданный различными компиляторами. Однако для этого третьими лицами создано множество стандартов для конкретных архитектур и операционных систем.

В 2003 году был опубликован стандарт языка ISO/IEC 14882:2003, где были исправлены выявленные ошибки и недочёты предыдущей версии стандарта.

В 2005 году был выпущен отчёт Library Technical Report 1 (кратко называемый TR1). Не являясь официально частью стандарта, отчёт описывает расширения стандартной библиотеки, которые, как ожидалось авторами, должны быть включены в следующую версию языка C++. Степень поддержки TR1 улучшается почти во всех поддерживаемых компиляторах языка C++.

С 2009 года велась работа по обновлению предыдущего стандарта, предварительной версией нового стандарта сперва был C++09, а спустя год C++0x, сегодня — C++11, куда были включены дополнения в ядро языка и расширение стандартной библиотеки, в том числе большую часть TR1.

C++ продолжает развиваться, чтобы отвечать современным требованиям. Одна из групп, разрабатывающих язык C++ и направляющих комитету по стандартизации C++ предложения по его улучшению — это Boost, которая занимается, в том числе, совершенствованием возможностей языка путём добавления в него особенностейметапрограммирования.

Никто не обладает правами на язык C++, он является свободным. Однако сам документ стандарта языка (за исключением черновиков) не доступен бесплатно.

## Обзор языка

Стандарт C++ на 2003 год состоит из двух основных частей: описание ядра языка и описание стандартной библиотеки.

Кроме того, существует огромное количество библиотек C++, не входящих в стандарт. В программах на C++ можно использовать многие библиотеки C.

Стандартизация определила язык программирования C++, однако за этим названием могут скрываться также неполные, ограниченные, достандартные варианты языка. Первое время язык развивался вне формальных рамок, спонтанно, по мере вставaвших перед ним задач. Развитию языка сопутствовало развитие кросс-компиляторacfront. Новшества в языке отражались в изменении номера версии кросс-компилятора. Эти номера версий кросс-компилятора распространялись и на сам язык, но применительно к настоящему времени речь о версиях языка C++ не ведут.

## Необъектно-ориентированные возможности

В этом разделе описываются возможности, непосредственно не связанные с объектно-ориентированным программированием (ООП), но многие из них, однако, особенно важны в сочетании с ООП.

### Комментарии

C++ поддерживает как комментарии в стиле C:

```
/*
это комментарий, который может состоять
из нескольких строчек
*/
```

так и однострочные:

```
// вся оставшаяся часть строки является комментарием
```

где // обозначает начало комментария, а ближайший последующий символ новой строки, который не предварён символом \ считается окончанием комментария.

**Типы**

В C++ доступны следующие встроенные типы:

• Символьные: char, wchar_t (char16_t и char32_t, в стандарте C++11).

• Целочисленныезнаковые: signed char, short int, int, long int (и long long int, встандарте C++11).

• Целочисленныебеззнаковые: unsigned char, unsigned short int, unsigned int, unsigned long int (и unsigned long long int, встандарте C++11).

• Сплавающейточкой: float, double, long double.

• Логический: bool, имеющий значения true и false.

• Операции сравнения возвращают тип bool. Выражения в скобках после if, while приводятся к типу bool.

Функции могут принимать аргументы по ссылке. Например, функция void f(int &x) {x=3;} присваивает своему аргументу значение 3. Функции также могут возвращать результат по ссылке, и ссылки могут быть вне всякой связи с функциями.

Например, {double &b=a[3]; b=sin(b);} эквивалентно a[3]=sin(a[3]);.

При программировании ссылки в определённой степени сходны с указателями, со следующими особенностями: перед использованием ссылка должна быть инициализирована; ссылка пожизненно указывает на один и тот же адрес; в выражении ссылка обозначает непосредственно тот объект или ту функцию, на которую она указывает, обращение же к объекту или функции через указатель требует разыменование указателя. Существуют и другие отличия в использовании указателей и ссылок. Концептуально ссылка — другое имя переменной или функции, другое название одного и того же адреса, существует лишь только в тексте программы, заменяемое адресом при компиляции; а указатель — переменная, хранящая адрес, к которому обращаются.

Спецификатор inline позволяет объявлять inline-функции. Функция, определённая внутри тела класса, является inline по умолчанию. Изначально inline-функции задумывались как функции, являющиеся хорошими кандидатами на оптимизацию, при которой в местах обращения к функции компилятор вставит тело этой функции, а не код вызова. В действительности компилятор не обязан реализовывать подстановку тела для inline-функций, но может, исходя из заданных критериев оптимизации, выполнять подстановку тела для функций, которые не объявлены как inline. Пожалуй, наиболее значимой особенностью inline-функции является то, что она может многократно определяться в нескольких единицах трансляции (при этом inline-функция должна быть определена во всех единицах трансляции, где она используется), в то время как функция, не являющаяся inline, может определяться в программе не более одного раза.

*Пример:*

inline double Sqr(double x) {return x*x;}.

Описатель volatile используется в описании переменных и информирует компилятор, что значение данной переменной может быть изменено способом, который компилятор не в состоянии отследить. Для переменных, объявленных volatile, компилятор не должен применять средства оптимизации, изменяющие положение переменной в памяти (например, помещающие её в регистр) или полагающиеся на неизменность значения переменной в промежутке между двумя присваиваниями ей значения. В мультиядерной системе volatile помогает избегать барьеров памяти 2-го типа

Если описана структура, класс, объединение (union) или перечисление (enum), её имя является именем типа, *например:*

```
struct Time {
   int hh, mm, ss;
};
Time t1, t2;
```

Добавлены пространства имён (namespace). Например, еслинаписать

```
namespace Foo
{
  const int x=5;
  typedef int** T;
  void f(int y) {return y*x};
double g(T);
  ...
}
```

то вне фигурных скобок следует обращаться к T, x, f, g как Foo::T, Foo::x, Foo::f и Foo::g соответственно. Если в каком-то файле нужно обратиться к ним непосредственно, можно написать

```
using namespace Foo;
```

Илиже

```
using Foo::T;
```

Пространства имён нужны, чтобы не возникало коллизий между пакетами, имеющими совпадающие имена глобальных переменных, функций и типов. Специальным случаем является безымянное пространство имён

```
namespace
{
   ...
}
```

Все имена, описанные в нём, доступны в текущей единице трансляции и больше нигде.

Один или несколько последних аргументов функции могут задаваться по умолчанию. К примеру, если функция описана как void f(int x, int y=5, int z=10), вызовы f(1), f(1,5) и f(1,5,10) эквивалентны.

При описании функций отсутствие аргументов в скобках означает, в отличие от C, что аргументов нет, а не то, что они неизвестны. Если аргументы неизвестны, надо пользоваться многоточием, например, int printf(const char* fmt, ...).

