МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование кафедры)

09.04.03 Прикладная информатика
(код и наименование направления подготовки, специальности)

Информационные системы и технологии корпоративного управления
(направленность (профиль)/специализация

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему «Математическое и программное обеспечение информационных систем с настраиваемой концептуальной моделью данных»

Студент А.И. Бортник _____
(И.О. Фамилия) (личная подпись)

Научный руководитель О.М. Гущина _____
(И.О. Фамилия) (личная подпись)

Руководитель программы д.т.н., доцент, С.В. Мкртычев _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«_____»_____20_____г.

Допустить к защите
Заведующий кафедрой к.т.н., доцент, А.В. Очеповский _____
«_____» _____20_____г. (личная подпись)

Тольятти 2019

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы. В настоящее время информационные системы прочно вошли во все сферы человеческой деятельности: бизнес, производство, управление, образование и т. д. Это объясняется все возрастающим объемом и обрабатываемой информации, и задач управления, с которыми приходится сталкиваться человеку. Эффективность работы сотрудника, а следовательно, и всей организации зависит теперь от эффективности используемых информационных систем.

Поэтому современный подход к управлению опирается на использование информационных технологий, причем их количество тем больше, чем больше предприятие. С ростом сложности решаемых задач происходит рост сложности информационных систем, который отражается в увеличивающейся сложности архитектуры информационных систем. Развиваются подходы к ее реализации, средства и методологии ее описания. В множестве сфер человеческой деятельность применение находят информационные системы, которые способны адаптироваться под предметную область, при этом, адаптация происходит не на этапе разработки модели, а на этапе ее использования, и пользователю предоставляется возможность управлять сущностями и их свойствами. Подобные информационные системы позволяют компаниям эффективнее настраивать бизнес-процессы под свои потребности, а при необходимости вносить в них изменения без внесения изменений в исходный код программного продукта. Одним из способов создания многоцелевых информационных систем является обеспечение гибкости информационной системы путем внедрения возможности внесения изменений в концептуальную модель базы данных. При этом пользователям предоставляются инструменты, которые позволяют изменять конфигурацию концептуальной модели базы данных в соответствии с изменением в предметной области.

Изучением гибкости информационных систем занимались ряд отечественных и зарубежных ученых, среди которых: А. Савидис, Б. Боэм, Д. Гебауер, Д. Зенг, Д. Тернер, Е. Монтейро, К. Паттен, Л. Бродский, Л. Жао, М. Хатлинг, Н. Дункан, О.

Хансет, Р. Банкер, Р. Доусинг, С. Мельник, С. Мукержи, Т. Бэрд, Ф. Бернштайн, Ф. Шобер. Системы з настраиваемой концептуальной моделью данных изучали: А. Гаврилов, А. Зуенко, А. Симановский, А. Тенцер, А.С. Клещев, В. А. Орлов, В.Э. Вольфенгаген, Д. Джонсон, Д. Кондраков, Д. Рябко, Е. Павлова, Л. Бродский, Л. Лядова, Л. Рейнгольд, Л. Черняк, М. Стоунбрейкер, М. Шпаков, П. Надкарни, П. Олейник, Р. Игнатович, Р. Морс, С. Завозкин, С. Мельник, Д. С. Целуйко.

Современные программные архитектурные решения позволяют разрабатывать качественные и надежные информационные системы, которые обеспечивают реализацию множества потребностей организаций и предприятий. Одним из популярных архитектурных решений при создании программного обеспечения является шаблон проектирования Модель-Представление-Контроллер (MVC). Данный шаблон позволят создавать большие легко масштабируемые приложения, которые не сложны в поддержке и позволяют быстро расширять свой функционал за счет разработки и подключения дополнительных модулей. Представленная архитектура будет использована для построения информационной системы с настраиваемой концептуальной моделью базы данных.

Объект исследования: информационная система с настраиваемой концептуальной моделью данных.

Предмет исследования: математическое и программное обеспечение информационных систем с настраиваемой концептуальной модель.

Цель работы: исследование математического и программного обеспечения, используемых при построении информационных систем с настраиваемой концептуальной моделью данных.

Задачи работы:

- провести анализ технологий разработки информационных систем с настраиваемой концептуальной моделью базы данных;

– рассмотреть особенности математического обеспечения информационных систем с настраиваемой концептуальной моделью данных;
– разработать проект информационной системы с настраиваемой концептуальной моделью данных;
– провести экспериментальную проверку математического и программного обеспечения информационной системы с настраиваемой концептуальной моделью данных.

Методология исследования. В процессе исследования применялся научный аппарат информатики, методы проектирования и разработки программного обеспечения, и такие научные методы исследования, как: абстрактно-логический метод; анализ научной и учебной литературы; классификация; метод обобщения; методы проектирования и разработки веб-приложений; моделирование; описательный метод; проектный метод; синтез; системный анализ и подход; сравнительный анализ; тестирование; эмпирический метод; метод концептуального проектирования баз данных.

Научная новизна исследования состоит в исследовании математического и программного обеспечения информационных систем с настраиваемой концептуальной моделью данных и разработке проекта гибкой информационной системы с применением соответственного математического аппарата и программного обеспечения.

Практическая значимость исследовательской работы заключается в том, что результаты исследования могут быть использованы в деятельности организаций множества предметных сфер. Теоретические наработки будут полезны при построении аналогичных систем.

Структура работы: работа состоит из введения, четырех глав и заключения, содержит 107 страниц машинописного текста, 9 таблиц, 35 рисунков, список литературы из 86 наименований.

Глава 1 РАЗВИТИЕ ИНФОРМАЦИОННЫХ СИСТЕМ. ГИБКИЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ

1.1 Особенности понятий информационная система и гибкая информационная система. Архитектура информационных систем

В настоящее время понятие «информационная система» является очень распространенным, при этом множество исследователей дают разные определения этому понятию [31]. По мнению М. Р. Когаловского [15], определения понятия «информационная система» главным образом связаны с предметной сферой их применения, в контексте которые исследователи изучают это понятие. Ряд авторов [33, 40, 54] рассматривают «информационную систему» в функциональном аспекте, как инструмент ввода, сохранения, обработки и передачи информации. Е. С. Охотникова [30] дает определение информационной системы, основанное на понятии информационных технологий — это организационно-упорядоченная взаимосвязанная совокупность средств и методов информационных технологий, используемых для хранения, обработки и выдачи информации в интересах достижения поставленной цели. Указанное определение можно принять и в его контексте осуществлять дальнейшее рассмотрение особенностей построение информационных систем с настраиваемой концептуальной моделью данных.

Важным направление изучения особенностей информационных систем является рассмотрение процессов их развития, что обосновано тем, что современный уровень новых технологий может вносить ряд изменений при создании, использовании и поддержке информационных систем. В связи с этим, информационные системы вынуждены постоянно развиваться и совершенствоваться для того, чтобы оставаться полезным для пользователя инструментом. Ю. И. Рогозов выделяет три наиболее важных фактора, влияющих на развитие информационных систем [35]:

– изменение предметной области и условий функционирования;

– развитие возможностей аппаратной составляющей информационных систем;
– развитие подходов к программной реализации информационных систем.

Для снижения влияния указанных факторов на развитие информационных предлагается придание гибкости информационным системам за счет настраиваемой концептуальной модели данных. Пользователям таких систем представляется возможность адаптации системы под требования предметной области, когда структура данных в базе определена на жестко, а может изменяться пользователем.

Чтобы понять положение адаптивных информационных систем в пределах современных информационных технологий и возможности их реализации необходимо рассмотреть классификацию информационных систем, которая возможна по разным критериям и признакам. Ряд исследователей, изучающих информационные системы, выделяют такой признак, как масштаб информационной системы. Соответственно этому признаку выделяют одиночные, групповые и корпоративные информационные системы.

Одиночные информационные системы реализуются без использования сетевых технологий на локальном персональном компьютере. В состав системы включается несколько приложений, работающих с общим хранилищем данных. С системой одновременно может работать только один пользователь. Приложения такого класса создаются на базе хранилища данных в виде набора файлов либо с помощью встраиваемых или локальных систем управления базами данных (СУБД). Среди таких: FoxPro, Microsoft Access, SQLite, BerkeleyDB, InnoDB.

Групповые информационные системы строятся на основе локальной вычислительной сети, в которой выделяется сервер, для хранения данных и группа клиентских ПК, предназначенных для работы группы пользователей. Системы такого уровня предназначены для решения задач отделов и подотделов организаций. Для создания таких систем используется архитектура клиент-сервер (файл-сервер или с сервером БД). Для хранения данных в основном используются

такие СУБД как InterBase/FireBird, MySQL, PostgreSQL, Oracle, DB2, Microsoft SQL Server.

Корпоративные информационные системы ориентированы на крупные компании и включают автоматизацию большинства бизнес-процессов организаций. Для реализации подобных систем используется корпоративная вычислительная сеть с поддержкой территориально удаленных узлов или сетей. При построении таких систем используется архитектура клиент-сервер со специализацией серверов или многоуровневая архитектура. Для организации хранилища данных используется один или несколько серверов, на которых установлены СУБД корпоративного масштаба: Oracle Database, IBM DB2 и Microsoft SQL Server и т. п.

Вторым классификационным исследователи выделяют сферу применения, т. е. предметную область в пределах которой применятся информационная система. По этому признаку информационные системы как правило делятся на четыре группы (рис. 1): системы обработки транзакций, системы принятия решений, информационно-справочные системы и офисные информационные системы.

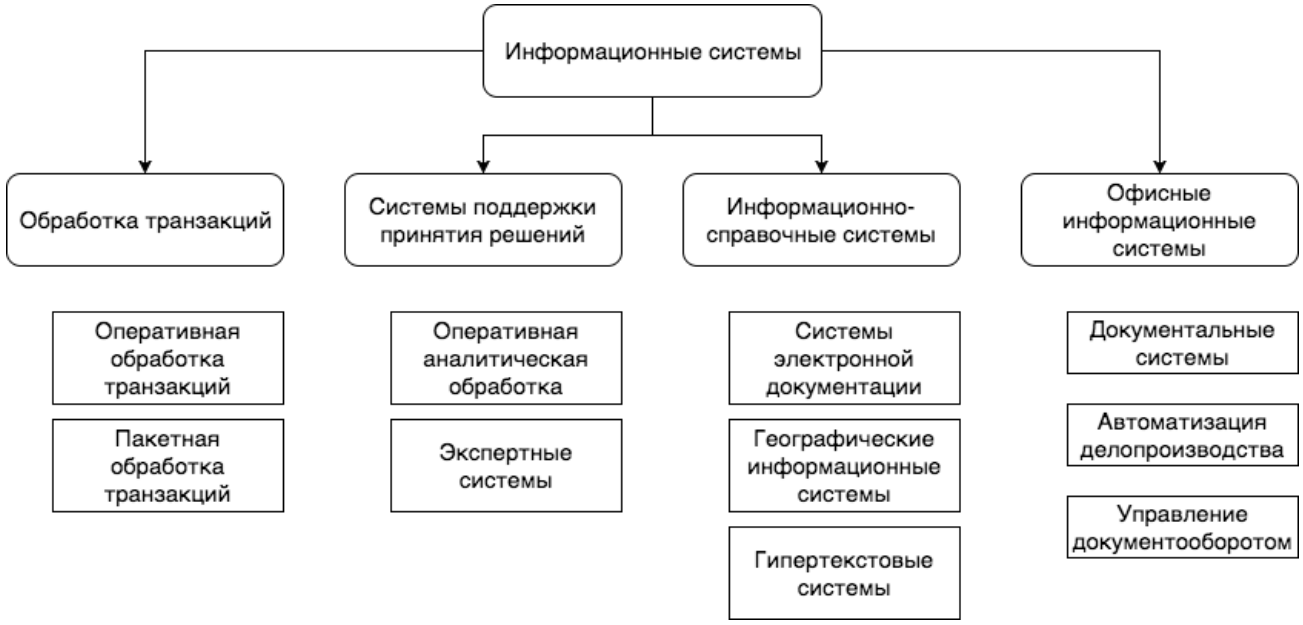

Рисунок 1 – Классификация информационных систем по сфере применения

В контексте сферы применения информационной системы приобретает новый смысл проектирование, разработка и внедрение адаптивных информационных систем, объяснить это можно тем, что адаптация структуры информации, которая хранится в базе данных позволяет обеспечить гибкость информационной системы, которая позволит включать в структуру базы данных новые сущности предметной области.

Обширный класс информационно-справочных систем основан на гипертекстовых документах и мультимедиа. Наибольшее развитие такие информационные системы получили в сети Интернет.

Класс офисных информационных систем нацелен на перевод бумажных документов в электронный вид, автоматизацию делопроизводства и управление документооборотом.

Кроме масштаба и предметной сферы исследователи выделяют такой признак, как архитектура, которая определяет мощность и масштабируемость информационных систем. По этому признаку выделяют такие информационные системы (рис. 2):

1. системы настольные (локальные);
2. системы, распределённые на основе архитектуры:
   - файл-сервер;
   - клиент-сервер;
   - многоуровневой архитектуры;
   - Интернет/интранет-технологий.

В настольных (локальных) информационных системах все компоненты находятся на одном компьютере, в распределенных – части системы размещены на разных вычислительных узлах. в зависимости от распределения функциональных компонентов можно отнести распределенную ИС к одному из классов. В таблице 1 представлены типовые компоненты информационных систем, с помощью которых можно описать архитектуру системы.

Таблица 1 – Типовые функциональные компоненты информационной системы

Обозначение	**Наименование**	**Характеристика**
PS	Presentation Services (средства представления)	Обеспечивает взаимодействие между пользователем и информационной системой, обеспечивает ввод от пользователя и отображают результаты работы ИС
PL	Presentation Logic (логика представления)	Определяет правила взаимодействия пользователя и ЭВМ. Ставит соответствие между действиями пользователям функциями ИС, а также определяет правила отображения результатов работы.
BL	Business/Application Logic (прикладная логика)	Совокупность алгоритмов и правил для обработки данных, принятия решений, вычислений и операций, которые должно выполнить приложение.
DL	Data Logic (логика управления данными)	Набор операций с базой данных, которые нужно выполнить для реализации прикладной логики управления данными (набор SQL-операторов)
DS	Data Services (операции с базой данных)	Операции в СУБД, вызываемые для выполнения логики управления данными, такие как
FS	File Services (файловые операции)	Операции чтения и записи данных из файлов для СУБД (файловые операции) и других компонентов (Реализуется ОС)

В информационных системах файл-серверной архитектуры выделяется 2 уровня: сервер, реализующий функции компонента FS (файловые операции), и клиент, включающий все остальные функциональные компоненты. Коммуникации между файловым сервером и клиентом осуществляются посредством локальной вычислительной сети.

Компонент FS на файл-сервере реализуется средствами сетевой операционной системы и отвечает за предоставление доступа к набору файлов, хранящихся на сервере. При этом файл-сервер не обладает информацией о структуре данных, хранящихся в файлах.

Рисунок 2 - Классификация информационных систем по архитектуре

Создание файл-серверного приложения заключается в разработке компонентов клиентского приложения, реализующего компоненты PS, PL, а также логику обработки BL и управление данными (DS и DL). Разработанное приложение реализуется в виде набора исполняемых модулей либо в виде специального кода для интерпретации (рис. 3).

Минусом подобной системы является неэффективная система взаимодействия с базой данных. При выполнении запросов к данным клиенту могут

передаваться большие объемы данных, так как файловый сервер не обладает информацией о структуре хранящихся данных, клиентское приложение вынуждено считывать содержимое файлов, и уже на стороне клиента выбирать необходимую информацию. Следствием такой организации является высокая загрузка сети и непредсказуемость времени реакции системы на действия пользователя. Данный недостаток усиливается при увеличении количества пользователей, работающих с системой, объема данных, хранящихся на сервере.