Внутри структуры или класса можно описывать вложенные типы, как через typedef, так и через описание других классов, а также перечислений. Для доступа к таким типам вне класса, к имени типа добавляется имя структуры или класса и два двоеточия:

```
struct S
{
   typedef int** T;
T x;
};
S::T y;
```

Могут быть несколько функций с одним и тем же именем, но разными типами или количеством аргументов (перегрузка функций; при этом тип возвращаемого значения на перегрузку не влияет). Например, вполнеможнописать:

void Print(int x);

void Print(double x);

void Print(int x, int y);

Смысл некоторых операторов применительно к пользовательским типам можно определять через объявление соответствующих операторных функций. Кпримеру, так:

struct Date {int day, month, year;};

void operator ++(struct Date& date);

Операторные функции во многом схожи с обычными (неоператорными) функциями. За исключением операторов new, new[], delete и delete[], нельзя переопределять поведение операторов для встроенных типов (скажем, переопределять умножение значений типа int); нельзя создавать новые операторы, которых нет в C++ (скажем, **); нельзя менять количество операндов, предусмотренное для оператора, а также нельзя менять существующие приоритеты и ассоциативность операторов (скажем, в выражении a+b*c сначала будет выполняться умножение, а потом сложение, к каким бы типам ни принадлежали a, b и c). Можно переопределить операции [] (с одним параметром) и () (с любым числом параметров).

Добавлены шаблоны (template).

Например, template<class T> T Min(T x, T y) {return x<y?x:y;} определяет функцию Min для любых типов. Шаблоны могут задавать не только функции, но и типы. Например, template<class T> struct Array{int len; T* val;}; определяет массив значений любого типа, после чего мы можем писать Array<float> x;

В дополнение к функциям malloc и free введены операторные функции operator new, operator new[], operator delete и operator delete[], а также операторы new, new[], delete и delete[]. Если T — произвольный объектный тип, не являющийся типом массива, X — произвольный объектный тип и A — тип массива из некоторого количества n элементов, имеющих тип X, то

new T выделяет память (посредством вызова функции operator new), достаточную для размещения одного объекта типа T, возможно, инициализирует объект в этой памяти, и возвращает указатель типа T* (например, T* p = new T).

new X[n] и new A выделяют память (посредством вызова функции operator new[]), достаточную для размещения n объектов типа X, возможно, инициализируют каждый объект в этой памяти, и возвращают указатель типа X* (например, X* p = new X[n]).

delete p — разрушает объект (не являющийся массивом), на который ссылается указатель p, и освобождает область памяти (посредством вызова функцииoperator delete), ранее выделенную для него new-выражением.

delete [] p — разрушает каждый объект в массиве, на который ссылается указатель p, и освобождает область памяти (посредством вызова функции operator delete[]), ранее выделенную для этого массива new-выражением.

Операция delete проверяет, что её аргумент не является нулевым указателем, в противном случае она ничего не делает. Для инициализации объекта non-POD классового типа new-выражение вызывает конструктор; для уничтожения объекта классового типа delete-выражение вызывает деструктор.

**Объектно-ориентированные особенности языка**

C++ добавляет к C объектно-ориентированные возможности. Он вводит классы, которые обеспечивают три самых важных свойства ООП: инкапсуляцию, наследование иполиморфизм.

В стандарте C++ под классом (class) подразумевается пользовательский тип, объявленный с использованием одного из ключевых слов class, struct или union, под структурой (structure) подразумевается класс, определённый через ключевое слово struct, и под объединением (union) подразумевается класс, определённый через ключевое слово union.

## 1.1.6 Сравнение языков C++ и C

Выбор C в качестве базового языка для C++ объясняется следующими его достоинствами:

*1. Универсальность, краткость и относительно низкий уровень;*

*2. Адекватность большинству задач системного программирования;*

*3. Он работает в любой системе и на любой машине;*

*4. Полностью подходит для программной среды UNIX.*

В C существуют свои проблемы, но в языке, разрабатываемом "с нуля" они появились бы тоже, а проблемы C, по крайней мере, хорошо известны. Более важно то, что ориентация на C позволила использовать язык "C с классами" как полезный (хотя и не очень удобный) инструмент в течение первых месяцев раздумий о введении в C классов в стиле Симулы.

C++ стал использоваться шире, но по мере роста его возможностей, выходящих за пределы C, вновь и вновь возникала проблема совместимости. Ясно, что отказавшись от части наследства C, можно избежать некоторых проблем. Это не было сделано по следующим причинам:

(1) существуют миллионы строк программ на C, которые можно улучшить с помощью C++, но при условии, что полной переписи их на язык C++ не потребуется;

(2) существуют миллионы строк библиотечных функций и служебных программ на C, которые можно было бы использовать в C++ при условиях совместимости обоих языков на стадии связывания и их большого синтаксического сходства;

(3) существуют сотни тысяч программистов, знающих C; им достаточно овладеть только новыми средствами C++ и не надо изучать основ языка;

(4) поскольку C и C++ будут использоваться одними и теми же людьми на одних и тех же системах многие годы, различия между языками должны быть либо минимальными, либо максимальными, чтобы свести к минимуму количество ошибок и недоразумений. Описание C++ было переработано так, чтобы гарантировать, что любая допустимая в обоих языках конструкция означала в них одно и то же.

Язык C сам развивался в последние несколько лет, что отчасти было связано с разработкой C++. Стандарт ANSI для C содержит, например, синтаксис описания функций, позаимствованный из языка "C с классами". Происходит взаимное заимствование, например, тип указателя void* был придуман для ANSI C, а впервые реализован в C++. Как было обещано в первом издании этой книги, описание C++ было доработано, чтобы исключить неоправданные расхождения. Теперь C++ более совместим с языком C, чем это было вначале. В идеале C++ должен максимально приближаться к ANSI C, но не более. Стопроцентной совместимости никогда не было и не будет, поскольку это нарушит надежность типов и согласованность использования встроенных и пользовательских типов, а эти свойства всегда были одними из главных для C++.

Для изучения C++ не обязательно знать C. Программирование на C способствует усвоению приемов и даже трюков, которые при программировании на C++ становятся просто ненужными. Например, явное преобразование типа (приведение) , в C++ нужно гораздо реже, чем в C. Тем не менее, хорошие программы на языке C по сути являются программами на C++. Например, все программы из классического описания C являются программами на C++. В процессе изучения C++ будет полезен опыт работы с любым языком со статическими типами.

### 1.1.7 Сравнение Java и C++

Java и C++ часто сравниваются как языки, унаследовавшие синтаксис Си, несмотря на огромные различия на всех уровнях, от семантики до сферы применимости. Можно сказать, сравнение C++ с Java идёт вторым по частоте после сравнения C++ с Си.

**Целевая ниша**

Java позиционируется для весьма конкретного сектора промышленности: безопасный язык с низким порогом вхождения для разработки прикладных пользовательских приложений широкого рынка с высокими показателями портируемости— и с этой задачей справляется. C++ претендует на «универсальную применимость» во всех задачах для всех категорий программистов, но не удовлетворяет в полной мере требованиям ни одной из заявленных сфер применимости (см. раздел Критика).

**Исполнение программы**

Java имеет формальную семантику, ориентированную на интерпретацию, но код Java компилируется в промежуточный код, который непосредственно перед запускомпрограммы компилируется в машинный (иногда говорят об интерпретации байт-кода, но в данном случае это неверно — у современных наиболее распространённых сред исполнения Java оба этапа трансляции являются полностадийными, не ограничиваясь работой в рамках AST, с соответствующим сужением возможностей). C++ имеет естественную семантику, ориентированную на компиляцию, так что уже на аппаратной Java-машине был бы крайне неэффективен и ограничен по возможностям. Одно это определяет разницу в сферах применения языков: Java нецелесообразно использовать в низкоуровневом программировании; C++ — в разработке интернет-приложений. Механизм исполнения Java делает программы полностью портируемыми, по принципу «написано один раз — запускается везде (write once — run everywhere)», хотя это не было первостепенной целью разработчиков. Стандартное окружение и среда исполнения позволяют выполнять программы на Java на любой аппаратной платформе и в любой ОС без каких-либо изменений, при условии существования на данной ОС и платформе среды исполнения. Усилия по портированию программ минимальны, и могут быть сведены к нулю соблюдением определённых рекомендаций при разработке. Ценой портируемости в данном случае становятся определённые накладные расходы (например, размер среды исполнения Java превышает даже их размеры у всех функциональных языков).

**Парадигма программирования**

Java в значительно более высокой степени, чем C++, отвечает фундаментальному принципу ООП «всё — объект» (но не в абсолютной — методы классов самостоятельными объектами не являются, в отличие от CLOS или Python). Для объявления глобальных функций или переменных в Java их необходимо оборачивать в фиктивные классы и назначать свойство статичности[43]. Для задания главной функции даже самой простой программы на Java необходимо поместить её в класс[44]. Программировать на Java в функциональном стиле затруднено[источник не указан 126 дней], поскольку

на уровне синтаксиса языка нет поддержки основных элементов ФП (таких как функций высшего порядка) и нет средств макрорасширения языка.

**Объектная модель**

Как и в C++, объектная модель Java наследуется из Симулы (в Java — через промежуточную ступень — язык Modula-2), то есть фундаментально отличается от оной в потомках языка Smalltalk (Objective-C, Python, Ruby). Но есть серьёзные отличия и от C++. В Java все методы являются виртуальными. Есть синтаксический сахар для определения абстрактных классов: использование ключевого слова interface делает все методы класса чистыми виртуальными — такие классы называются в Java «интерфейсами». Множественное наследование допустимо только для них, но не для обычных классов, что улучшает дисциплину программирования — на этапе реализации нет возможности нарушить структуру проекта, построенную на этапе архитектурного проектирования (в C++ это делается легко).

**Операторы**

Безусловный переход в Java отсутствует. Тернарная условная операция в Java также отсутствует. Однако, большинство конструкций являются типичными для всех потомков Алгола: императивный порядок вычислений, присваивания, ветвления, циклы, инфиксные арифметические операции, аргументы в объявлениях и вызовах функций, и пр. В целом спецификация Java отвечает принципу наименьшего удивления и позволяет быстрый переход на Java с любого языка семейства Алгола. Однако, есть и исключения — например, Java, как и C++, позволяет пропускать break в ветви оператора switch.

**Синтаксис**

Основные конструкции Java и характерное оформление блоков кода наследованы из Си; большинство синтаксических отличий обусловлены разницей в семантике. Спецификаторы видимости компонентов класса в Java указываются на каждый компонент, а не группами, как в C++. В Java нет механизма ввода синтаксического сахара в программу — перегрузки операторов.

**Адресная арифметика**

C++ сохраняет возможность работы с низкоуровневыми указателями — это является причиной труднообнаруживаемых ошибок, но необходимо для низкоуровневого программирования. В Java адресной арифметики нет.

**Кодогенерация**

C++ не только сохраняет препроцессор Си, но и дополняет его Тьюринг-полным языком шаблонов, существенно расширяя возможности автоматического построения кода. В Java макроопределения времени компиляции отсутствуют.

**Интроспекция**

В C++ RTTI ограничена возможностью сравнивать типы объектов между собой и с буквальными значениями типов. В системе Java доступна более подробная информация о типах.

**Управление ресурсами**

C++ позволяет использовать принцип «захват ресурсов путём инициализации» (RAII), при котором ресурсы ассоциированы с объектом и автоматически освобождаются при разрушении объекта (например, std::vector и std::ifstream). Также возможен подход, когда программист, выделяя ресурсы (память под объекты, открытые файлы и т. п.), обязан явно позаботиться о своевременном их

освобождении. Java работает в среде со сборкой мусора, которая автоматически отслеживает прекращение использования объектов и освобождает занимаемую ими память, если в этом есть необходимость, в некоторый неопределённый момент времени. Ручное управление предпочтительнее в системном программировании, где требуется полный контроль над ресурсами, RAII и сборка мусора удобнее в прикладном программировании, поскольку в значительной степени освобождают программиста от необходимости отслеживать момент прекращения использования ресурсов. Сборщик мусора Java требует системных ресурсов, что снижает эффективность выполнения программ, лишает программы на Java детерминированности выполнения и способен следить только за памятью. Файлы, каналы, сокеты, объекты графического интерфейса программист на Java всегда освобождает явно.

**Окружение**

В состав среды исполнения Java уже входят библиотеки для графики, графического интерфейса, доступа к базам данных и для прочих типовых задач, которые определяют стандарт де-факто их использования. Состав базовой библиотеки C++ предоставляет много меньше возможностей, с другой стороны предоставляя больше свободы в выборе сторонних библиотек.

## 1.1.8 Среда разработки Qt

**Qt** (произносится «кьют») — кроссплатформенный инструментарий разработки ПО на языке программированияC++.

Есть также «привязки» ко многим другим языкам программирования:

- Python — PyQt,
- PySide;
- Ruby —QtRuby;
- Java — Qt Jambi;
- PHP — PHP-Qt

Позволяет запускать написанное с его помощью ПО в большинстве современных операционных систем путём простой компиляции программы для каждой ОС без изменения исходного кода. Включает в себя все основныеклассы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементовграфического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Qt является полностью объектно-ориентированным, легко расширяемым и поддерживающим технику компонентного программирования.

Существуют версии библиотеки для Microsoft Windows, систем класса UNIX с графической подсистемой X11,Android, iOS, Mac OS X, Microsoft Windows CE, QNX, встраиваемых Linux-систем и платформы S60. Идет портирование на Windows Phone и Windows RT. Также идёт портирование на Haiku и Tizen.

До недавнего времени библиотека Qt также распространялась ещё в одной версии: Qt/Embedded. Теперь эта платформа переименована в Qtopia Core и распространяется как отдельный продукт. Qtopia Core обеспечивает базовую функциональность для всей линейки платформ, предназначенных для разработки приложений для встраиваемых и мобильных устройств (КПК, смартфонов и т. п.).

Начиная с версии 4.5 Qt распространяется по 3 лицензиям (независимо от лицензии, исходный код Qt один и тот же):

▪ Qt Commercial — для разработки ПО с собственнической лицензией, допускающая модификацию самой Qt без раскрытия изменений;

▪ GNU GPL — для разработки ПО с открытыми исходниками, распространяемыми на условиях GNU GPL;

▪ GNU LGPL — для разработки ПО с собственнической лицензией, но без внесения изменений в Qt.

До версии 4.0.0 под свободной лицензией распространялись лишь Qt/Mac, Qt/X11, Qt/Embedded, но, начиная с 4.0.0 (выпущенной в конце июня 2005), Qt Software «освободили» и Qt/Windows. Следует отметить, что существовали сторонние свободные версии Qt/Windows < 4.0.0, сделанные на основе Qt/X11.

Со времени своего появления в 1996 году библиотека Qt легла в основу тысяч успешных проектов[14] во всём мире. Кроме того, Qt является фундаментом популярной рабочей среды KDE, входящей в состав многих дистрибутивовLinux.