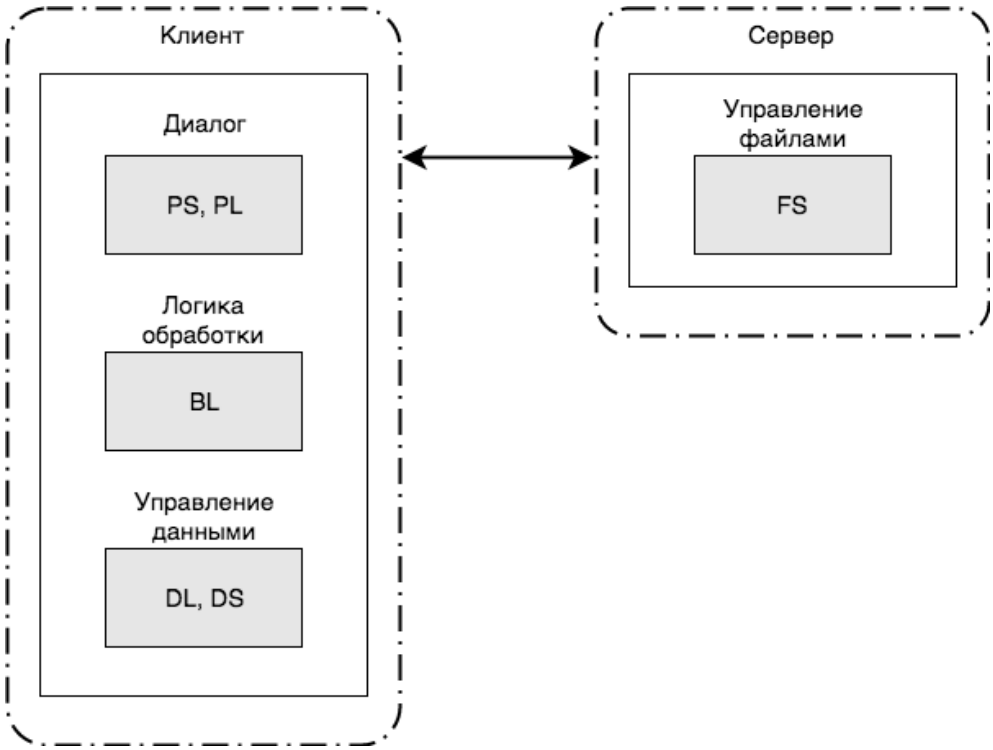

Рисунок 3 - Классический вариант файл-серверной информационной системы

При выборе оборудования для реализации системы необходимо учитывать требования к каждому уровню, которые определяются распределением функциональных компонентов. Так, для файл-серверной архитектуры нужно принимать во внимание требования к ряду параметров аппаратного обеспечения, как со стороны клиента (производительность центрального процессора, оперативная память, периферийные устройства), так и сервера (дисковая подсистема – скорость, объем, надежность).

Архитектура клиент-сервер подобно файл-серверной архитектура включает два уровня, при этом отличием в ней является размещение компонентов, которое дает возможность исключить часть негативных сторон файл-серверной архитектуры.

В архитектуре клиент-сервер компонент работы с базой данных (DS) переносится на сервер и реализуется в виде системы управления базой данных (СУБД). СУБД – специальный программный компонент, реализующий операции с данными, понимающий запросы на языке структурированных запросов SQL (Structured Query Language) и выполняющих поиск, сортировку и агрегирование информации. Важной особенностью серверов БД является наличие справочника данных (рис. 4).

Показанное на рис. 4 разделение компонентов позволяет снизить количество предаваемой от сервера к клиенту информации, так как наличие репозитория и компонента DS на сервере позволяют осуществлять фильтрацию и пересылать только необходимые клиенту (запрошенные) данные. Использование данной архитектуры позволяет строить системы большего масштаба, с более сложной логикой работы, что, в свою очередь, обостряет проблему администрирования приложений, разбросанных по различным клиентским узлам и имеющим различный набор функций, реализованных компонентом BL.

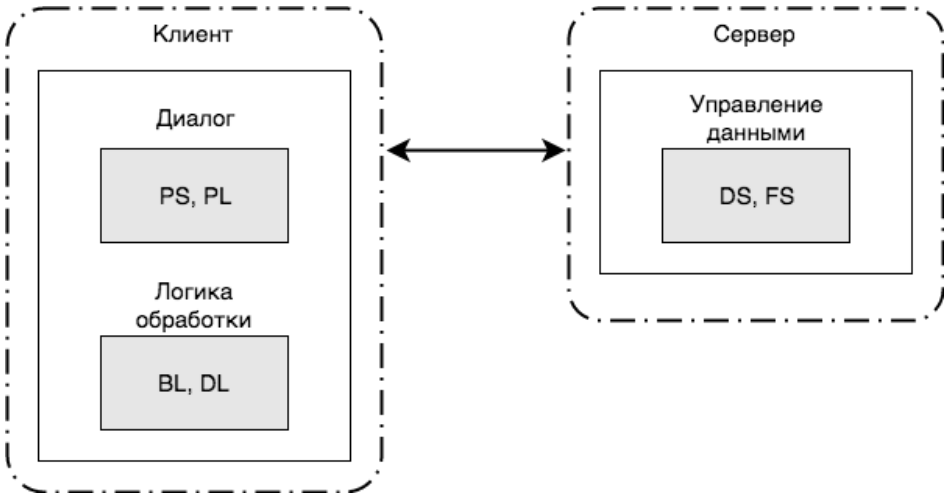

Рисунок 4 – Классический вариант клиент-серверной информационной системы

Использование такой технологии улучшает целостность приложений и БД. Упрощается сопровождение бизнес-логики, а также увеличивается безопасность данных (у клиентских приложений нет прямого доступа к данным).

При выборе оборудования для реализации системы необходимо учитывать требования к каждому уровню, которые определяются распределением функциональных компонентов. Так, для клиент-серверной архитектуры нужно принимать во внимание требования к следующим аппаратным компонентам средств вычислительной техники: клиент - процессор, периферийные устройства, сервер – дисковая подсистема, оперативная память, процессор.

При использовании хранимых процедур на сервер ложится большая вычислительная нагрузка, так как вся бизнес-логика для всех клиентов обрабатывается на сервере. При большом количестве обращений сервер может быть перегружен.

Решением проблем клиент-серверной архитектуры может стать использование многоуровневой архитектуры. В многоуровневой архитектуре, по сравнению с архитектурой клиент-сервер, вводится дополнительный уровень бизнес-логики (рис. 5).

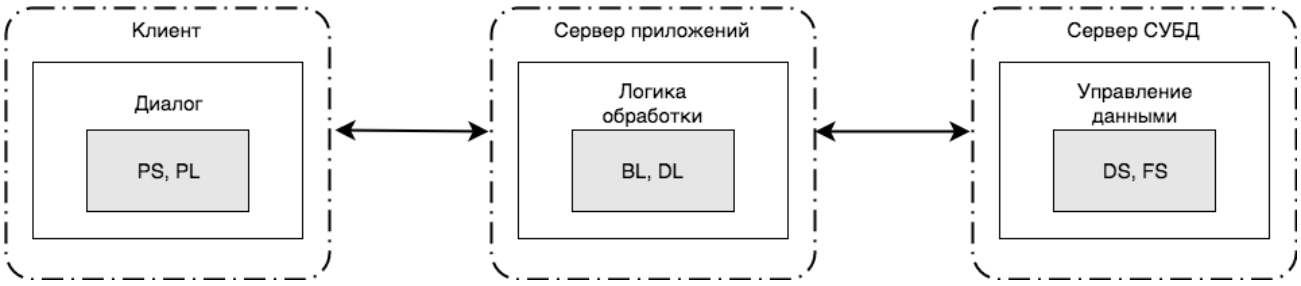

Рисунок 5 – Классический вариант многоуровневой информационной системы

При выборе оборудования для реализации системы с трехуровневой архитектурой необходимо учитывать требования к каждому уровню, которые определяются распределением функциональных компонентов. При этом следует принимать во внимание требования к следующим аппаратным компонентам

средств вычислительной техники: клиент (процессор, периферийные устройства), сервер (дисковая подсистема, оперативная память, процессор).

Примером многоуровневой Интернет/Интранет-архитектуры можно назвать веб-приложения. Их отличительной особенностью является передача запросов и результатов их обработки через сеть Интернет и использование в качестве клиентского приложения (как правило) стандартного Интернет-браузера (Internet Explorer, Chrome, Opera и др.) (рис. 6).

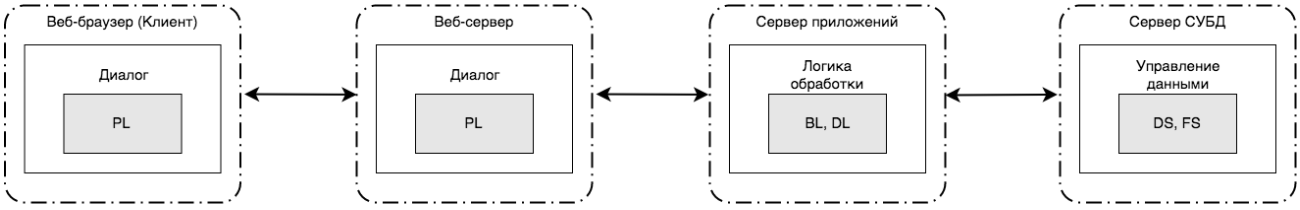

Рисунок 6 – Классический вариант многоуровневого Интернет/Интранет-приложения

При выборе оборудования для реализации многоуровневой Интернет/Интранет-архитектуры необходимо учитывать требования к каждому уровню, которые определяются распределением функциональных компонентов. При этом следует принимать во внимание требования к следующим аппаратным компонентам средств вычислительной техники: клиент (процессор, периферийные устройства), сервер (дисковая подсистема, оперативная память, процессор), интернет-канал (пропускная способность).

Рассмотренные архитектурные решения также подходят для построения адаптивных информационных систем, так как обеспечивают наличие всех элементов, на основании которых можно построить систему с настраиваемой концептуальной моделью данных.

1.2 Базы данных в построении гибких информационных систем (систем с настраиваемой концептуальной моделью данных)

Как показано в п. 1.2 важной составляющей информационной системы является база данных, при этом, в информационных системах с настраиваемой концептуальной моделью данных база данных приобретает новые смыслы, поскольку позволяет управлять данных на уровне концептуальных данных, которые формируются из данных предметной области.

Д. С. Целуйко в своем научном труде [52] рассматривает понятие базы данных, указывая что это структурированных набор данных, который описывает характеристики физических или виртуальных систем. Концептуальную модель данных ученый рассматривает как модель данных, которая состоит из перечня взаимосвязанных понятий, которые используются для описания предметной области.

Ю. И. Рогозов [41] обозначает, что база данных является хранилищем информации информационной системы и ее целью является сбор, систематизация, сохранение и актуализация информации о какой-либо предметной области, с целью обеспечения пользователя необходимой информацией, для принятия последним решения или получения знаний. По своей сути база данных – совокупность взаимосвязанных данных, имеющих определенную структуру для отображения реальных отношений между объектами предметной области.

Р. А. Жуков [10] указывает, что конкретные способы и средства размещения данных, описанных в логической модели, в физической среде хранения, определяют построение внутренней, физической модели организации баз данных. При этом физическая модель должна отвечать следующим требованиям:

- сохранение смыслового содержания логической модели;
- максимальная экономия внешней памяти;
- минимизация затрат по управлению данными;
- максимальное быстродействие при поиске и при обработке запросов.

Создаются базы данных с помощью множества средств проектирования, которые позволяют создавать не только логические модели, но и генерировать с этих моделей SQL-код и взаимодействовать с СУБД, в результате чего осуществляется создание физических моделей баз данных на выбранных серверах баз данных.

Предварительным этапом разработки и реализации базы данных является описание модели данных [10]. Модель данных представляет собой некоторую абстракцию, которая, будучи применима к конкретным данным, позволяет пользователям и разработчикам трактовать их уже как информацию. То есть это уже сведения, содержащие не только данные, но и взаимосвязь между ними. На практике используют три основных модели описания данных:

1. Иерархическая модель данных.
2. Реляционная модель данных.
3. Сетевая модель данных.

Г. А. Мирошниченко [27] указывает, что наиболее распространенной и удобной моделью является реляционная, на основании которые создается преимущественное большинство информационных систем. Реляционная база данных – база данных, основывающаяся на реляционной модели. Для работы с реляционными базами данных применяют реляционные системы управления базами данных.

Реляционные СУБД делятся на системы с открытым кодом (OpenSource) и с закрытым. Согласно исследованиям компании IDC, компании из Европы чаще отдают предпочтении открытому программному обеспечению (ПО). Делают это они из–за повышения качества и гибкости данного ПО.

Система управления базами данных представляет собой программное обеспечение, с помощью которого пользователи могут создавать, определять и поддерживать базу данных. Особенность СУБД с открытым кодом является то, что

они имеют при себе исходный код, который можно изменить, оптимизировать, ускорить за счет какого – либо алгоритма.

СУБД является основным компонентом информационной системы. Конструирование баз данных, способных осуществлять накопление и обработку информации, оказывает определяющее воздействие на эффективное функционирование информационных систем, трудозатраты при ее проектировании, разработке, эксплуатации и сопровождении. Очень большое количество методов конструирования баз данных ориентированы на узкоспециализированные нужды, т. е. обеспечивают гибкость информационной системы только в определенной предметной области. Примерами таких методов конструирования баз данных являются разработки Д. Ю. Кондракова и Р. В. Игнатовича [16] (объектно-ориентированная модель учётных баз данных), М. В. Шпакова [59] (настраиваемая объектная модель предметной области геоинформационных систем), С Ю. Завозкина [12] (автоматизация документооборота).

Существует ряд методологий создания баз данных, не зависящих от предметной области, что позволяет повышать гибкость информационных систем. Одной из таких методологий является модель А. Тенцера [46], которая выделяет принципы для построения баз данных, которые на зависят от предметной области:

1. Каждая сущность, информация о которой хранится в БД — это объект.
2. Каждый объект уникален в пределах БД и имеет уникальный идентификатор.
3. Объект имеет свойства (строковые, числовые, временные, перечислимые), которые описывают атрибуты сущности.
4. Объекты могут быть связаны между собой произвольным образом. Связь характеризуется связанными объектами и типом связи. Связь в некотором роде аналогична понятию ссылки на таблицу-справочник в традиционной модели БД.

5. Объект может быть хранилищем. В этом случае допускается хранение на нем других объектов.

По мнению Д. С. Целуйко [52], такая БД не привязана ни к какой бизнес модели и позволяет реализовать «над собой» практически любую бизнес-логику. Логика выделяется в отдельный программный слой и реализуется на сервере приложений. Модель Тенцера имеет возможность описать произвольный набор свойств любой сущности, данные о которой хранятся в БД и отобразить её на единообразно хранимые объекты. В структуре данных Тенцера не предусмотрено отдельных полей для хранения ссылок на другие таблицы. Вместо них в БД вводятся дополнительные таблицы. К недостаткам данной модели можно отнести низкий уровень гибкости разработки и управления, отсутствие поддержки разграничения прав доступа пользователей к данным.

1.3 Концепция гибких автоматизированных информационных систем

В процессе развития и совершенствования информационных систем и технологий их создания было разработано множество способов и технологий создания информационных систем, которые в своей основе имели разнообразные идеи, концепции и цели. По мнению автора работы, отдельным и очень существенным направлением в развитии информационных систем являются гибкие информационные системы, которые посредством возможностей изменения концептуальной модели базы данных могут приобретать разные конфигурации и за счет этого расширять свой функционал и возможности конечного назначения. Для понимания особенностей гибких информационных систем необходимо рассмотреть концептуальные основы их построения и функционирования, которые приведены ниже:

– гибкая информационная система строится на базе множества групп однородных моделей, с предусмотренной возможностью обеспечения их соединения. Каждый включенный в систему модуль может

настраиваться на выполнение определенных и включенных в его состав функций по автоматизации рабочего места. При необходимости модуль может быть перенастроен на выполнение функционала, потребность в котором возникла в процессе функционирования информационной системы.

- гибкая автоматизированная информационная система может изменять свою функциональную специализацию в отличии от традиционных информационных систем, в которых специализация фиксирована;
- в отличии от традиционных информационных систем, в которых первичным является функционал модулей, которые выполняют обработку определенных блоков информации, информационные системы с настраиваемой концептуальной моделью данных первичными являются цели и задачи, которые должна выполнять система и технологическая сторона реализации здесь вторична.

Таким образом, можно выделить концептуальные особенности адаптивных автоматизированных информационных систем:

- система разрабатывается на базе гибких универсальных модулей, которые осуществляют обработку блоков информации;
- в модулях системы предусмотрен набор инструментов, с помощью которых можно описать объекты предметной области, их свойства и взаимосвязи между ними, а также алгоритмы обработки информационных блоков;
- в зависимости от переопределённых целей и задач модули гибкой информационной системы специализируются по определенном инструментальном признаку, обеспечивая таким образом возможности адаптации системы для функций учета и функций проведения сложных расчетов;

- гибкая автоматизированная информационная система может работать по принципу групповой и унифицированной обработки данных, результатом чего является то, что в рамках одного рабочего места посредством гибких унифицированных модулей реализуются группы различных операций, которые являются технологически однородными;
- в гибкой информационной системе информация обрабатывается централизовано.