Отличительная особенность Qt от других библиотек — использование Meta Object Compiler (MOC) — предварительной системы обработки исходного кода (в общем-то, Qt — это библиотека не для чистого C++, а для его особого наречия, с которого и «переводит» MOC для последующей компиляции любым стандартным C++ компилятором). MOC позволяет во много раз увеличить мощь библиотек, вводя такие понятия, как слоты исигналы. Кроме того, это позволяет сделать код более лаконичным. Утилита MOC ищет в заголовочных файлах наC++ описания классов, содержащие макрос Q_OBJECT, и создаёт дополнительный исходный файл на C++, содержащий метаобъектный код.

Qt позволяет создавать собственные плагины и размещать их непосредственно в панели визуального редактора. Также существует возможность расширения привычной функциональности виджетов, связанной с размещением их на экране, отображением, перерисовкой при изменении размеров окна.

Qt комплектуется визуальной средой разработки графического интерфейса «Qt Designer», позволяющей создавать диалоги и формы в режиме WYSIWYG. В поставке Qt есть «Qt Linguist» — графическая утилита, позволяющая упростить локализацию и перевод программы на многие языки; и «Qt Assistant» — справочная система Qt, упрощающая работу с документацией по библиотеке, а также позволяющая создавать кросс-платформенную справку для разрабатываемого на основе Qt ПО. Начиная с версии 4.5.0 в комплект Qt включена среда разработки «Qt Creator», которая включает в себя редактор кода, справку, графические средства «Qt Designer» и возможность отладки приложений. «Qt Creator» может использовать GCC или Microsoft VC++ в качестве компилятора и GDB в качестве отладчика. Для Windows версий библиотека комплектуется компилятором, заголовочными и объектными файлами MinGW.

Библиотека разделена на несколько модулей, для четвёртой версии библиотеки это:

▪ QtCore — классы ядра библиотеки, используемые другими модулями;

▪ QtGui — компоненты графического интерфейса;

▪ QtNetwork — набор классов для сетевого программирования. Поддержка различных высокоуровневых протоколов может меняться от версии к версии. В версии 4.2.x присутствуют классы для работы с протоколами FTP и HTTP. Для работы с протоколами TCP/IP предназначены такие классы, как QTcpServer, QTcpSocket для TCP и QUdpSocket для UDP;

▪ QtOpenGL — набор классов для работы с OpenGL;

- QtSql — набор классов для работы с базами данных с использованием языка структурированных запросов SQL. Основные классы данного модуля в версии 4.2.x: QSqlDatabase — класс для предоставления соединения с базой, для работы с какой-нибудь конкретной базой данных требует объект, унаследованный от класса QSqlDriver — абстрактного класса, который реализуется для конкретной базы данных и может требовать для компиляции SDK базы данных. Например, для сборки драйвера под базу данных Firebird/InterBase требует .h файлы и библиотеки статической линковки, входящие в комплект поставки данной БД;
- QtScript — классы для работы с Qt Scripts;
- QtSvg — классы для отображения и работы с данными Scalable Vector Graphics (SVG);
- QtXml — модуль для работы с XML, поддерживается SAX и DOM модели работы;
- QtDesigner — классы создания расширений QtDesigner'а для своих собственных виджетов;
- QtUiTools — классы для обработки в приложении форм Qt Designer;
- QtAssistant — справочная система;
- Qt3Support — модуль с классами, необходимыми для совместимости с библиотекой Qt версии 3.x.x;
- QtTest — модуль для работы с UNIT тестами;
- QtWebKit — модуль WebKit, интегрированный в Qt и доступный через её классы;
- QtXmlPatterns — модуль для поддержки XQuery 1.0 и XPath 2.0;
- Phonon — модуль для поддержки воспроизведения и записи видео и аудио, как локально, так и с устройств и по сети;
- QtCLucene — модуль для поддержки полнотекстового поиска, применяется в новой версии Assistant в Qt 4.4;
- ActiveQt — модуль для работы с ActiveX и COM технологиями для Qt-разработчиков под Windows.
- QtDeclarative — модуль, предоставляющий декларативный фреймворк для создания динамичных, настраиваемых пользовательских интерфейсов.

Также реализована технология WoC — widgets on canvas, с помощью которой реализована Plasma в KDE 4.1, Будет возможным использовать виджеты библиотеки Qt прямо в аплетах. Обеспечивает расположение виджетов на QGraphicsView с возможностью масштабирования и различных графических эффектов.

Библиотека использует собственный формат проекта, именуемый .pro файлом, в котором собрана информация о том, какие файлы будут скомпилированы, по каким путям искать заголовочные файлы и много другой информации. Впоследствии при помощи утилиты qmake из них получаются makefile для make-утилиты компилятора. Также есть возможность работы при помощи интеграторов с Microsoft Visual Studio 2003/2005/2008/2010. Совсем недавно стала доступна интеграция в Eclipse для версии библиотеки 4.x.x.

**Документация**

Одним из весомых преимуществ проекта Qt является наличие качественной документации, в отличие, например, от wxWidgets. Статьи документации снабжены большим количеством примеров. Исходный код самой библиотеки хорошо форматирован, подробно комментирован и легко читается, что также упрощает изучение Qt.

**Исходный код**

Исходный код, единый для всех вариантов лицензий, свободно доступен в Git-хранилище, расположенном на Gitorious-е по адресу — qt.gitorious.org/qt. Кроме самого исходного кода Qt, на Gitorious-е расположены хранилища сопутствующих библиотек, разрабатываемых авторами библиотеки и сообществом. Корневой адрес связанных с Qt официальных проектов — qt.gitorious.org. Для внесения собственного вклада в развитие Qt, можно склонировать официальное хранилище, внести необходимые исправления/улучшения в код, а затем подать запрос на объединение (Merge Request) ваших изменений с официальным кодом.

**Использование**

Qt используется в:

- Autodesk,
- Adobe Photoshop Elements,
- OPIE,
- Skype,
- Медиапроигрыватель VLC,
- VirtualBox,
- Google,
- HP,
- KDE,
- Lucasfilm,
- Panasonic,
- Philips,
- Samsung,
- Siemens,
- Volvo
- Walt Disney Animation Studios

Кроме того, на Qt основана среда рабочего стола KDE, графический интерфейс мобильной ОС MeeGo и Qt Creator — среда разработки на Qt.

## 1.1.9 Сравнение QT Creator и Visual Studio

Для разработки в рамках Qt любой QC гораздо лучше любой виденной мной IDE. Так что дальше речь пойдет только о c++ разработке без использования Qt.

Версии креатора до 2,0 не идут ни в какое сравнение даже с голым VS :

1. Нормальная рабочая система сборки только qmake, имеется в виду не только возможность собирать

проекты, а инструменты для автоматической работы с деревом проекта, и построение модели кода, для автокомплита и прочих плюшек.
2. Страшно доставлял глюк при смене раскладки - подвисание среды на несколько секунд/минут
3. Не удобная и не интуитивная система добавления кастомных действий при сборке
4. Страшно медленный и отладчик с бедным и ориентированным в основном на Qt набором визуализаторов, особенно при использовании CDB
5. Убогая поддержка шаблонов вообще и stl в частности, системой автодополнения

Правда к версии QT 2.3 голая студия, правда справедливости ради надо сказать до 2010 версии, стала выглядеть уже не так привлекательно.

Недостатки студии общие(включая 2010):

*Убогая навигация по коду, есть только переход по reference, который не всегда и срабатывает
*Более бедная система настройки и поддержки редактором code style
*Нету, либо нету на виду и под рукой, системы снипетов
*Система сборки менее гибкая чем с CMake, хуже интегрируется со всякими Continuous Integration
*Хорошо и без дополнительной платы поддерживает только один компилятор
*Трудность написания нетривиальных визуализаторов для отладчика
*Нет стандартных средст для рефакторинга, вообще, даже тухлого add implementation
*Медленно развивается, редко выпускаются релизы
*Стоит бабла, хотя если на работе, то это не критично конечно
*хуже/отсутствует поддержка нового стандарта
*слабо настраиваемая подсветка синтаксиса

**Недостаки VisualStudio**

*Глючащий, частенько вообще отключающийся авто комплит (особенно заметно при большом code base)
*Убогая система назначения путей для не стандартных библиотек - все пути прописываются общие, для всех проектов сразу
*Не кроссплатформенная
*Сложновато а иногда и дороговато использовать не мелкомягкие тулсеты.

**Достоинства VisualStudio**

*Крайне навороченный отладчик, присутствуют например такие фичи как перекомпилировать кусок кода и продолжить отладку, дофига встроенных, правда Microsoft specific в основном визуализаторов
*Легкость написания тривиальных визуализаторов для отладчика, под тривиальными, понимаются визуализаторы типов, реализация которых не скрыта, т.е. есть прямой, пусть и в обход модификаторов доступа, доступ к членам-данным, хотя для QC это тоже справедливо.
*Лучше поддержка Microsoft Specific, всяких COM, ATL и т.д, по крайней мере лучше из коробки.

*Хорошо и без дополнительной платы поддерживает один компилятор, используется максимум фич которые можно извлечь из мелкомягкого тулсета, особенно связанных с COM, всякие #import, навороченные _declspec, __property и тому подобное, при том всё это корректно обрабатывается комплитом, в тех случаях когда он пашет вообще.
*Лучеше поддержка дебажного рантайма, например бряки на ассертах.
*Огромное количество плагинов и дополнений, в том числе и бесплатных.

Проанализировав статьи и форумы в интернете, пообщавшись с программистами, я пришел к выводу, что для всего что слабо или вообще не использует Microsoft Specific, QT подходит лучше, да и обходится дешевле.

В противном случае всё зависит от наличия финансовых средств, если их достаточно, то VisualStudio в разы лучше.

### 1.1.10 Вывод из анализа предметной области.

В анализе предметной области были рассмотрены четыре аспекта:

- Аутентификация
- Выбор СУБД
- Заполнения/запросы БД
- Язык программирования C++ в различных средах.

В результате проведенного анализа объектной среды было решено использовать алгоритм аутентификации паролем из-за того метод наиболее удобен в среднестатическом случае, когда, не имеются средства на специальное оборудование.

Так как в  случае непосредственной разработки open source СППР следует использовать бесплатыные СУБД с многопоточным доступом и несомненно высоким уровнем безопасности и в первую очередь быстрое чтение данных, в этом проекте крайне подходит mySQL

С другой стороны мы имеем среду программирования Qt языка C++. Ее выбор был обусловлен тем, что наша программа относительно проста с точки зрения использования дополнительных компонентов (без .NET и параллельного программирования).Среда Qt вполне позволяет нам посредством кодирования реализовать алгоритм Крускала максимально удобно и быстро, используя все средства среды разработки.

Еще одним преимуществом является кросплатформенность среды Qt, ведь в настоящее время все больше и больше персональных компьютеров работают на базе операционной системы Ubuntu (Linux). Таким образом, есть возможность переноса разработанной программы в операционную систему Android.

В силу работы с Qt и c++ крайне удобно использовать  алгоритм запросов заполнений SQL;

Таким образом, на выходе анализа мы имеем выбранные нами алгоритмы аутентификации паролем, СУБД mySQL, SQL алгоритмы запросов/заполнениц и среду Qt 5.4 (самой стабильной версии на 15.05.2015 г. доступной для бесплатного скачивания и обладающей низким порогом вхождения из-за мношества доступной докментации на Английском и Русском языках) для реализации поставленной задачи.

## 1.2 Анализ объектной среды

Целью анализа объектной среды является изучение сред разработки и языков программирования, исследование объектов(картинки, органы), а так же выбор наиболее подходящей, исходя из начальных условий и конечного результата.
В наше время компьютеры и техника развиваются огромными темпами, но тем не менее необходимо помнить, что технические возможности современного оборудования ограничены, в связи с чем надо четко осознавать ограничения в рамках которых будет работать программа. Обработка изображений в целом и распознание изображений в частности является крайне ресурсозатратным процессом, требующим от компьютера большое количество оперативной памяти и вычислительных мощностей, кроме того сами изображения занимают много места, что выливается в необходимость организации огромного архива для их хранения. Однако со временем возможности компьютеров увеличиваются и становится возможным более быстрое применение сложных методов обработки, что повышает как точность и качество обработки изображений, так и общее быстродействие системы. Как уже упоминалось выше, большинство существующих систем являются жестко запрограммированными и не могут существенно изменяться, что приводит к их крайне быстрому устареванию и несоответствию требованиям времени.

**Характеристики изображений**

| | |
|---|---|
| Формат изображения: | *.jp |
| Ширина: | 720 |
| Высота: | 576 |
| Глубина цвета (бит/пиксель): | 24 |

Объем изображения: 24*720*576=9953280 бит = 1244160 байт = 1,18 Мбайт

Необходимо для резервирования памяти: Кол-во изображений * 1,18 Мбайт=53,1 Мбайт

**Требуемый объем по текстовой информации:** 1 символ = 27 бит = 0,00000286 Мбайт

Количество символов в строке: 60 символов. Объем 1й строки = 0,000172 Мбайт

Количество строк на странице : 30х0,000172 Мбайт = 0,00516 Мбайт

Количество страниц 20х0,00516 Мбайт = 0,1032 Мбайт

Т.к. объем текстовой информации много меньше объема изображения, то им можно пренебречь.

Сколько занимает пиксел в графическом режиме будет зависеть от поддерживаемых цветов:

в монохромном режиме пиксел — это один бит (1/8 байта)

в 16-цветном режиме пиксел — это 4 бита (1/2 байта)

в 256-цветном пиксел — это 1 байт

для тру-колор пиксел — это 24 бита (3 байта)

Критерии для выбора являются:

- Точность.
- Простота установки и работы.
- Простой и понятный интерфейс.
- Доступность программы.
- Многофункциональность

*Аденокарцинома простаты*

*Признаки аденокарциномы.*

1. Инфильтративный рост (ацинусы опухоли расположены между и/или окружают нормальные ацинусы).
2. Отсутствие базального слоя клеток.
3. Увеличение ядра в размерах.
4. Видимые ядрышки.
5. Голубой муцин в просвете желез.
6. Розовый аморфный секрет в просвете желез.
7. Периневральная инвазия.
8. Амфофильная цитоплазма.