Глава 2 АНАЛИЗ ТЕХНОЛОГИЙ РАЗРАБОТКИ ИНФОРМАЦИОННЫХ СИСТЕМ С НАСТРАИВАЕМОЙ КОНЦЕПТУАЛЬНОЙ МОДЕЛЬЮ БАЗЫ ДАННЫХ

2.1 Технология создания гибкой информационной системы

Как было указано в п. 1.1, гибкая информационная система является системой, которая включает возможность изменения концептуальной модели базы данных, что позволяет абстрагировать предметную область от процесса проектирования информационной системы. Рассмотрим особенности создания гибких информационных систем. На начальном этапе создания гибкой информационной системы формируется технологическая основа – выбирается платформа разработки. После этого формируется база данных системы. Для реализации предложенной концепции база данных должна реализовывать функции хранения метамоделей, а часть программных компонентов – обеспечивать доступ и работу с метамоделями, хранящимися в базе данных.

Требованиями к базе данных в такой системе будут:

- хранение метамодели, модели и пользовательских данных;
- отсутствие ограничений по структуре хранимых данных;
- отсутствие априорной информации о структуре хранимых данных;
- возможность формального описания и манипуляции данными.

Вторым этапом является внесение в базу данных метамодели предметной области, формируемой экспертами предметной области и помещаемой в хранилище метамоделей. Сочетание фреймворка и метамодели предметной области образует среду разработки информационной системы для заданной предметной области.

Третий этап заключается в заполнении данными о конкретных бизнес-процессах, автоматизируемых в создаваемой информационной системе. По окончании третьего этапа получается готовое решение – информационная система, которая сконфигурирована под автоматизацию бизнес-процессов пользователя.

Четвёртый этап заключается в наполнении информационной системы рабочими данными, что соответствует этапу эксплуатации в классической модели жизненного цикла информационной системы.

Для различных технологических требований формируются разные метамодели и инструментальные среды [17-19], реализующие их. Также при использовании одной и той же метамодели могут быть описаны и реализованы разные решения ИС, что определяется разными моделями бизнес-процессов, вносимыми пользователями на этапе конфигурирования.

Концепция предлагаемого подхода заключается в использовании принципа метапроектирования, реализованного на базе совокупности моделей специального вида (метамодель предметной области + метамодель ИС –> модель бизнес-процессов + модель ИС –> готовое решение»). На технологическом уровне данная формула отражается последовательностью «платформа -> фреймворк -> среда разработки -> готовое решение».

Подобная концепция позволяет максимальным образом включить экспертов и конечных пользователей в процесс разработки, устраняя разрыв между постановкой задачи и ее решением.

При реализации возможности конфигурирования в информационной системе важно определить объекты структуры информационной системы, нуждающиеся в конфигурировании, т.е. определить в какие компоненты ИС необходимо внести точки изменчивости. В зависимости от класса, к которому относится конфигурируемая система, перечень таких объектов может меняться. Если использовать трехуровневое представление ИС в виде слоев пользовательского интерфейса, бизнес-логики и данных [21], то наиболее полно объекты всех трех уровней позволяют конфигурировать ИС с изменяемой структурой.

При создании конфигурируемых информационных систем существует задача обеспечения гибкости баз данных. Она, в частности, решается с помощью

структурно-независимых баз данных (СНБД) [22, 23, 24]. Таковые базы данных обладают динамическими структурами.

В качестве исходного материала динамические базы данных предоставляют:

1. Множество характеристик, в терминах которых описывается структура данных. В структурно-независимых базах данных таковыми являются: сущность, связь, атрибут, тип данных.
2. Ограничения на связи. Таковыми ограничениями являются: запрет на связывание сущностей без указания связующего атрибута, невозможность присвоения типа данных сущности и т. д.
3. Механизмы реализации указанных ограничений.

Существующие на сегодняшний день информационные системы (ИС) в своей основе имеют модель, состоящую из трех ключевых аспектов: данных, интерфейса и бизнес-логики [25]. Находясь на одном уровне абстракции эти аспекты создают полную модель информационной системы, позволяющую при ее реализации решать отдельные локальные задачи предприятий и конкретных пользователей.

Необходимость создания ИС, которые без дополнительных материальных и временных затрат имеют возможность учитывать меняющиеся требования пользователя и динамику предметной области, дала возможность для развития идеи разработки ИС, основанных на метамодели [25]. Метамодель представляет собой модель предметной области [26], описанную на нескольких уровнях абстракции, где каждый высший уровень полностью, целостно и непротиворечиво задает структуру данных, функциональность, отображение и связи низших уровней. Метамодель является моделью модели, которая задает правила, ограничения, закономерности и элементы для построения моделей [26]. Также важно, чтобы информационная система была основана на трех метамоделях – метамодели процесса создания данных, метамодели процесса создания бизнес-логики и метамодели процесса создания интерфейса.

Для создания таких информационных систем необходимо на первом этапе построить и описать метамодели трех существующих аспектов, состоящие из элементов и связей между ними.

Далее необходимо описать метамодели процесса создания данных, как неотъемлемой части ИС, ориентированных на данные.

Модель данных в существующих информационных системах представляется в виде базы данных (БД). Основными элементами модели (классическая модель «сущность-связь») являются сущность, связь и атрибут. Эти элементы являются необходимой базой для создания не только модели, но и метамодели данных. Уже само понятие модели данных подразумевает наличие уровня абстракции [28].

Метамодель данных позволяет агрегировать изменяемую во времени информацию по объекту [29], модифицировать структуры хранения данных без перепроектирования и внесения изменений в программный код системы [26].

Метамодель, находясь на одну степень абстракции выше, позволяет получить необходимую «свободу» от навязываемых решением конкретных задач экземпляров сущностей, атрибутов и связей и придает необходимую для реализации конфигурируемых информационных систем динамику.

Такую «свободу» метамодели данных придает структурно-независимая база данных, которая позволяет работать со слабоструктурированными данными, вносить изменения в логическую структуру (добавлять сущности, атрибуты) без изменения физической [16]. Статическая структура базы данных позволяет хранить сущности и их атрибуты в виде справочников, а связи «сущность-сущность», используемые в ER-диаграммах, заменяются связями «сущность-атрибуты» для придания необходимой гибкости.

Перед тем как описать основные составляющие метамодели процесса создания данных как основы для создания структуры БД следует отметить, что как и проектирование, описание системы можно проводить на разных уровнях, в зависимости от точки отсчета, особенностей предметной области, конкретных

условий, а также прогноза и анализа будущих условий, ресурсов, времени, мотивации и т.д., возможно рассматривать систему как на более высоком уровне абстракции, так и на уровне ниже. Такое детальное изучение и описание элементов послужит этапом в решении задач на другом, качественно новом, высоком уровне.

Итак, метамодель можно представить состоящую из:

1. статических элементов (элементы, которые не могут быть изменены без перехода на более высокий уровень);
2. динамических элементов (элементы, которые изменяются на данном уровне с помощью функций управления данного уровня);
3. функций управления (действия, которые необходимо совершить над динамическими элементами);
4. инструмента управления (средство, с помощью которого реализуются функции управления).
5. методики управления (определение того, как именно, в какой последовательности относительно необходимого нам результата, следует выполнять функции над динамическими элементами и с помощью каких инструментов).

Для каждого уровня абстракции у системы нужны свои статические и динамические элементы, функции, инструменты и методика.

Относительно приведенных выше составляющих метамодели опишем метамодель процесса создания данных. Целевым результатом в такой метамодели, который планируется получить на данном (низком) уровне абстракции, является получение структуры БД, которая для следующего уровня будет являться статической относительно сформулированного целевого результата. Статическими элементами в такой метамодели, как и в модели «сущность-связь» являются понятие сущности, понятие атрибута, понятие связи. Динамическими элементами являются конкретные экземпляры сущности, экземпляры атрибутов, экземпляры связи. Функциями управления динамическими элементами являются:

1. на уровне сущностей (создание сущности с ее названием, синонимом (уникальное имя сущности, которое будет храниться в базе данных и использоваться для обращения к сущности через скрипт), изменение названия сущности, удаление экземпляров сущности;

2. на уровне атрибутов (создание атрибута, определение его названия, синонима и типа данных, изменение атрибута, его названия и типа данных, удаление экземпляра атрибута);

3. на уровне связей (создание связи по экземпляру сущности и экземпляру атрибута, удаление экземпляра связи).

Инструментами для реализации указанных функций управления на данном уровне являются:

1. на уровне сущностей – редактор справочника сущностей;

2. на уровне атрибутов – редактор справочника атрибутов;

3. на уровне связей – редактор сущностей.

Методика управления в описываемой метамодели процесса создания данных представляет собой перевод модели «сущность-связь» в структуру БД и состоит из этапов, которые условно можно разделить на внесение сущности, внесение атрибутов и установление связи.

I. Внесение сущности.

1. Добавить сущность в базу данных с помощью редактора справочника сущностей.
2. Заполнить для вносимой сущности такие строки, как наименование, описание, синоним. После внесения всех необходимых параметров сущность будет добавлена в базу данных и в справочник сущностей.

II. Внесение атрибутов.

1. Внести атрибуты сущностей с помощью редактора справочника атрибутов.

2. Заполнить для вносимых атрибутов такие строки, как наименование, описание и синоним.
3. Указать тип атрибута путем выбора его из списка: число, строка, дата, список, значение списка, логический атрибут, ссылка на сущность, большой текст, двоичный файл.
4. Сгруппировать внесенные атрибуты по типам сущностей.
5. Добавить все необходимые атрибуты.

III. Установление связи.

1. Добавить атрибуты к сущности отметив необходимые из них соответствующим образом.
2. Сформировать список выбранных атрибутов.
3. Посредством редактора сущностей привязать атрибуты к конкретной сущности (установить связь полученного набора сущностей и атрибутов).
4. Сохранить внесенные изменения после добавления всех необходимых атрибутов.

Представим приведенное выше описание элементов метамодели процесса создания данных (аспект – данные) в виде логической схемы.

На рис. 8 представлены базовые элементы метамодели на примере метамодели процесса создания данных, где:

– на основе существующих статических элементов (сущность, атрибут и связь) посредством определенных функций создаются конкретные динамические элементы в виде экземпляров сущностей, экземпляров атрибутов и экземпляров связей;
– функции по созданию, изменению и удалению динамических элементов выполняются при помощи определенных инструментов (редактора справочника сущностей, редактора справочника атрибутов, редактора сущностей);

– методика управления задает определенную последовательность в выполнении функций над динамическими элементами в рамках рассматриваемого уровня абстракции.

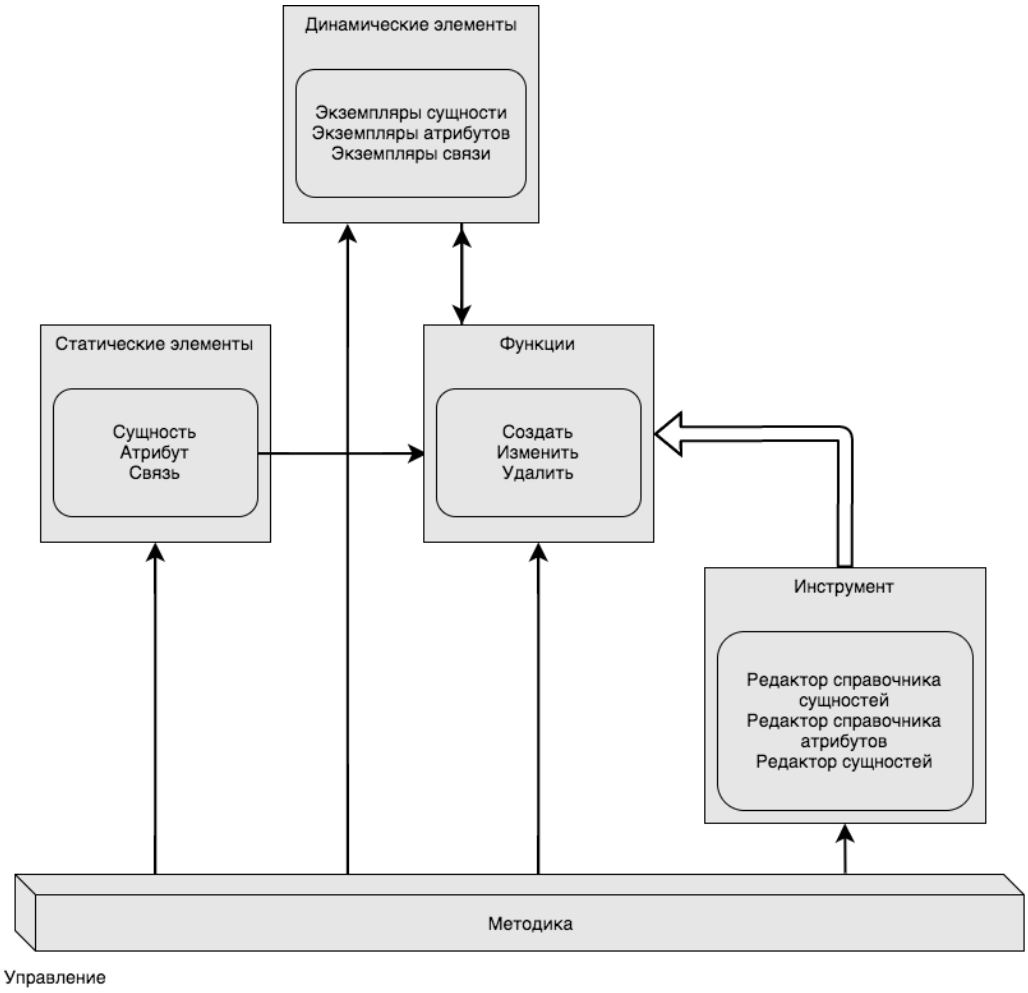

Рисунок 7 – Метамодель процесса создания данных для конфигурируемых информационных систем

Полученная таким образом метамодель процесса создания данных будет являться базовым статическим элементом для следующего уровня абстракции, на котором присутствуют свои функции, динамические элементы, инструменты и методика, соответствующая этому уровню.

Приведенная метамодель процесса создания данных представлена в соответствие с идеями синтетического подхода, где система уже не

рассматривается как целое, состоящее из частей. Она сама может являться частью метасистемы и ее состав, и структура описываются относительно этой метасистемы, т.е. относительно целого, что и показано на примере метамодели процесса создания данных как части целой метасистемы, в которой она будет являться статической и наряду с метамоделями процессов создания других аспектов составлять полную метамодель, позволяющую динамически конфигурировать информационные системы, ориентированные на данные.

При построении структурно-независимых баз данных используются следующие принципы:

1. Хранение данных по столбцам (в отличие от традиционного хранения по строкам в реляционной модели).
2. Явное хранение метаданных.

Оба принципа были заложены при разработке модели данных EAV (Entity–attribute–value model). Модель данных EAV на сегодняшний день эффективно используется в условиях неопределенности относительно структуры хранимых данных, например в электронной коммерции платформой Magento Commerce [30]. Особенностью модели является представление данных в виде троек "сущность–атрибут–значение".

Базы данных, создаваемые с учетом обозначенных выше принципов, можно считать динамическими. Они обладают физической структурой данных, которая не зависит от структуры и состава (динамики) хранимой информации, и сохраняют работоспособность при любых ее изменениях. Обозначить такие базы данных можно понятием «структурно-независимые базы данных» (СНБД).

Рисунок 9 иллюстрирует создание и использование структурно-независимых баз данных. На первом этапе создается СНБД, которая даже будучи абсолютно пустой может быть сопряжена с информационной системой за счет статического физического уровня и фиксации логики управления данными.

На втором этапе производится концептуальное проектирование предметной области, на третьем – логическое. Логическая модель через заранее определенные средства (термины описания) отображается в СНБД и является метаданными, хранимыми в явном виде.

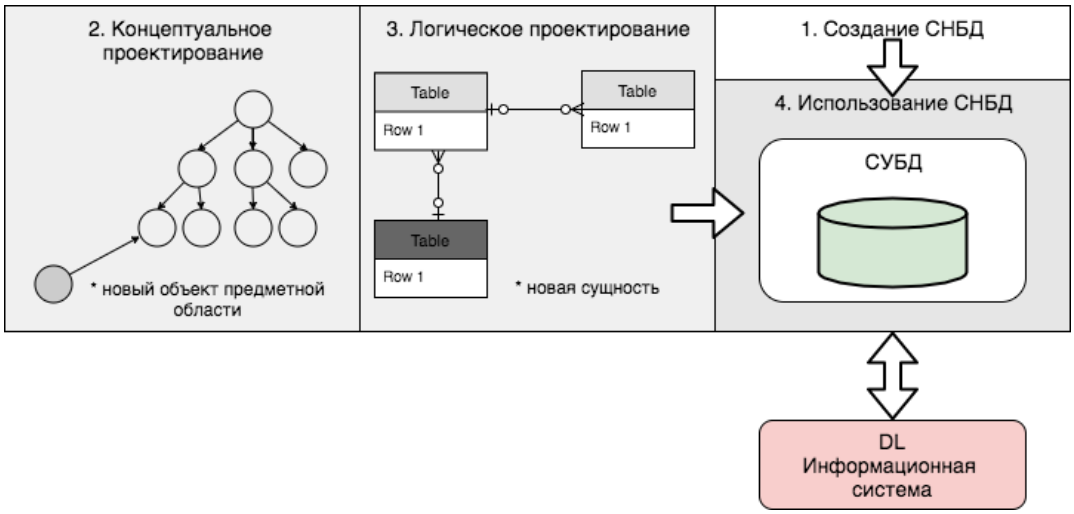

Рисунок 8 - Создание и использование СНБД

Метаданные, заданные в явном виде через логическую модель, позволяют пользоваться СНБД как прикладной реляционной БД. При этом логика управления данными остается неизменной, что не нарушает работоспособности информационной системы.

Добавление нового объекта предметной области отражается на логической модели данных, но не затрагивает СНБД и средства ее взаимодействия с ИС.

Динамические базы данных должны обеспечивать изменение пользовательских метаданных. Эти метаданные могут принимать произвольную структуру в зависимости от конфигурации. Чтобы обеспечить динамическое изменение конфигурации метаданные должны содержаться внутри БД, а не на системном уровне СУБД как это принято в классическом подходе.

Добиться такого можно вводом нового мета-уровня (рис. 9). Для СНБД существует два мета–уровня:

1. Мета-уровень. Метаданные – средство представления знаний о данных БД. Они являются описательными и хранятся аналогичным образом с данными.
2. Мета-метауровень. Мета-метаданные – средство описания различных метаданных. Являются декларативным инструментом, содержащим средства для описания произвольных метаданных.

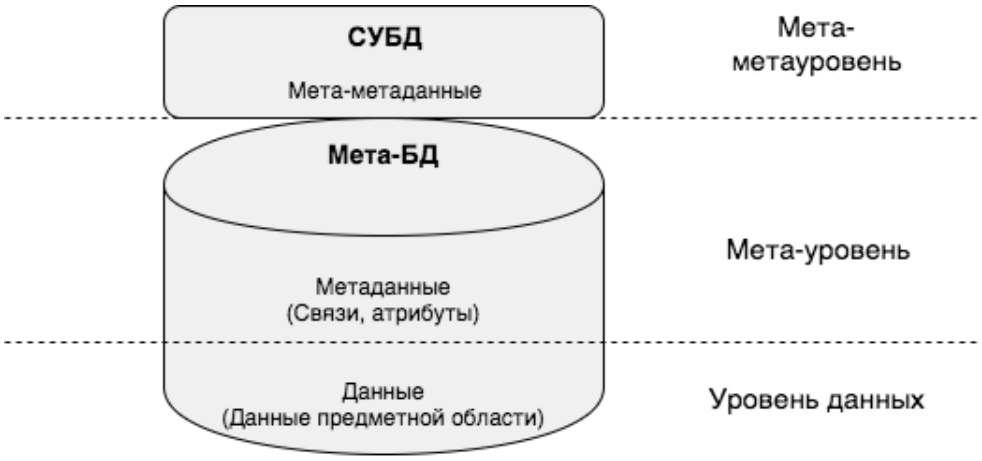

Рисунок 9 - Концепция двух метауровней данных в СНБД

Таким образом, первый вид исходного материала (элементы) содержатся на мета-метауровне. Путем практических и теоретических исследований было установлено, что минимальным и достаточным набором элементов являются: сущность, атрибут, тип данных, типовые связи [32].

Обратимся к типизированным связям, чтобы описать второй вид исходного материала. При разработке структуры СНБД закладываются ограничения на связывание элементов – так называемые типовые связи.

Типовая связь является ограничением на связывание элементов и закладывается в правила, реализуемые механизмом конфигурирования динамических баз данных [35].

На уровне реализации СНБД каждый элемент представляется одной реляционной таблицей. Соответственно механизм реализации ограничений (третий

вид исходного материала) представляет собой создание и поддержание связей между реляционными таблицами (табл. 2).

Таблица 2 – Механизм реализации типовых связей

Элемент	Функция	Инструмент	Результат
Э1. Таблица	Ф1. Установить связь	И1. DDL	Р1. Типовая связь

В соответствии с принятой формулировкой, механизм реализации типовых связей (ограничений на связи) звучит следующим образом: результат Р1 (типовая связь) получен из двух экземпляров элемента Э1 (Таблица) путем выполнения функции Ф1 (установить связь) с помощью инструмента И1 (DDL подмножество языка SQL).

Обобщив вышесказанное, представим концептуальную модель СНБД (рис. 10).

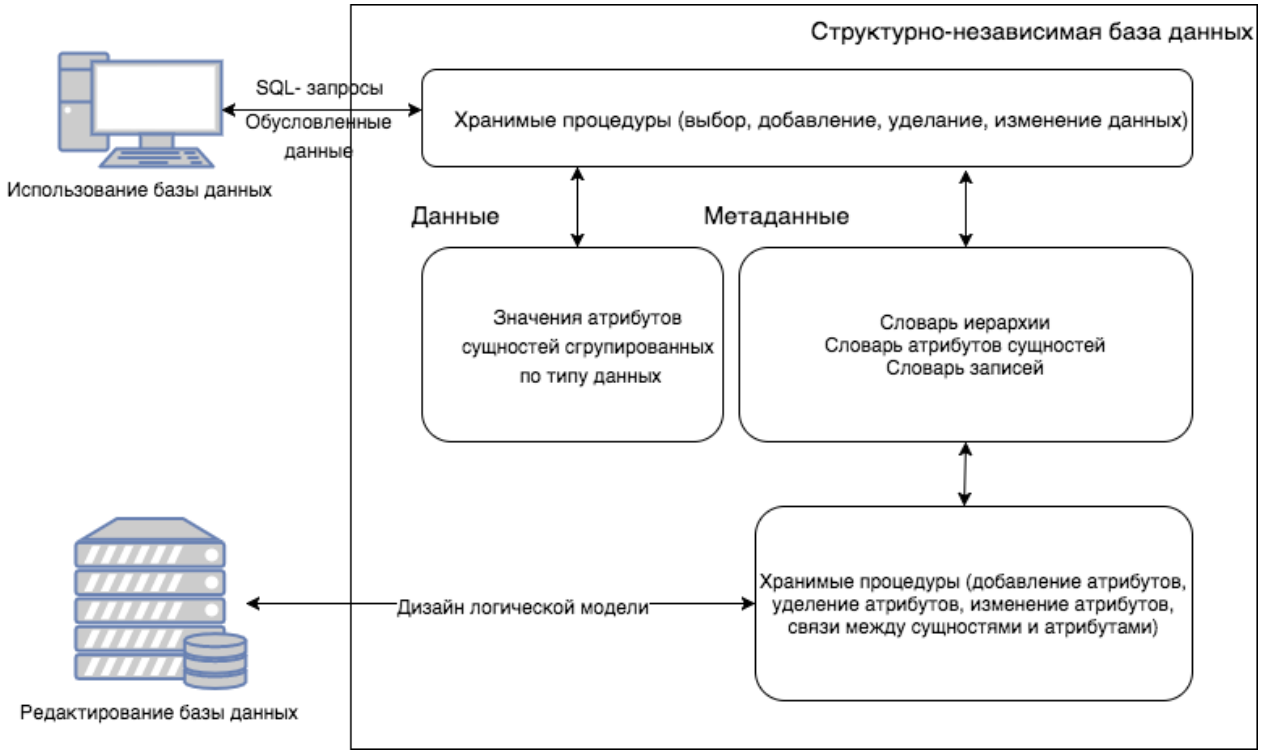

Рисунок 10 - Концептуальная модель структурно-независимой базы данны

2.2 Обоснование выбора языка программирования для создания адаптивной информационной системы

Проектируемая система является клиент-серверным приложением, для построения которого необходим серверный язык программирования. Существует множество серверных языков, которые можно использовать для программирования веб-приложений, наиболее подходящими для реализации информационной системы являются PHP, Ruby, Python. Проведем анализ этих языков программирования (таблица 3).

Таблица 3 – Сравнение языков программирования

	PHP	**Ruby**	**Python**
Предназначение	Создание динамических веб-приложений	Создание веб-приложений и компьютерах программ	Создание динамических веб-приложений и компьютерных программ
Год создания	1995	1995	1991
Разработан под влиянием	C, PERL, JAVA, C++, TCL	ADA, C++, CLU, DYLAN, EIFFEL, LISP, PERL, PYTHON	ABC, ALGOL 68, C, C++, ICON, JAVA, LISP, PERL
Сложность освоения	6	4	5
ООП	Да	Да	Да
Инструкция break	Да	Да	Да
Многомерные массивы	Да	Да	Да
Цикл foreach	Да	Да	Да
Множественное наследование	Нет	Нет	Да
Макросы	Нет	+/-	Нет
Именованные параметры	Нет	Да	Да

	PHP	Ruby	Python
Наличие библиотек для работы с графикой и мультимедиа	Да	Да	Да
Работа с MySQL	Да	Да	Да
Наличие удобных и доступных средств разработки	Да (NetBeans)	Нет	Да (Visual Studio)
Поддержка популярными веб-серверами	Да (по умолчанию)	Нет (дополнительные службы)	Нет (дополнительные службы)

Как видно из таблицы, рассмотренные языки программирования сходятся по рядку показателях, но PHP имеет преимущества в направлении доступности средств разработки и доступности на популярных веб-серверах, в связи с этим, в качестве языка программирования для построения информационной системы будет использован PHP. PHP — язык программирования, работающий на стороне сервера и представляющий своеобразное дополнение к нему. Этот язык позволяет не просто отправлять браузерам точные копии запрашиваемых файлов, но и запускать небольшие программки для выполнения разных задач — PHP-скрипты.

Код PHP интерпретируется веб-сервером и генерирует HTML-код или другой вывод, который отсылается браузеру пользователя. Схема работы веб-сервера представлена на рис. 11.

PHP используется для разработки веб-сайтов различного направления. PHP-программа представляет собой текстовый файл, обычно с расширением – «.php». В файле хранится последовательность команд (инструкций), которые выполняет веб-сервер. PHP читает файл построчно сверху вниз и выполняет записанные команды.

PHP исполняет код, находящийся внутри таких ограничителей, как <?php и ?>. Имена переменных начинаются с символа $, тип переменной объявлять не

нужно. PHP интерпретирует переход на новую строку в качестве пробела. Инструкции разделяются с помощью точки с запятой (;), за исключением некоторых случаев.

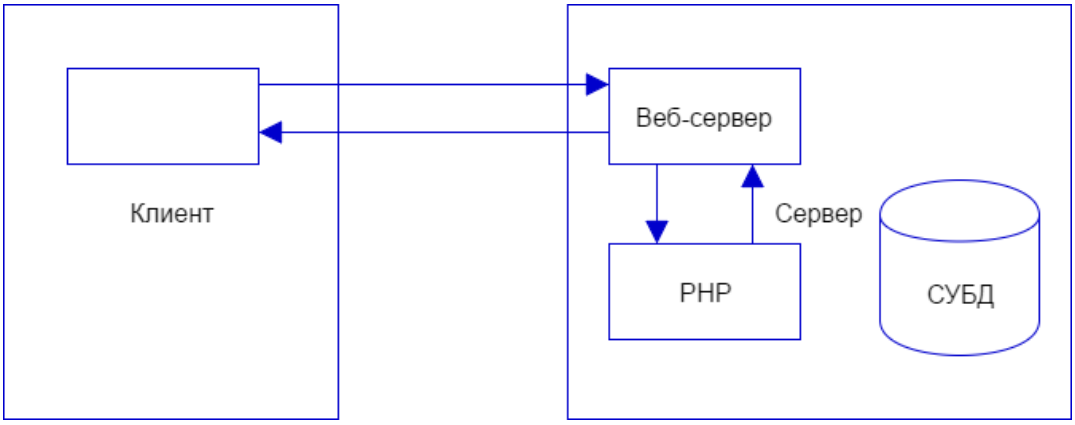

Рисунок 11 - Архитектура веб-приложений, реализуемых на языке PHP

PHP поддерживает три типа комментариев: в стиле языка JavaScript: ограниченные /* */, начинающиеся с // и идущие до конца строки и оболочки UNIX, начинающиеся с # до конца строки.

PHP позволяет включать в текущий документ данные из внешних файлов. Содержащиеся в них PHP-сценарии также будут исполнены, если это допустимо. Для данной задачи используются следующие функции:

- include(«имя_файла») - текст внешнего файла помещается в место вызова функции, после чего сразу исполняется. В случае помещения в условный оператор выполняется только при требуемом условии.
- require(«имя_файла») - текст внешнего файла подключается заранее и определенные в нем функции доступны в любом месте основного сценария. Не используется для вставки текстовых данных.

Работа с файлами может быть обозначена как одна из основных возможностей PHP, дающая возможность хранения данных между вызовами скриптов.

В PHP-скрипте есть несколько способов получения доступа к сведениям, которые клиент передал по протоколу http.

Чтобы обратиться к переменным, переданным при помощи HTTP-запросов, применяется специальный массив - $_REQUEST. В данном массиве содержатся данные, которые переданы методами POST и GET, а также при помощи HTTP cookies. Является суперглобальным ассоциативным массивом, т.е. его значения могут быть получены в любом месте программы, применяя как ключ имя соответствующей переменной (элемент формы).

После того, как форма отправлена, в вызываемом скрипте появится возможность использования переданного значения в HTML-коде так: echo $_REQUEST[«name»].

При понимании полей формы, возможно формирование условий вызова функций отображения и обработки сведений формы.

При отправке формы элемент массива $_REQUEST[«stage»] получает значение results и вызывается функция-обработчик формы. Если же сведения в скрипт не были переданы (первая загрузка), то осуществляется функция вывода формы.

Функция зависимо от осуществленного пользователем выбора создает, а также выводит строку-сообщение.

Если все вышеуказанные функции поместить в один файл, то подготовится работоспособный скрипт.

Кроме суперглобального массива $_REQUEST в php есть еще также несколько суперглобальных ассоциативных массивов, дающих возможность обработки передаваемых клиентом данных:

- $_GET[] - содержит все значения, которые передаются в сценарий при помощи метода формы GET.
- $_POST[] - содержит все значения, которые передаются в сценарий при помощи метода формы POST.

$_SERVER[] - содержит все значения, которые получаются от сервера.

2.3 Обоснование выбора СУБД для создания «гибкой» информационной системы

Как СУБД для проекта выбрана MySQL. Это реляционная БД, дающая с высокой степени удобства и надежности возможность хранения информации. Реляционная БД является базой данных, основывающейся на реляционной модели. Само слово «реляционный» произошло от английского слова «relation» (отношение). В целях работы с реляционными БД используются реляционные СУБД [8].

Реляционная база данных – база данных, основывающаяся на реляционной модели. Слово «реляционный» происходит от английского слова «relation» (отношение). Для работы с реляционными базами данных применяют реляционные системы управления базами данных. Использование реляционных баз данных было предложено в 1970 году доктором Коддом, работающим в компании IBM [8].

В реляционной модели данные организуются в форме двумерных таблиц. У каждой реляционной таблицы, являющейся двумерным массивом, должны быть такие свойства [3]:

- каждый табличный элемент является только одним элементом данных;
- каждая ячейка в столбце таблицы однородна – все его элементы должны обладать одинаковым типом (INT, FLOAT, VARCHAR и т. д.);
- у каждого столбца есть уникальное имя, которое идентифицирует этот столбец;
- таблица не может содержать одинаковых строк;
- порядок следования столбцов и строк является произвольным.

Почти каждая система управления реляционными БД обеспечивает работу с языком SQL, с помощью которого возможно определение и модификация структуры данных, добавление, изменение и удаление данных, а также осуществление самых разнообразных выборок данных.

Более распространены серверы реляционных БД, основанные на клиент-серверной архитектуре. Данными серверами обеспечивается устойчивая работа с базами данных одновременно большого количества клиентов (ими могут быть десятки, сотни, тысячи и миллионы клиентов – все находится в зависимости от применяемого оборудования и ПО).

Реляционные БД строятся на строгой теории реляционных БД, основывающейся, в свою очередь, на теории отношений и теории множеств. Реляционная БД является набором таблиц, между которыми задаются связи. Табличные строки называются записями, а элементы, из которых запись состоит — полями. В теории реляционных БД таблицы называются отношениями, поля — атрибутами, а записи — кортежами.

В таблице реляционной БД не может быть повторяющихся записей (строк). Данное требование пришло из теории множеств. Минимальный набор полей, дающий возможность отличить запись от любых других записей, называется ключом. Все показатели ключа в рамках одной таблицы должны являться уникальными. Каждая таблица должна обладать как минимум одним ключом, что прямо вытекает из того, что таблица не может иметь повторяющихся записей. Ключи таблицы могут включать одно поле – данные ключи называются атомарными либо простыми ключами. Ключи могут включать несколько полей — составные ключи. Таблица БД может обладать одним ключом или несколькими ключами. Один из ключей назначается как первичный ключ, а остальные называются потенциальными (в теории реляционных БД) либо альтернативными (в определенных реализациях некоторых БД) [8].