*Признаки аденокарциномы 2*

1. Атипичное строение ацинуса: мелкие ацинусы, вариация в размерах незначительна
2. Из ацинусов формируется хорошо очерченный узел

*Признаки аденокарциномы 3*

1. Атипичное строение ацинуса: мелкие ацинусы неправильной формы

*Признаки аденокарциномы 4*

1. Атипичное строение ацинусов:
2. Слияние ацинусов с формированием
    1. Ветвящихся желез
    2. Крибриформных структур
3. Гломеруляции

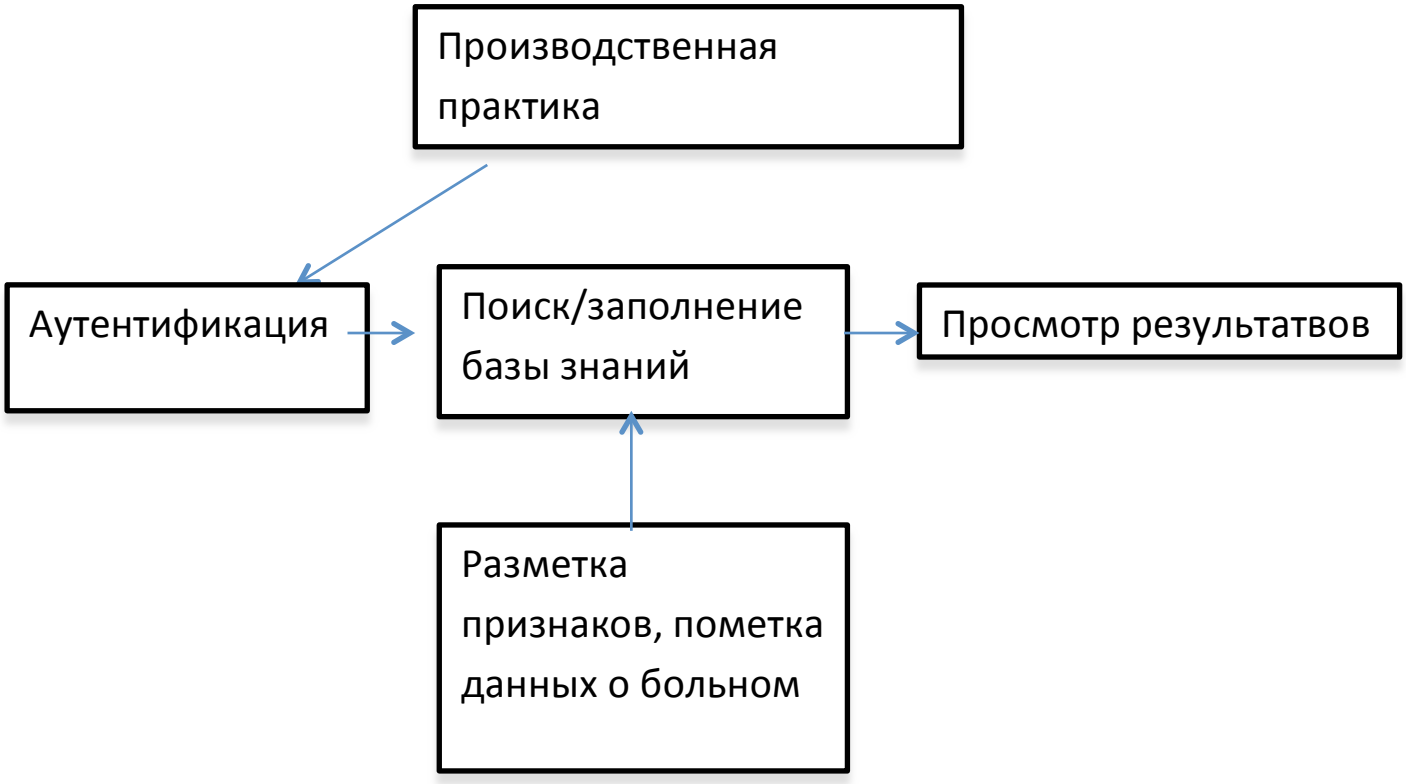
*Признаки аденокарциномы 5*

1. Атипичное строение ацинусов:
2. Слияние с формированием крибриформных структур с некрозом в центре (комедонекроз).
3. Цепочки из клеток (видны как цепочки из ядер)
4. Отдельно лежащие клетки.
5. Инфильтрация стромы (ацинусы опухоли расположены между и/или окружают нормальные ацинусы).

## 1.3 Разработка требований

### 1.3.1 Функциональные требования к разрабатываемой системе

Разрабатываемая система должна представлять собой научно – исследовательскую систему программных продуктов изображенных на схеме:




БД должна предоставлять информацию о тканях рака предстательной железы (снимки, морфологические характеристики) в ответ на запросы пользователя. Главным критерием качества системы является точность, менее важно быстродействие.

Изображение с растровой сеткой размером 683x768 и 8 битной глубиной цвета занимает 37.7Кбайт на диске (т.к. 1 пиксель = 8 бит). Расчет высчитывается следующим образом: размер изображения умножается на 24(8 бит на каждую компоненту RGB). Отсюда 683x768x24.
1 килобайт = 1 Кб = 1024 байта
1 мегабайт = 1 Мб = 1024 Кб
1 гигабайт = 1 Гб = 1024 Гб
Так как база данных может содержать в себе не больше 10000 изображений, то были сформулированы следующие требования:

При загрузке Базы данных есть 3 кнопки где в ниспадающем меню можем выбрать; быстрая загрузка (карточка пациента), пациенты (добавление и изменение), визуальные признаки (добавление и изменение), болезни (добавление и изменение), изображения(добавление и изменение) и отчет по пациентам, визуальным признакам и болезням(где выводятся занесенные данные), расчет постановки диагноза

**Требования к разрабатываемой системе**

**Програмная система** должна обеспечить выполнение следующих функций:

- ✓ Разработано в среде Qt с использованием языка кодирования c++
- ✓ Программа должна представлять собой исполняемый файл формата *.exe

- ✓ Все необходимые библиотеки для запуска программы содержатся в папке учебного пособия (*.dll)
- ✓ Возможность масштабирования главного окна программы и вместе с ним элементов формы без искажения и потери функуиональности
- ✓ Работа с различными разрешениями монитора (800*600, 1024*768, 1600*900, 1920*1080)
- ✓ Не использовать (подключать) библиотеки большого рамера данных (OpenCV)
- ✓ запускаться с использованием ярлыков и через командную строку
- ✓ загружать изображение в программу
- ✓ выводить информацию о выделенных областях на монитор
- ✓ иметь интерфейс для пользователя системы имеет спокойные цвета и логичное расположение элементов управления на форме
- ✓ обрабатывать исключения (ошибки) во время работы
- ✓ реализован в среде разработки Qt с использованием open source библиотек
- ✓ язык кодирования c++
- ✓ время обработки каждого запроса в ПО не более 10 сек.

**Ограничения накладываемые на входные изображения**:

- ✓ не требующие предобработки
- ✓ разрешения BMP JPEG PNG TIFF
- ✓ максимальный размер хранимых в программе не более 1 Gbyte

1 изображение(единовременно) = 500 MByte
количество иных данных не более = 500 Mbyte

### 1.3.2 Аппаратные требования к разрабатываемой системе

Процессор: IntelCore 2 Duo/AMDTurionX2 1,8ГГц
Оперативная память: 2048 Мб памяти
Видеоадаптер: 3D-ускоритель с 128 Мб памяти
Жесткий диск (винчестер): 500 Мб
Монитор: 15,4 дюймов
Клавиатура и мышь.

Это набор рекомендуемых аппаратных требований, использование аппаратных ресурсов в более низкими характеристиками может замедлить выполнение сегментации.