Часто используются суррогатные первичные ключи, являющиеся ключами, состоящими из поля (либо полей), не несущих данных из предметной области, а служащими как замена (суррогат) для натурального (естественного) первичного ключа. Как суррогатные ключи чаще всего применяются счетчики (последовательности, генераторы) autoincrement либо же глобально–уникальные

идентификаторы (GUID). Правильно проработанная структура БД должна удовлетворять специальным правилам, основывающимся на теории отношений и называющимся нормальными формами [13].

Понятие нормальной формы ввел Эдгар Кодд в процессе создания реляционной модели БД. Нормализация является процессом преобразования базы данных, к виду, соответствующему нормальным формам и обеспечивающему наименьшую избыточность. Цель нормализации – защитить базу данных от логических и структурных проблем, выступающих как аномалии данных. Например, если имеется ряд одинаковых записей в таблице, то в данном случае существует риск нарушения целостности данных при дальнейшем обновлении таблицы. Прошедшая нормализацию таблица меньше подвержена подобным проблемам, так как структура данной таблицы определяет связи между данными, соответственно, исключается необходимость в существовании записей с повторяющимися данными. Избыточность может быть устранена с помощью разбиения отношений (таблиц) так, чтобы в каждом отношении осуществлялось хранение только первичных фактов (факты, не выводимые из иных хранимых фактов). Следовательно, нормализация не ставит целью уменьшить или увеличить производительность работы или же уменьшить, или увеличить объём базы данных. Конечная цель нормализации – уменьшить потенциальную противоречивость хранимых в БД сведений. Нормализация применима к таблице, являющейся правильным отношением.

Далее, более детально опишем СУБД MySQL, которая будет использована для создания гибкой информационной системы. MySQL является высокопроизводительной многопоточной и многопользовательской системой управления реляционными БД, основанной на архитектуре «клиент–сервер». За последнее время данная надежная, мощная и дружественная к пользователю система управления БД явилась стандартом для применения в личной и деловой сфере во многом за счет ее продвинутого набора инструментов для управления

данными, мягкой политики лицензирования, а также общемировой поддержки со стороны сообщества, в которую входят обычные пользователи и серьезные разработчики.

Популярность MySQL объясняет определенное сочетание уникальных особенностей, в число которых входят надежность, скорость, расширяемость, а также открытость исходного кода [7].

Для системы управления реляционными БД скорость является временем, затрачиваемым на исполнение запроса и возврат результата тому, кто его осуществляет, – это очень важный показатель. По любым меркам MySQL — это производительная система, чаще всего оказывающаяся на порядок быстрее, чем решения–конкуренты. Тесты производительности, которые доступны на сайте MySQL, указывает на то, что MySQL опережает практически все прочие доступные сейчас системы управления БД.

В некоторых случаях высокую производительность сопровождает низкая надежность. Тем не менее, MySQL, созданная в целях обеспечения максимальной надежности и оптимального времени работы, прошла сертификацию и тесты для применения в высокоуровневых критически важных приложениях. MySQL осуществляет поддержку транзакций, гарантирующих согласованность данных и сокращающих риск их потери, а также репликации и кластеризации, являющихся методиками, значительно уменьшающими время простоя при сбое сервера. И наконец, огромная аудитория пользователей MySQL оказывает содействие в быстром обнаружении и устранении ошибок, а также тестировании данного программного продукта в разных окружениях. Данный профилактический подход повлек то, что в MySQL фактически отсутствуют ошибки [7].

MySQL способна осуществлять обработку чрезвычайно больших и сложных баз данных с сохранением при этом достойного уровня производительности. Таблицы, имеющие объем в несколько Гб, состоящие из сотней тысяч записей —

это распространенное явление в информационных системах, которые как базу данных применяют MySQL.

Невзирая на то, что большую часть взаимодействия с MySQL выполняют посредством интерфейса командной строки, есть также несколько графических инструментов на браузерной либо другой базе, упрощающих исполнение задач по управлению и администрированию сервера баз данных MySQL.

В отличие от собственных платных аналогов, в которых требуется настройка буквально сотен параметров, система MySQL достаточно проста в настройке и оптимизации даже при самых требовательных приложениях. Если вести речь о коммерческом окружении, то тут в полной мере поддерживает профессиональное обучение, консультирование, а также техническое сопровождение MySQL.

MySQL осуществляет поддержку большинства важных требований стандарта ANSI SQL и нередко дает возможность дополнения его пользовательскими функциями, расширениями и типами данных, призванными улучшить переносимость, а также обеспечить для потребителей расширенную функциональность. MySQL поддерживают как операционные системы UNIX, так и отличные от них, в том числе Linux, FreeBSD, Solaris, OS/2, Windows 95-Vista, MacOS, и может функционировать на различных архитектурах, среди которых Intel x86, Alpha, SPARC, PowerPC и IA64.

Так как MySQL, это полноценная многопользовательская система, соответственно, большое число клиентов могут одновременно иметь доступ и пользоваться одной (или более) базой данных MySQL. Это очень важно в процессе разработки веб–приложений, которым нужно поддерживать одновременное подключение большого числа удаленных клиентов. MySQL включает также гибкую и мощную систему привилегий, дающую администраторам возможность защиты доступа к критическим данным, применяя комбинацию из схем по проверке подлинности хостов и пользователей [12].

Так как MySQL — это программное решение, которым пользуются миллионы людей во всем мире, она в полной мере поддерживает кодировку Юникод, а также самые существенные наборы символов (а также китайских и латинских). Наборы символов учитывают в процессе сортировки, сравнения и сохранения данных.

MySQL предполагает наличие интерфейсов к программированию приложений API на многих языках, что дает возможность создания управляемых баз данных приложения на многих языках программирования. Сейчас MySQL дает возможность работы с такими языками, как C, C++, Java, Eiffel, Perl, Python, PHP, Ruby и Tcl, также есть коннекторы для ODBC, JDBC– и .NET–приложений.

2.4 Обоснование выбора среды проектирования базы данных

На рынке программного обеспечения существует множество средств, которые автоматизируют моделирование баз данные, среди которых NaviCat, ToadModeler, MySQLWorkbench. Проведем анализ этих сред проектирования баз данных (таблица 4).

Таблица 4 – Сравнение сред проектирования баз данных

	MySQLWorkbench	ToadModeler	NaviCat
Предназначение	Проектирование баз данных	Проектирование баз данных	Проектирование баз данных
Работа с MySQL	Да	Да	Да
Визуальное проектирование	Да	Да	Да
Свободна к распространению	Да	Нет (демоверсия)	Нет (демоверсия)
Связь с СУБд	Да	Да	Да
Генерация SQL	Да	Нет	да
Сложность освоения	6	8	8
Кроссплатформенность	Да	Нет	Нет

Таким образом, на основании анализа сред проектирования баз данных определено, что для проектирования базы данных наиболее подходящей является среда MySQLWorkbench, которая предназначена для визуального проектирования баз данных и управления сервером MySQL. На рис. 12 представлено окно визуального редактора модели базы данных программы MySQLWorkbench.

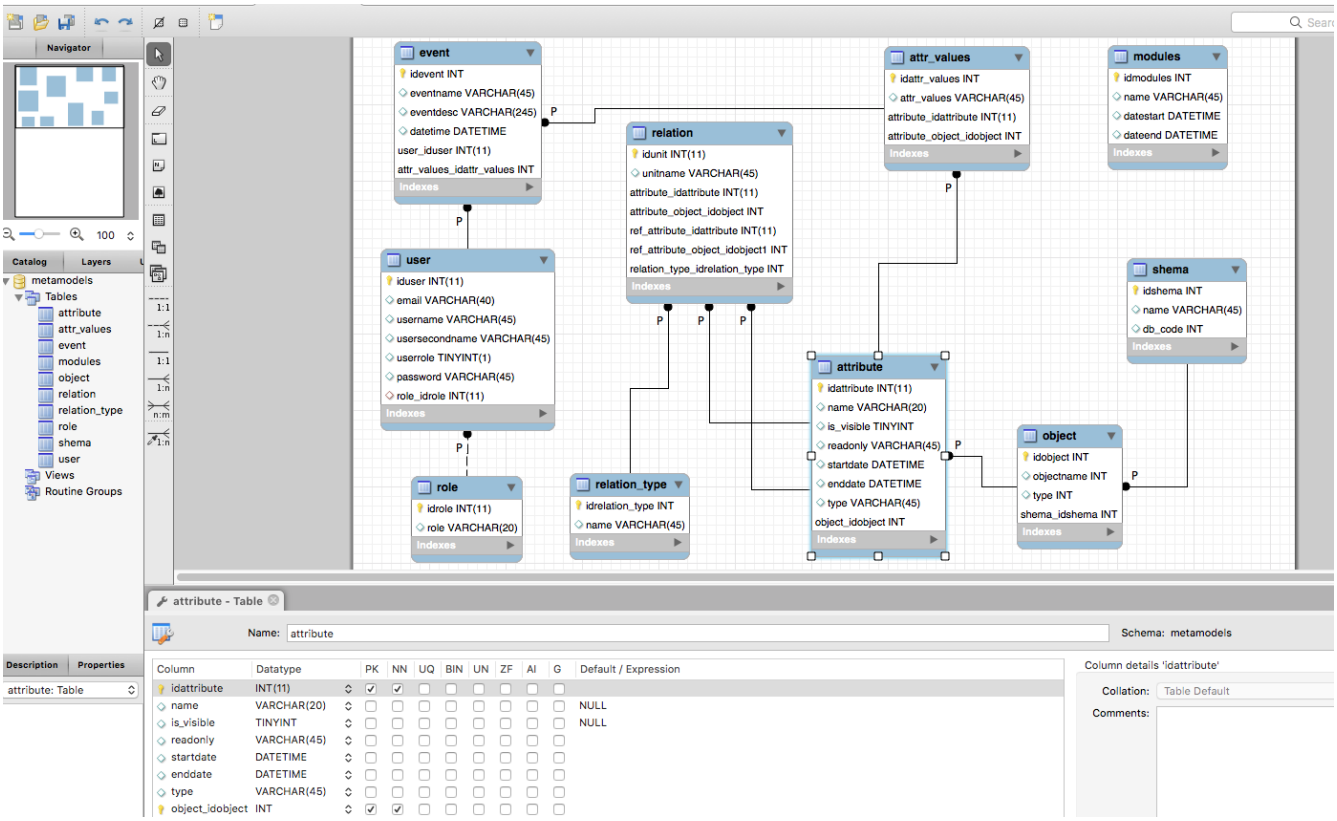

Рисунок 12 – Визуальный редактор модели базы данных

Программа предоставляет пользователю возможности в режиме визуального отображения создавать, редактировать и удалять таблицы и данные в этих таблицах, создавать связи между таблицами (идентифицирующие и не идентифицирующие связи, связи один к одному, один к многим, многие ко многим), задавать тип сохраняемых данных в полях таблиц. После создания модели базы данных с помощью визуального редактора можно посредством функции прямого инжиниринга сгенерировать SQL-код для созданной модели и выполнить его на MySQL-сервере, в результате чего будет создана физическая модель базы данных.

Кроме функции прямого инжиниринга, есть функция обратного инжиниринга, которая позволяет считывать модель базы данных с сервера MySQL и отображать ее посредством визуального редактора.

2.5 Анализ сред разработки для создания гибкой информационной системы

В целях создания приложений используется множество программных средств. Веб-ориентированные информационные системы создают при помощи текстовых редакторов, дающих возможность создания и редактирования разметки документов. Текстовых редакторов сейчас очень много. Но наиболее популярны и удобны такие: Brackets, Atom, Notepad++, SublimeText. Также с этой целью можно применять среды разработки: PHPStorm, Eclipse, Netbeans, Lobster. Для разработки адаптивной информационной системы выбрана профессиональная среда разработки PHPStorm.

Данная программа является интегрированной средой разработки проектов на PHP с интеллектуальным редактором, поддерживающей PHP 7.0 и другие его версии. Программа обеспечивает автоматическое дополнение кода, рефакторинги, а также предотвращение ошибок и поддерживающей смешивание языков.

Главные возможности программы:

- интеллектуальный редактор PHP кода при подсветке синтаксиса, автоматическое дополнение кода, расширенные настройки форматирования кода, предотвращение ошибок непосредственно в процессе написания кода;
- PHP рефакторинги, code (re)arranger, детектор дублируемого кода;
- поддержка Vagrant, Composer, Command Line Tools, встроенный REST клиент, SSH консоль;
- поддержка фреймворков (MVC view для Symfony2, Yii) и специализированных плагинов для ведущих PHP фреймворков (Symfony, Yii, Drupal, Magento, CakePHP и многих иных) (рис. 13);

- поддержка 7.0, 5.6-5.3, сопрограмм, генераторов, синтаксические улучшения;

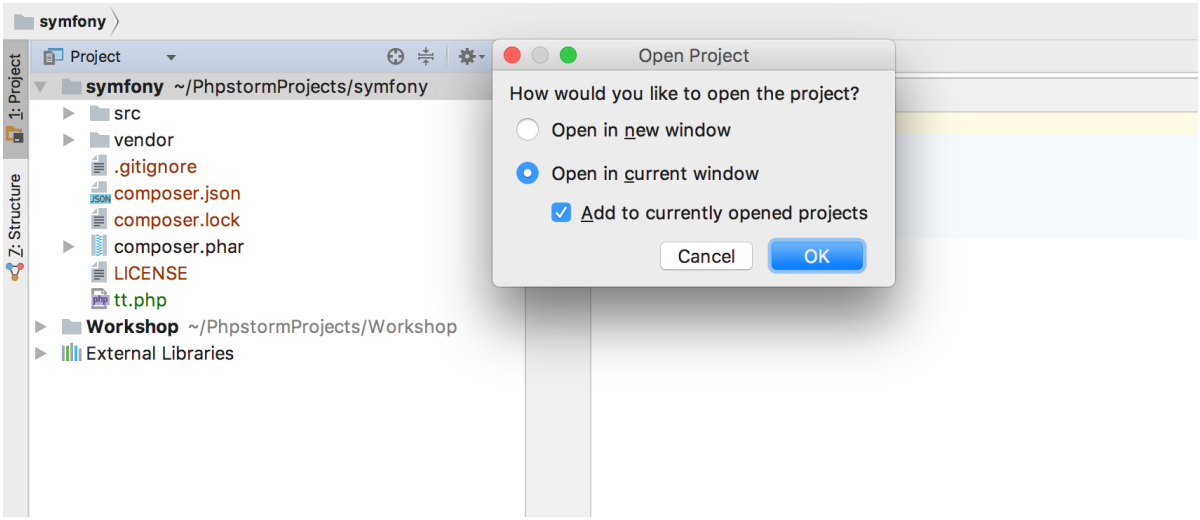

Рисунок 13 - Создание проектов на основании фреймворков

- HTML, CSS, JavaScript редактор. Отладка и модульное тестирование для JS. Поддержка HTML5, CSS, Sass, SCSS, Less, Stylus, Compass, CoffeeScript, TypeScript, ECMAScript Harmony, Emmet и другие передовые технологии веб-разработки;
- визуальный отладчик для PHP приложений, валидация конфигурации отладчика, PHPUnit с покрытием кода (поддержка PHPUnit 5), а также интеграция с профилировщиком;
- интеграция с системами управления версиями (GitHub и др.), включая унифицированный интерфейс (рис. 14);
- поддержка стилей кода, встроенные стили PSR1/PSR2, Symfony2, Zend, Drupal и других;
- полный инструментарий для фронтенд-разработок;
- удаленное развертывание приложений и автоматическая синхронизация с использованием FTP, SFTP, FTPS;

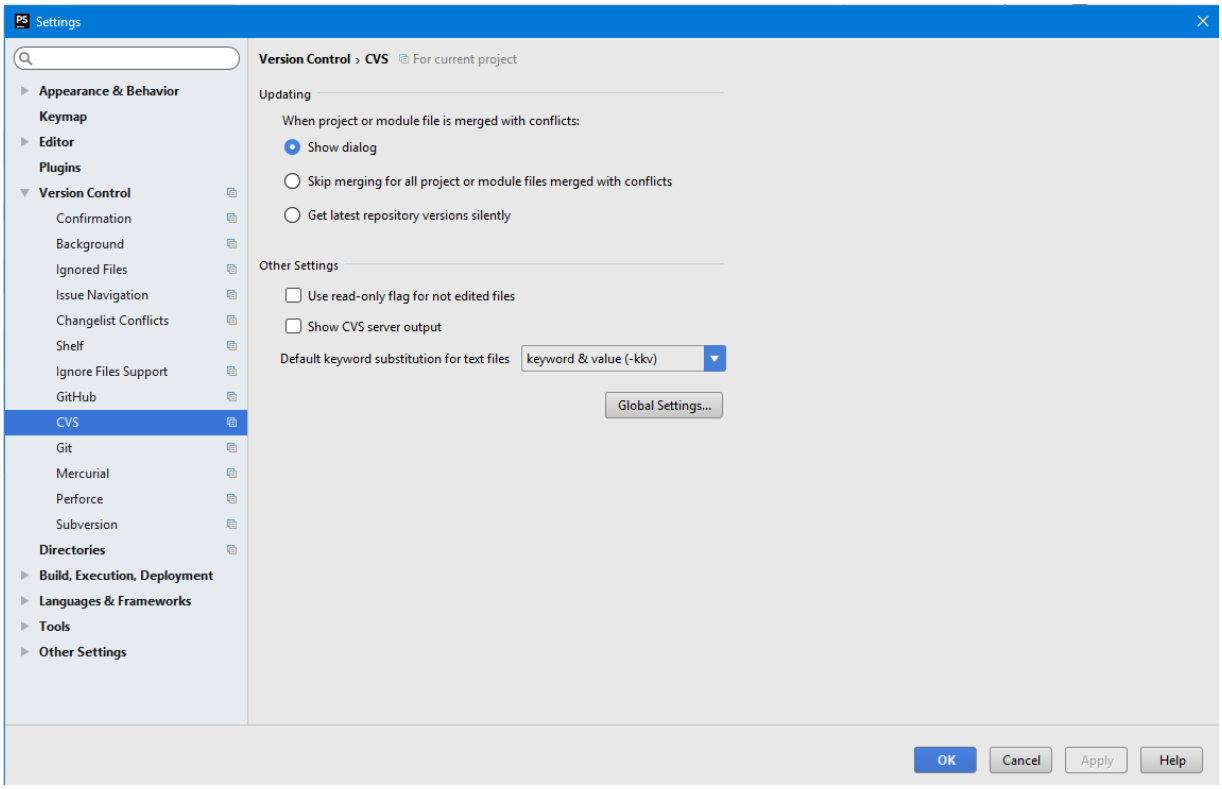

Рисунок 14 - Контроль версий в PHPStorm

- Live Edit: изменение в коде можно мгновенно посмотреть в браузере без необходимости перезагружать страницу;
- PHP UML;
- инструментарий работы с БД, SQL редактор (рис. 15);
- интеграция с баг-трекерами;
- кроссплатформенность (Mac OS X, Windows, Linux).

Новые возможности версии PhpStorm [21]:

- инструменты и фреймворки: поддержка Docker в удаленных интерпретаторах, поддержка фреймворка тестирования PHPSpec, автоматическое обнаружение и конфигурирование PHPUnit, Behat и PHPSpec из composer.json, поддержка открытия нескольких проектов в одном фрейме, поддержка стиля кода Codeigniter;

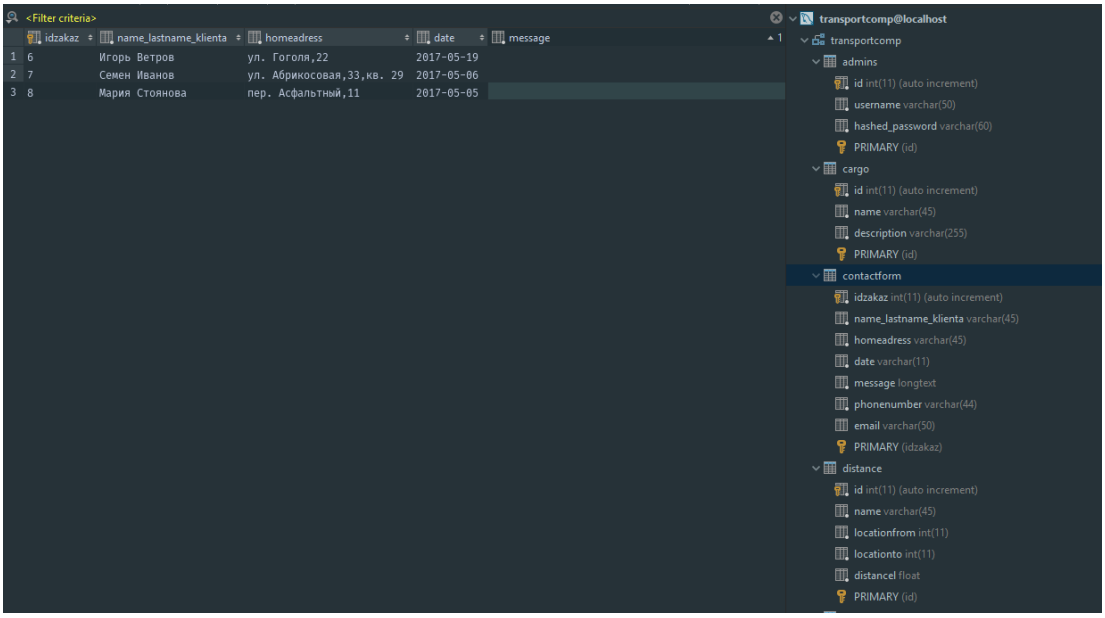

Рисунок 15 - Работа с базами данных

- новый подход к редактированию: семантическая подсветка переменных и параметров, автодополнение переопределенных методов и полей без ключевого слова function и var, улучшенная поддержка PSR-0/PSR-4;
- анализ качества кода: строгие типы PHP 7 во всем проекте, новые инспекции соглашения по присвоению имен, улучшенное предотвращение ошибок времени выполнения [21].

2.6 Обоснование выбора веб-сервер для создания информационной системы с настраиваемой концептуальной моделью данных

В качестве веб-сервера была использована сборка OpenServer. OpenServer — это свободно распространяемой программное обеспечение, подходящее для разработки веб-приложений в локальных условиях.

Программа распространяется в трех версиях ultimate, premium и basic. Ultimate-сборка включает ряд дополнительного программного обеспечения, которое упрощает веб-разработку. Файлы программы после установки представлены на рис. 16.

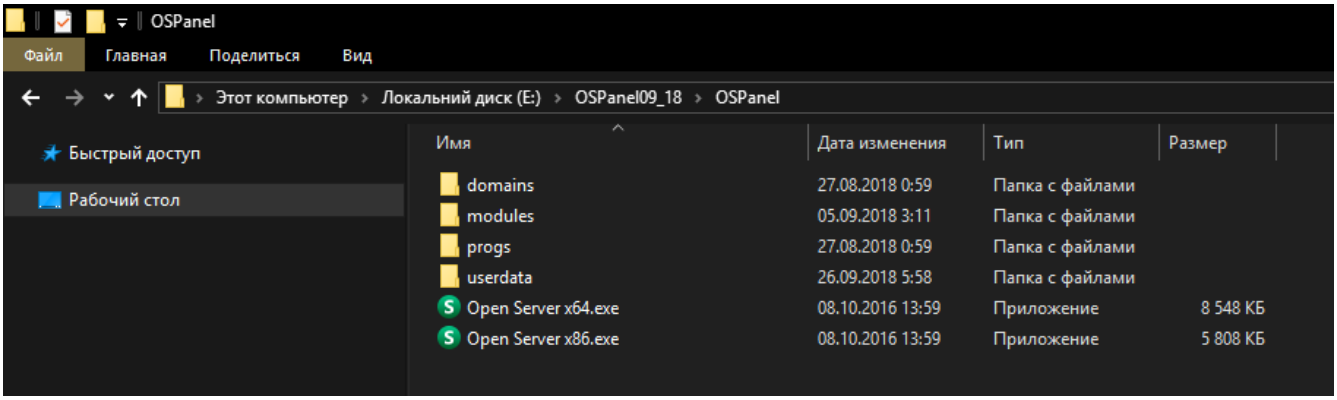

Рисунок 16 – Структура файлов локального сервера

Папка «domains» предназначена для хранения виртуальных хостов. Изначально в ней одна директория «localhost» и переход отображает, работает веб-сервер, или нет. В директории «modules» представлены модули системы: conemu, cron, database, dns, ftp, ghostscript, git, heidisql, http, imagemagick, Memcached, php, redis, sendmail, system, wget. Директория «userdata» включает пользовательский настройки компонентов OpenServer. В настройках сервера можно настроить виртуальный диск, путь по которому находятся хосты, запуск сервера в отладочном, или агрессивном режиме, внесение изменений в файл hosts операционной системы, а также IP-адрес сервера и корневую папку доменов, а также настраивать порты.

Таким образом, в качестве языка программирования выбран PHP, язык программирования, который позволяет создавать веб-ориентированные информационные системы. Как СУБД для гибкой информационной системы выбрана MySQL. Средой разработки базы данных послужит программа MySQL Workbench, как среда создания исходного кода выбрана программа PHPStorm. Функция веб-сервера для разработки будет реализована с помощью решения – OpenServer. Созданная информационная система с настраиваемой концептуальной моделью данных позволит пользователям удобно подстраивать информационную систему под новые требования предметной области в связи с изменениями бизнес-процессов.

Глава 3 ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ С НАСТРАИВАЕМОЙ КОНЦЕПТУАЛЬНОЙ МОДЕЛЬЮ ДАННЫХ

3.1 Архитектура информационной системы с настраиваемой концептуальной моделью данных

Большинство современных веб-приложений и компьютерных программ строятся на базе шаблона проектирования Модель-Представление-Контроллер (MVC). Данный шаблон построен на основе сохранения представления данных отдельно от методов, которые взаимодействуют с данными. Подобная схема приложений позволяет реализовать функционал за счет модулей, что определяет возможность быстрого изменения функционала программы.

Логика работы web-приложения с архитектурой MVC показана на рис. 17.

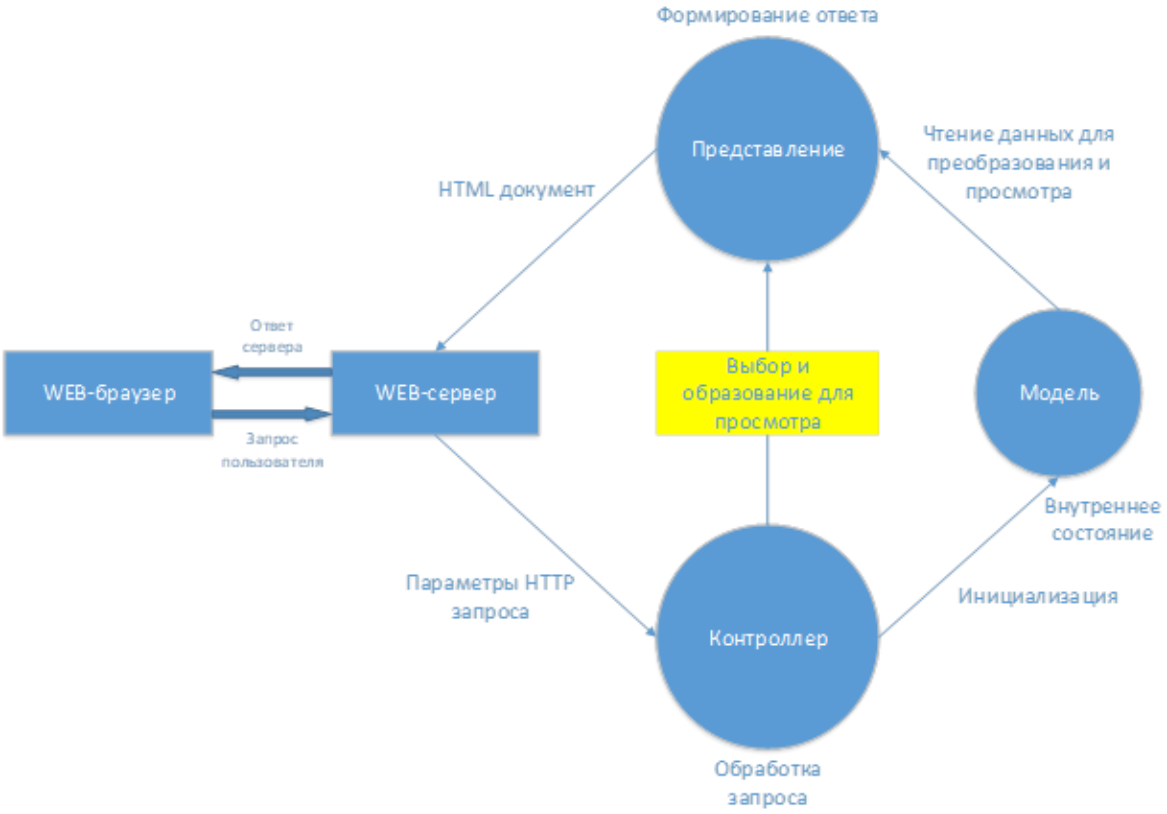

Рисунок 17 - Построение web-приложения на основе шаблона MVC

Шаблон MVC стоит использовать, т. к. это правильный и проверенный подход к разработке приложений, который делает их легко поддерживаемыми, модульными и позволяет быстро вести разработку. Разделяя приложение на отдельные модели, представления и контроллеры можно сделать программу проще и понятнее. Это позволяет без проблем добавлять новые возможности и с легкостью изменять уже внедренные функции. Модульный и разделенный на слои дизайн приложения позволяет одновременно работать программистам и дизайнерами, давая возможность очень быстро создавать прототип приложения. Логическое разделение приложения позволяет разработчикам производить изменения в одной части системы, не затрагивая другие.

Суть шаблона в разделении веб-приложения на три основные части: модель, контроллер, представление. Рассмотрим более подробнее особенности этих составных частей MVC-архитектуры.

Моделью называют постоянное хранилище данных, используемых во всей структуре. Она должна обеспечивать доступ к данным для их просмотра, отбора или записи. В общей структуре «Модель» является мостом между компонентами «Представление» и «Контроллер».

Модели представляют ту часть приложения, которая реализует за бизнес логику. Они отвечают за получение данных и представление их в необходимом для приложения виде. Это может включать в себя такие действия, как обработка, валидация, ассоциация, а также другие задачи, связанные с обработкой данных.

В самом грубом приближении объекты модели могут быть представлены, как первый слой общения с базой данных. В целом они отвечают за основные принципы, которые необходимо реализовать в приложении.

При этом «Модель» не имеет никакой связи или информации о том, что происходит с данными, когда они передаются компонентам «Представление» или «Контроллер». Единственная задача «Модели» — обработка данных в постоянном хранилище, поиск и подготовка данных, передаваемых другим составляющим

MVC. Это наиболее сложная часть системы MVC и вершина всей структуры, так как без нее невозможна связь между «Контроллером» и «Представлением».

Представление — это часть системы, в которой данным, запрашиваемым у «Модели», задается окончательный вид их вывода. В веб-приложениях, созданных на основе MVC, «Представление» — это компонент, в котором генерируется и отображается HTML-код.

Представление также перехватывает действие пользователя, которое затем передается «Контроллеру». Характерным примером этого является кнопка, генерируемая «Представлением». Когда пользователь нажимает ее, запускается действие в «Контроллере». Будучи отделенным от объектов модели, представление отвечает за вывод доступной ему информации в любом виде, который может потребоваться в приложении.

Слой представления не ограничен лишь отображением HTML или текстовых данных. Он может использоваться, чтобы представить данные в различных форматах в зависимости от функций приложения. Это может быть видео, музыка, документы или любые другие форматы.

Компоненту «Представление» никогда не передаются данные непосредственно «Контроллером». Между «Представлением» и «Контроллером» нет прямой связи — они соединяются с помощью «Модели».

Контроллеры отвечают за обработку запросов от пользователей и отображение ответа с использованием слоев моделей и представлений. «Контроллер» можно определить, как сборщик информации, которая затем передается в «Модель» с последующей организацией для хранения. Он не содержит никакой другой логики, кроме необходимости собрать входящие данные. «Контроллер» также подключается только к одному «Представлению» и одной «Модели». Это создает систему с односторонним потоком данных с одним входом и одним выходом в точках обмена данными.

«Контроллер» получает задачи на выполнение только когда пользователь взаимодействует с «Представлением», и каждая функция зависит от взаимодействия пользователя с «Представлением».

Контроллер может рассматриваться, как менеджер, который отвечает за то, чтобы все ресурсы необходимые для обработки запроса были переданы соответствующим слоям приложения. Он ждет запросы от пользователей, проверяет их корректность, правила аутентификации и авторизации, передает задачи получения или обработки данных моделям, определяет в каком виде данные должны быть отображены клиенту и передает задачи на отображение слою представления.

Контроллер, в который передается запрос, общается со слоем модели, чтобы выполнить любые необходимые операции по получению, обработке или сохранению данных. После того как эти задачи выполнены, контроллер передает соответствующему объекту представления задачу на отображение данных, полученных от модели.