### 1.3.3 Программные требования к разрабатываемой системе

Операционная система: Microsoft Windows XP, Vista, 7, 8, 8.1
Среда разработки: Qt 5.4

Это набор рекомендуемых программных требований, использование альтернативных программных ресурсов может привести к возникновению неизвестных исключений (несовместимости).

### 1.3.4 Требования к тестированию разрабатываемой системы

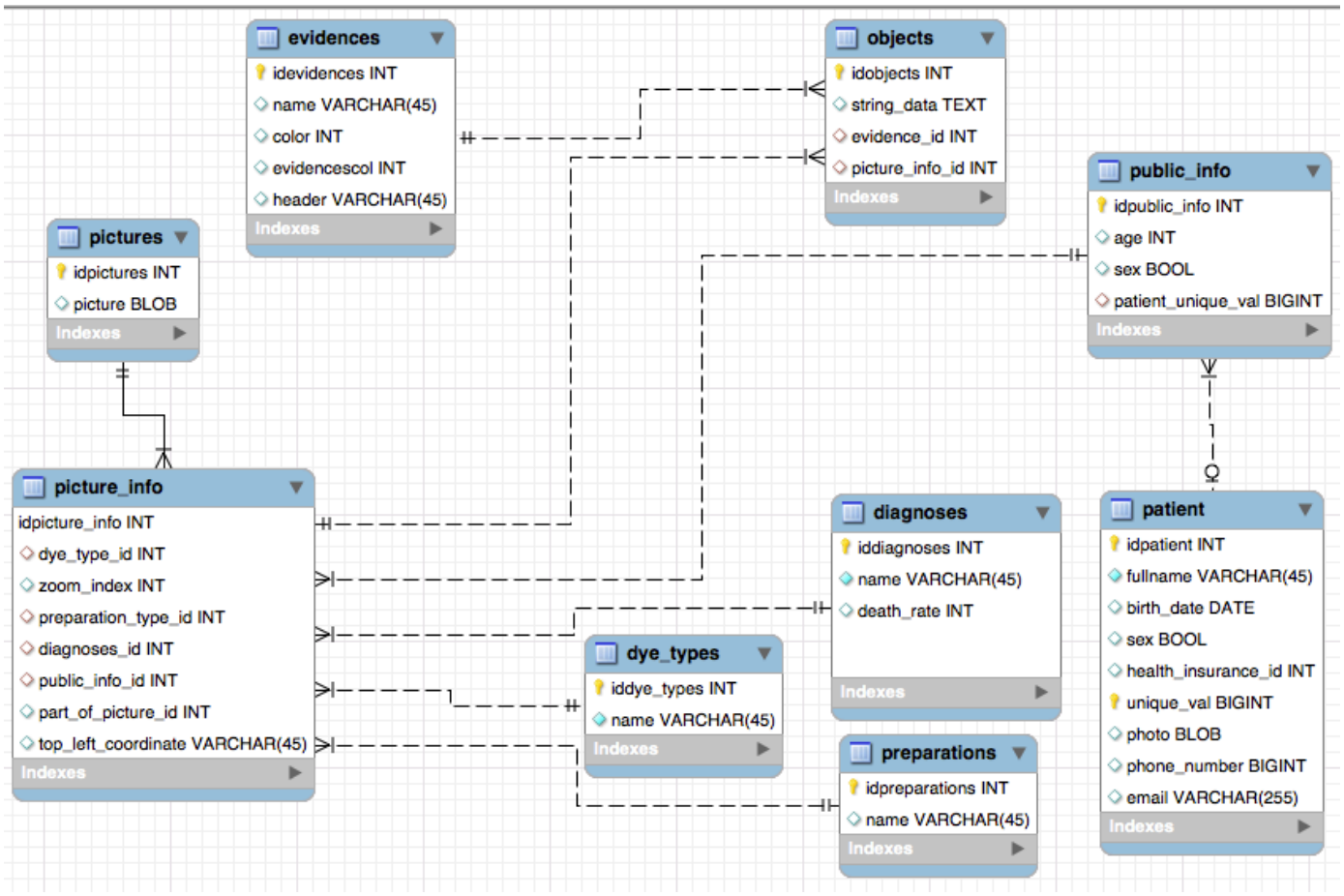
При проведении тестирования необходимо протестировать следующие основные возможности программного продукта:

- запуск программы
- возможность загрузки изображения в программу
- корректность отображения интерфейса пользователя (визуально)

# Глава 2. Практическая часть

## 2.1 Физическая реализация

Структура базы данных.




Заполнение таблиц базы данных производится средствами Qt при помощи класса QSqlDatabase. Класс QSqlDatabase предоставляет абстрактный интерфейс для доступа к базе данных. Он использует специальный QSqlDriver базы данных для доступа и манипуляции данными.

Подключение к базе данных производилось следующим образом:

QSqlDatabase db; - создаем объект класса QSqlDatabase

```
db = QSqlDatabase::addDatabase("QMYSQL");
- подключаем драйвер дазы данных MySQL
    db.setHostName("127.0.0.1");
    db.setPort(3306);
   -  Вводим адрес сервера базы данных
    db.setDatabaseName("mydb"); //название бд
    db.setUserName("root");
    db.setPassword("12345");
- вводим имя пользователя и пароль
    if (db.open())
        qDebug() << "База данных подключена";
```

```
    else
        qDebug() << "Ошибка подключения";
- проверяем получилось ли подключится
```

Заполнение таблицы производится следующим образом

```
    QSqlQuery q;
//создание объекта класса QSqlQuery необходимого для формирования запроса

//получение данных пациента из заполеной формы в окне программы:
    QString first_name = ui->lineEdit1n->text();
    QString last_name = ui->lineEdit2n->text();
    QString patronymic = ui->lineEdit3n->text();
    QString full_name = first_name+" "+last_name+" "+patronymic;
    bool sex;
    if(ui->comboBoxS->currentIndex() == 0)
        sex = 0;
    else
        sex = 1;
```

//создание индивидуального идентификатора пользователя

```
qsrand(QTime(0,0,0).secsTo(QTime::currentTime()));
    uint unique_val=qrand()%10000;


    q.exec("select MAX(idpatient) from patient");
    q.next();
    int idpatient=q.value(0).toInt() + 1;

    QString health_insurance_id = ui->lineEditMI->text();
    QString phone_number = ui->lineEditPh->text();
    QString email = ui->lineEditEm->text();
    uint y = ui->spinBoxYr->value();
    uint m = ui->comboBoxM->currentIndex()+1;
    uint d = ui->comboBoxM->currentIndex()+1;
    QDate birth_date = QDate(y,m,d);
    QFile photo(image_path);//ввод фотографии, как массива QByteArray
    photo.open(QIODevice::ReadOnly);
    QByteArray bytes = photo.readAll();
//формирование шаблона запроса
    q.prepare("insert into
patient(idpatient,fullname,birth_date,sex,health_insurance_id,unique_val,phone
_number,email,photo) values
(:idpatient,:fullname,:birth_date,:sex,:health_insurance_id,:unique_val,:phone
_number,:email,:photo)");
    q.bindValue(":idpatient",idpatient);
    q.bindValue(":fullname",full_name);
    q.bindValue(":birth_date",birth_date);
    q.bindValue(":sex",sex);
    q.bindValue(":health_insurance_id",health_insurance_id);
    q.bindValue(":unique_val",unique_val);
    q.bindValue(":phone_number",phone_number);
    q.bindValue(":email",email);
    q.bindValue(":photo",bytes)ы;
```