На рис. 18 показан укрупненный архитектурный уровень разрабатываемого приложения.

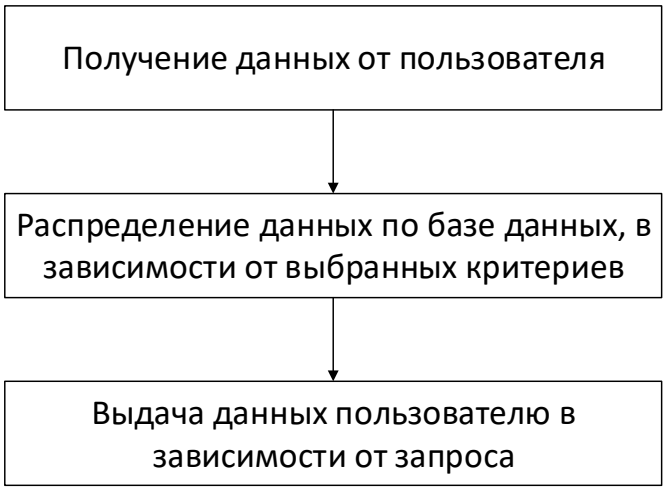

Рисунок 18 – Укрупненный уровень архитектуры приложения

Для реализации приложения с настраиваемой концептуальной моделью данных в качестве архитектурного основания выбран фреймворк Yii2, который реализует паттерн MVC. Yii2 представляет собой высокопроизводительный компонентный PHP фреймворк, предназначенный для быстрой разработки современных веб приложений. Для организации кода Yii2 использует архитектурный паттерн MVC (Model-View-Controller).

Достоинства данного фреймфорка:

- Для доступа к базам данных используются четыре метода: YII DAO (обеспечивают объектно-ориентированный API для доступа к реляционным базам данных), построитель запросов (позволяет конструировать SQL выражения в программируемом и независимом от СУБД виде и генерировать более безопасные SQL выражения), Active Record (обеспечивает доступ к базе данных через методы класса), миграция баз данных (позволяет отслеживать изменения в базах данных при помощи терминов миграции баз данных).
- Проверка входящих данных от пользователя осуществляется с помощью простого вызова метода yii\base\Model::validate(), для которого можно задавать определенные правила проверки.
- Yii2 предоставляет два метода авторизации: фильтры контроля доступа (ACF) и контроль доступа на основе ролей (RBAC).
- Для обеспечения безопасного хранения паролей фреймворк предоставляет механизмы хеширования паролей.

Каждая подсистема данной системы представляет собой модули в фреймворке, структура которых представлена ниже. Так, кроме MVC, Yii2 приложения также имеют следующие сущности:

- входные скрипты: это PHP скрипты, которые доступны напрямую конечному пользователю приложения. Они ответственны за запуск и обработку входящего запроса;

- приложения: это глобально доступные объекты, которые осуществляют корректную работу различных компонентов приложения и их координацию для обработки запроса;
- компоненты приложения: это объекты, зарегистрированные в приложении и предоставляющие различные возможности для обработки текущего запроса;
- модули: это самодостаточные пакеты, которые включают в себя полностью все средства для MVC. Приложение может быть организованно с помощью нескольких модулей;
- фильтры: это код, который должен быть выполнен до и после обработки запроса контроллерами;
- виджеты: это объекты, которые могут включать в себя представления. Они могут содержать различную логику и быть использованы в различных представлениях.

Схематически структура приложения представлена на рис. 19.

Все запросы, обрабатываемые Yii2 приложением, проходят подобный путь:

- Пользователь создает запрос ко входному скрипту web/index.php.
- Входной скрипт загружает конфигурацию и создает экземпляр приложения для обработки запроса.
- Приложение определяет запрошенный маршрут при помощи компонента request.
- Приложение создает экземпляр контроллера для обработки запроса.
- Контроллер создает экземпляр действия и выполняет фильтры для этого действия.
- При неудачном выполнении любого фильтра действие не выполняется.

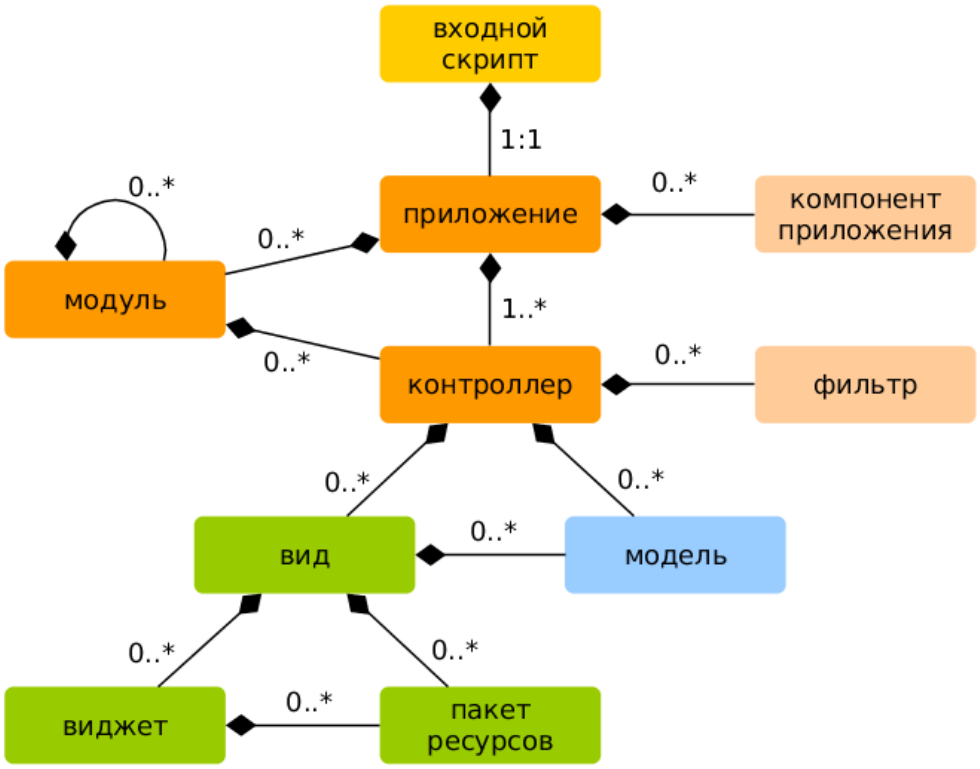

Рисунок 19 – Схема приложения

- При успешном выполнении всех фильтров выполняется действие.
- Действие загружает модель данных, возможно, из базы данных.
- Действие рендерит представление и передает ему модель данных.
- Результат рендеринга передается в компонент приложения response.
- Компонент response посылает готовые данные пользователю.

На рис. 20 представлена диаграмма обработки запроса приложением.

Для организации функционала важным элементом приложения является работа с куки и сессиями.

Сессии и куки позволяют сохранять пользовательские данные между запросами. При использовании чистого PHP можно получить доступ к этим данным через глобальные переменные $_SESSION и $_COOKIE, соответственно. Yii2 инкапсулирует сессии и куки в объекты, что дает возможность обращаться к ним в объектно-ориентированном стиле и дает дополнительное удобство в работе.

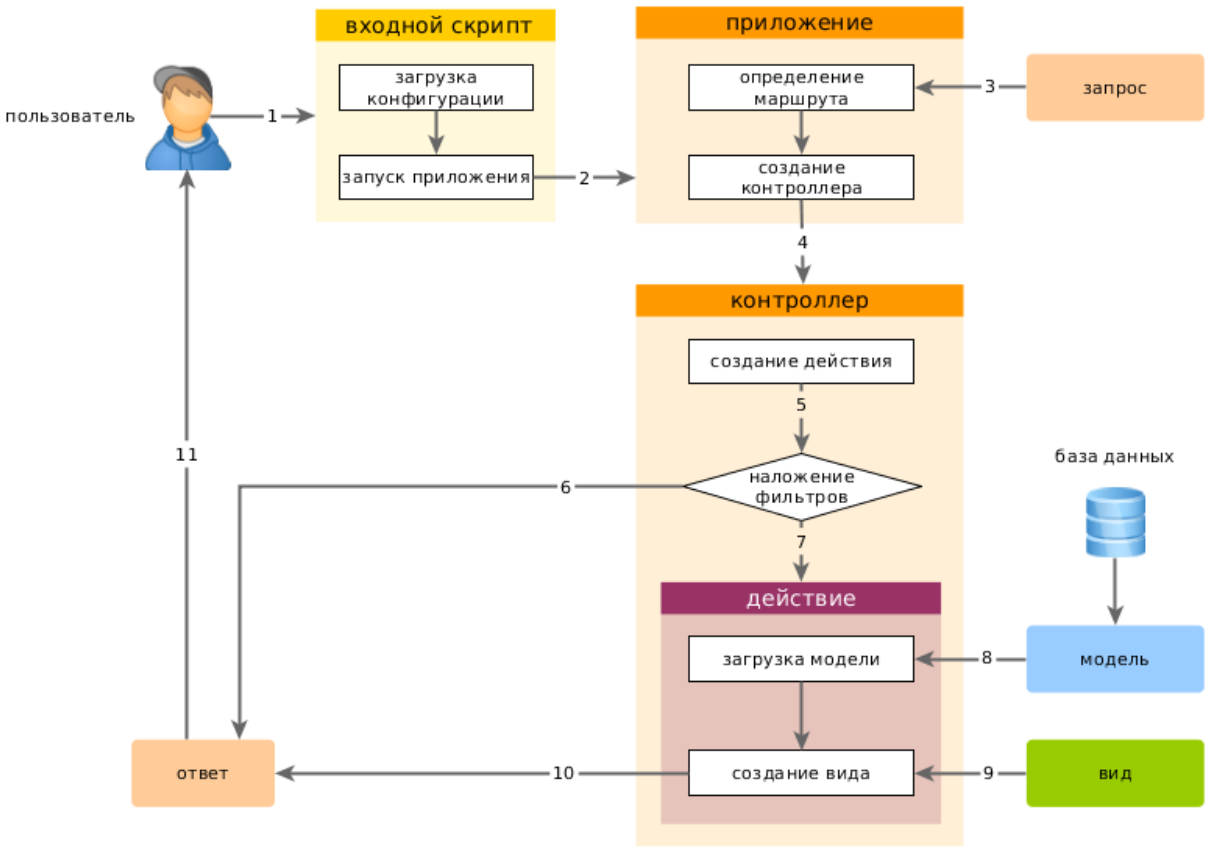

Рисунок 20 – Схема обработки запроса

По аналогии с запросами и ответами, к сессии можно получить доступ через session компонент приложения, который по умолчанию является экземпляром yii\web\Session.

Открыть и закрыть сессию можно следующим образом:

```
$session = Yii::$app->session;
// проверяем что сессия уже открыта
if ($session->isActive) ...
// открываем сессию
$session->open();
// закрываем сессию
$session->close();
// уничтожаем сессию и все связанные с ней данные.
$session->destroy();
```

Можно вызывать open() и close() многократно без возникновения ошибок; внутри компонента все методы проверяют сессию на факт того, открыта она или нет.

Получить доступ к сохраненным в сессию данным можно следующим образом:

```
$session = Yii::$app->session;

// получение переменной из сессии. Следующие способы использования эквивалентны:
$language = $session->get('language');
$language = $session['language'];
$language = isset($_SESSION['language']) ? $_SESSION['language'] : null;

// запись переменной в сессию. Следующие способы использования эквивалентны:
$session->set('language', 'en-US');
$session['language'] = 'en-US';
$_SESSION['language'] = 'en-US';

// Удаление переменной из сессии. Следующие способы использования эквивалентны:
$session->remove('language');
unset($session['language']);
unset($_SESSION['language']);

// проверка на существование переменной в сессии. Следующие способы использования эквивалентны:
if ($session->has('language')) ...
if (isset($session['language'])) ...
if (isset($_SESSION['language'])) ...

// Обход всех переменных в сессии. Следующие способы использования эквивалентны:
foreach ($session as $name => $value) ...
foreach ($_SESSION as $name => $value) ...
```

При получении данных из сессии через компонент session, сессия будет автоматически открыта, если она не была открыта до этого. В этом заключается отличие от получения данных из глобальной переменной $_SESSION, которое требует обязательного вызова session_start().

Yii2 представляет каждую cookie как объект yii\web\Cookie. Оба компонента приложения yii\web\Request и yii\web\Response поддерживают коллекции cookie через свойство cookies. В первом случае коллекция cookie является их представлением из HTTP-запроса, во втором - представляет cookies, которые будут отправлены пользователю.

Получить cookies из текущего запроса можно следующим образом:

```
// получение коллекции кук (yii\web\CookieCollection) из компонента "request"
$cookies = Yii::$app->request->cookies;

// получение куки с названием "language. Если кука не существует, "en"  будет возвращено как значение по-умолчанию.
$language = $cookies->getValue('language', 'en');

// альтернативный способ получения куки "language"
if (($cookie = $cookies->get('language')) !== null) {
    $language = $cookie->value;
}

// теперь переменную $cookies можно использовать как массив
if (isset($cookies['language'])) {
    $language = $cookies['language']->value;
}

// проверка на существование куки "language"
if ($cookies->has('language')) ...
if (isset($cookies['language'])) ...
```

Во время записи и чтения cookie через компоненты request и response, фреймворк предоставляет автоматическую валидацию, которая обеспечивает защиту cookie от модификации на стороне клиента. Это достигается за счет подписи каждой cookie секретным ключом, позволяющим приложению распознать cookie, которая была модифицирована на клиентской стороне. В таком случае cookie не будут доступны через свойство cookie collection компонента request.

Валидация cookie защищает только от их модификации. Если валидация не была пройдена, получить доступ к cookies все еще можно через глобальную

переменную $_COOKIE. Это связано с тем, что дополнительные пакеты и библиотеки могут манипулировать cookies без вызова валидации, которую обеспечивает Yii.

По умолчанию валидация cookie включена. Её можно отключить, установив свойство yii\web\Request::enableCookieValidation в false, однако мы настоятельно не рекомендуем это делать.

С помощью описанных выше механизмов работы с базой данных, разделения ролей, cookie и сессией обеспечивается безопасность и надежность работы приложения, которое основано на MVC-фреймворке Yii2.

3.2 Проектирование базы данных для гибкой информационной системы

Как было обозначено выше (п. 2.4) для проектирования базы данных информационной системы использована программа MySQLWorkbench. Структура проектируемой базы данных и ее таблиц представлена на рис. 21 – 31.

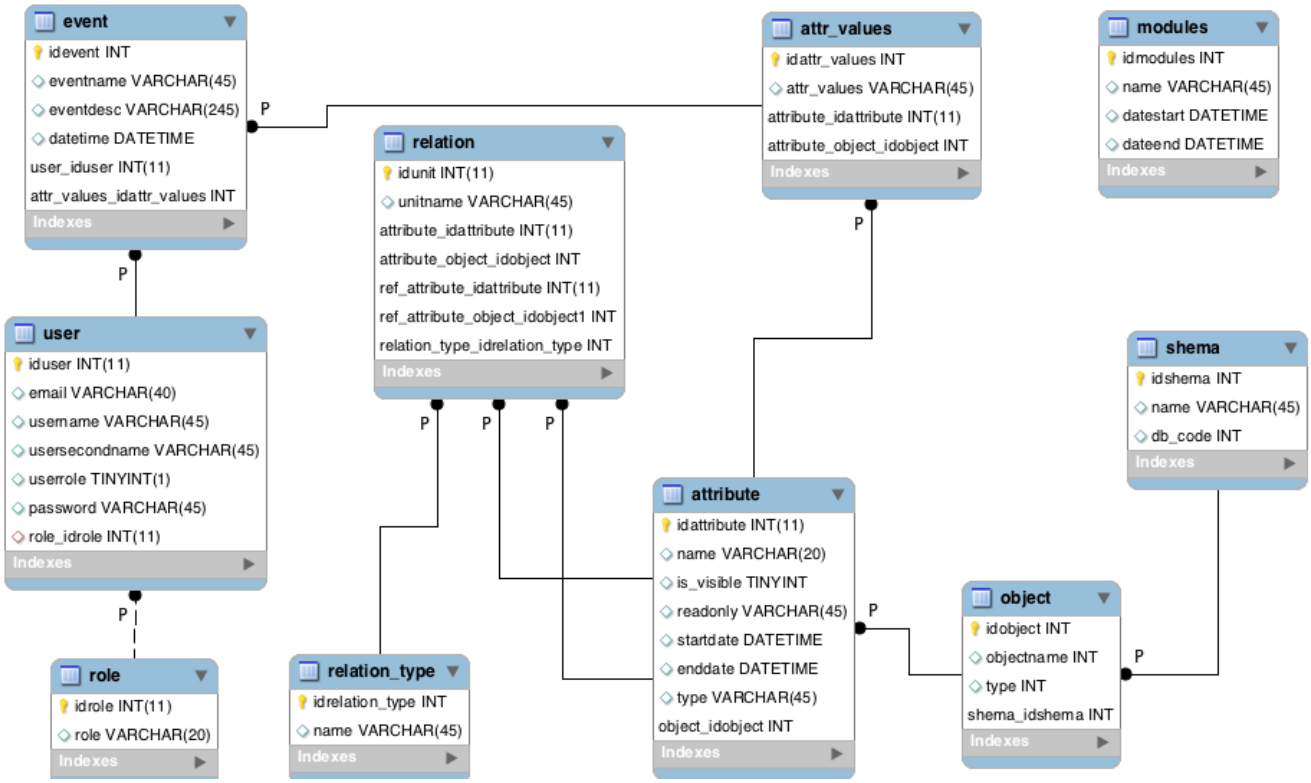

Рисунок 21 – Модель базы данных

Проектируемая база данных включает 10 таблиц, которые предназначены для сохранения данных пользователей и метаэлементов, которые предназначены для описания сущностей предметной области.