q.exec(); //отправка запроса

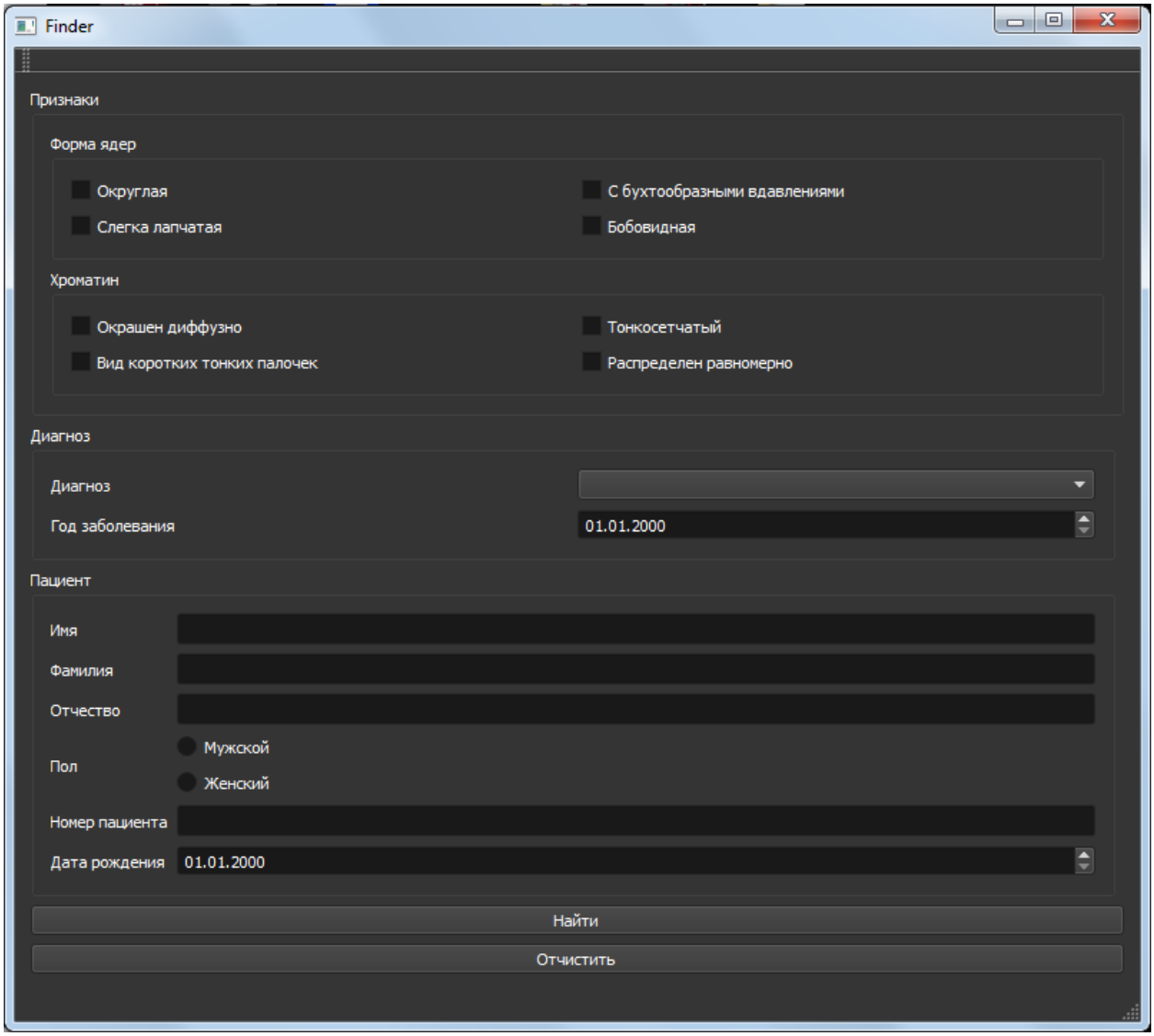
## Система поиска изображений

Задачи

1. Создать систему поиска изображений в базе данных.

## Требования

1. Поиск производится путем заполнения полей в окне программы.
2. Поиск должен производится по данным пациента, по диагнозу, по признакам.
3. Поиск должен производится по всем заполненным полям.
4. Не заполненные поля игнорируются.




## Реализация

Поиск может производится по диагнозу, пациенту и признакам. Все эти группы соответствуют своим таблицам в БД, что упрощает поиск.

Поиск производится по всем заполненным полям окна поиска. То есть можно, например, искать одновременно по признакам и по диагнозу. Не заполненные поля игнорируются.

```
    QSqlQuery q;

QString first_name = ui->lineEdit1n->text();
```

# Заключение

Изучена предметная и объектные области на основе них разработанны требования, выбранны алгоритмы аутентификации паролем, СУБД mySQL, SQL алгоритмы запросов/заполнениц и среду Qt 5.4 (самой стабильной версии на 15.05.2015 г. доступной для бесплатного скачивания и обладающей низким порогом вхождения из-за мношества доступной докментации на Английском и Русском языках) для реализации поставленной задачи.

На данный момент большая часть системы сделана: спроектирована база данных, разработан алгоритм заполнения полей и архитектура проекта в целом, остается доработать алгоритм разметки изображения.

# Список литературы

1. Физика визуализации изображений в медицине: в 2-х т. Пер. с англ.; под ред. С. Уэбба. - М.: Мир, 1991. - Т. 2 - 406 с.

2. Виллевальде, А. Ю. О системном подходе к медицинской визуализации / А. Ю. Виллевальде // Известия СПбГЭТУ «ЛЭТИ» (Известия Государственного электротехнического университета). Сер. Биотехнические системы в медицине и экологии. — 2006. — Вып. 1. - С. 37-43.

3. Прет, У. Цифровая обработка изображений: в 2-х т. Пер. с англ. - М.: Мир, 1982. - Т. 1 - 306 е., Т. 2 - 480 с.

4. Методы компьютерной обработки изображений; под ред. В.А. Сойфера. -М.: ФИЗМАТЛИТ, 2003. - 784 с.

5. Виллевальде, А. Ю. Обобщенный подход к построению систем анализа и обработки медицинских малоконтрастных изображений / А. Ю. Виллевальде // Труды Международной конференции по мягким вычислениям и измерениям. Санкт-Петербург, Издательство СПбГЭТУ «ЛЭТИ», 25-27 июня 2007 г. -С. 116-119. Форсайт, Д. А. Компьютерное зрение. Современный подход / Д. А. Форсайт, Дж. Понс. -М.,

СПб., Киев: Вильяме, 2004. - 926 с.

8. Петров, М. Н. Компьютерная графика: Учебник / М. Н. Петров, В. П. Мо- лочков. - СПб.: Питер, 2002. - 736 с.

9. C# для профессионалов, Робинсон, С.; Корнес, О.; Глинн, Д, Издание: М.: Лори, 2005. – 1480 с. www.wikipedia.ru

10. www.simple-cs.ru

11. Материалы habrahabr.ru, computergraphics.ru, ru.cppreference.com выложенные в общий

12. доступ
13. http://www.jetinfo.ru/stati/silnye-i-slabye-storony-nosql