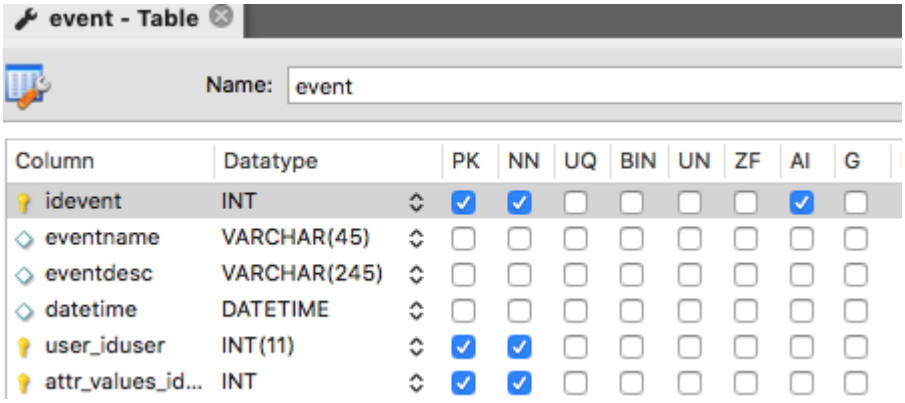

Рисунок 22 – Модель таблицы «event» (События)

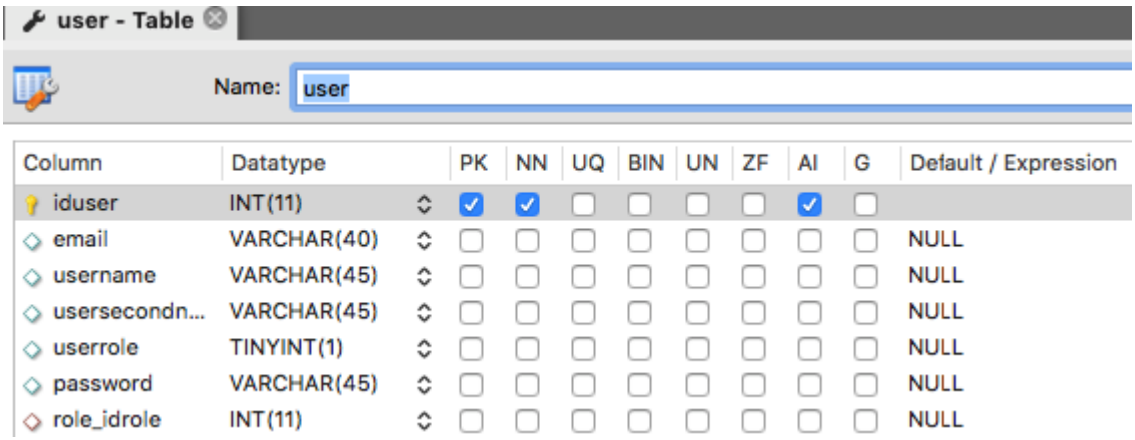

Рисунок 23 - Модель таблицы «user» (Пользователи)

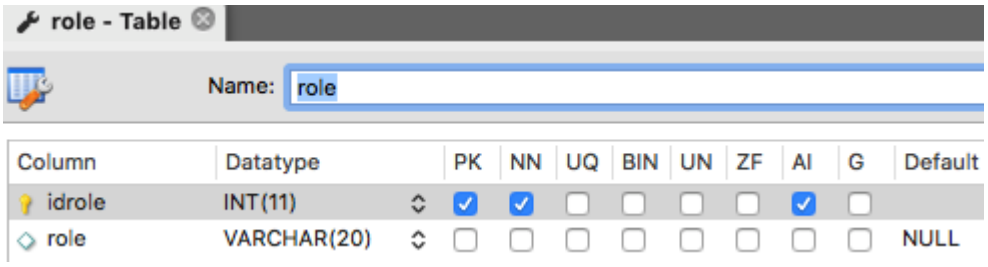

Рисунок 24 - Модель таблицы «role» (Роли пользователей)

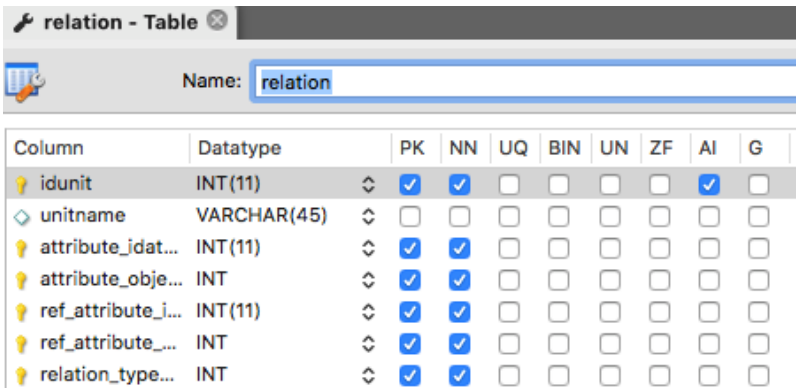

Рисунок 25 – Модель таблицы «relation» (Связи)

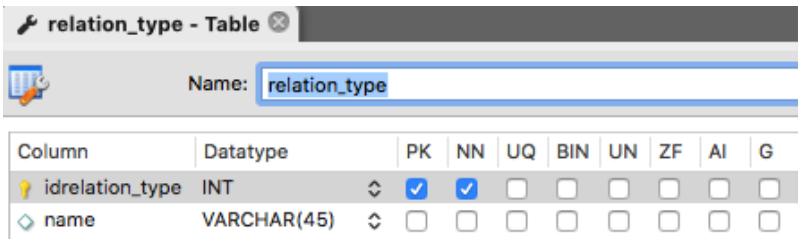

Рисунок 26 - Модель таблицы «relation_type» (Типы связей)

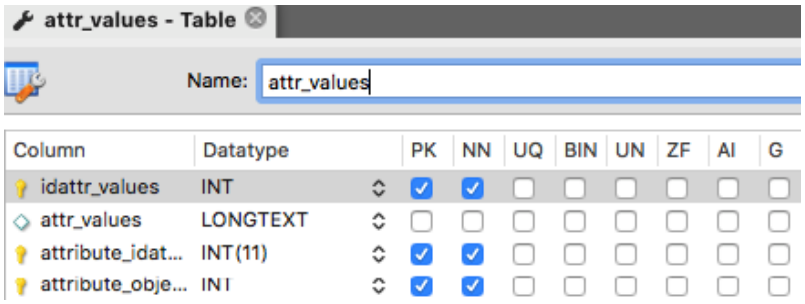

Рисунок 27 - Модель таблицы «attr_value» (значения атрибутов)

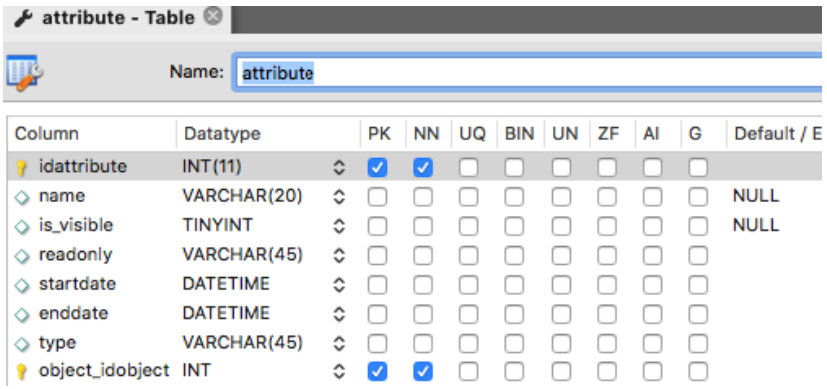

Рисунок 28 - Модель таблицы «attributes» (Атрибуты)

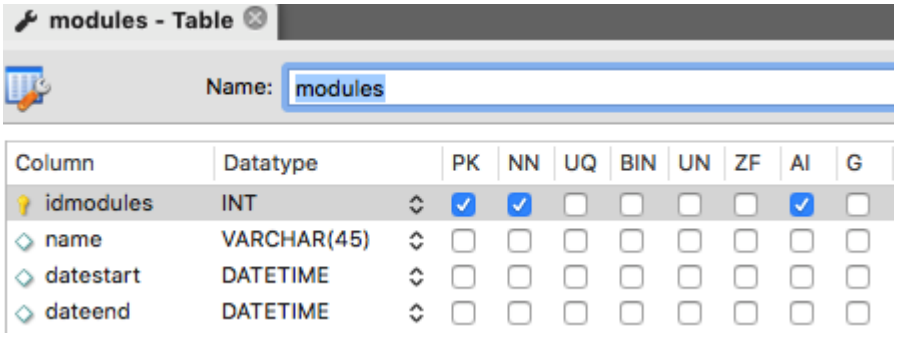

Рисунок 29 - Модель таблицы «modules» (Модули)

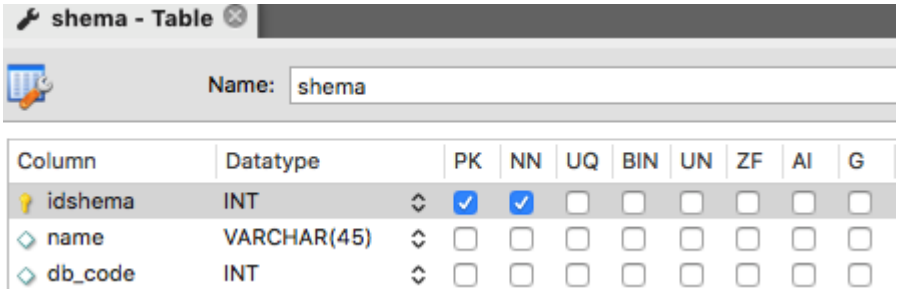

Рисунок 30 - Модель таблицы «shema» (Базы данных)

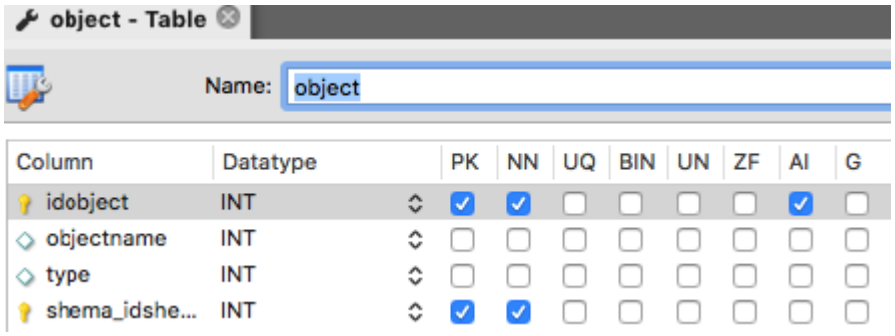

Рисунок 31 - Модель таблицы «object» (Таблицы)

После этого, для создания базы данных на сервере MySQL необходимо выполнить следующий запрос:

```
CREATE TABLE IF NOT EXISTS `metamodels`.`shema` (
  `idshema` INT NOT NULL,
  `name` VARCHAR(45) NULL,
  `db_code` INT NULL,
  PRIMARY KEY (`idshema`))
ENGINE = InnoDB;
```

```
CREATE TABLE IF NOT EXISTS `metamodels`.`object` (
  `idobject` INT NOT NULL AUTO_INCREMENT,
  `objectname` INT NULL,
  `type` INT NULL,
  `shema_idshema` INT NOT NULL,
  PRIMARY KEY (`idobject`, `shema_idshema`),
  INDEX `fk_object_shema1_idx` (`shema_idshema` ASC),
  CONSTRAINT `fk_object_shema1`
    FOREIGN KEY (`shema_idshema`)
    REFERENCES `metamodels`.`shema` (`idshema`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
CREATE TABLE IF NOT EXISTS `metamodels`.`attribute` (
  `idattribute` INT(11) NOT NULL,
  `name` VARCHAR(20) NULL DEFAULT NULL,
  `is_visible` TINYINT NULL DEFAULT NULL,
  `readonly` VARCHAR(45) NULL,
  `startdate` DATETIME NULL,
  `enddate` DATETIME NULL,
  `type` VARCHAR(45) NULL,
  `object_idobject` INT NOT NULL,
  PRIMARY KEY (`idattribute`, `object_idobject`),
  INDEX `fk_attribute_object1_idx` (`object_idobject` ASC),
  CONSTRAINT `fk_attribute_object1`
    FOREIGN KEY (`object_idobject`)
    REFERENCES `metamodels`.`object` (`idobject`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
AUTO_INCREMENT = 5
DEFAULT CHARACTER SET = utf8;
CREATE TABLE IF NOT EXISTS `metamodels`.`role` (
  `idrole` INT(11) NOT NULL AUTO_INCREMENT,
  `role` VARCHAR(20) NULL DEFAULT NULL,
  PRIMARY KEY (`idrole`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
CREATE TABLE IF NOT EXISTS `metamodels`.`relation_type` (
```

```
  `idrelation_type` INT NOT NULL,
  `name` VARCHAR(45) NULL,
  PRIMARY KEY (`idrelation_type`))
ENGINE = InnoDB;
CREATE TABLE IF NOT EXISTS `metamodels`.`relation` (
  `idunit` INT(11) NOT NULL AUTO_INCREMENT,
  `unitname` VARCHAR(45) NULL DEFAULT NULL,
  `attribute_idattribute` INT(11) NOT NULL,
  `attribute_object_idobject` INT NOT NULL,
  `ref_attribute_idattribute` INT(11) NOT NULL,
  `ref_attribute_object_idobject1` INT NOT NULL,
  `relation_type_idrelation_type` INT NOT NULL,
  PRIMARY KEY (`idunit`, `attribute_idattribute`, `attribute_object_idobject`, `ref_attribute_idattribute`, `ref_attribute_object_idobject1`, `relation_type_idrelation_type`),
  INDEX `fk_relation_attribute1_idx` (`attribute_idattribute` ASC, `attribute_object_idobject` ASC),
  INDEX `fk_relation_attribute2_idx` (`ref_attribute_idattribute` ASC, `ref_attribute_object_idobject1` ASC),
  INDEX `fk_relation_relation_type1_idx` (`relation_type_idrelation_type` ASC),
  CONSTRAINT `fk_relation_attribute1`
    FOREIGN KEY (`attribute_idattribute` , `attribute_object_idobject`)
    REFERENCES `metamodels`.`attribute` (`idattribute` , `object_idobject`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_relation_attribute2`
    FOREIGN KEY (`ref_attribute_idattribute` , `ref_attribute_object_idobject1`)
    REFERENCES `metamodels`.`attribute` (`idattribute` , `object_idobject`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_relation_relation_type1`
    FOREIGN KEY (`relation_type_idrelation_type`)
    REFERENCES `metamodels`.`relation_type` (`idrelation_type`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
AUTO_INCREMENT = 7
DEFAULT CHARACTER SET = utf8;
CREATE TABLE IF NOT EXISTS `metamodels`.`user` (
  `iduser` INT(11) NOT NULL AUTO_INCREMENT,
  `email` VARCHAR(40) NULL DEFAULT NULL,
  `username` VARCHAR(45) NULL DEFAULT NULL,
```

```
  `usersecondname` VARCHAR(45) NULL DEFAULT NULL,
  `userrole` TINYINT(1) NULL DEFAULT NULL,
  `password` VARCHAR(45) NULL DEFAULT NULL,
  `role_idrole` INT(11) NULL DEFAULT NULL,
  PRIMARY KEY (`iduser`),
  INDEX `fk_user_role1_idx` (`role_idrole` ASC),
  CONSTRAINT `fk_user_role1`
    FOREIGN KEY (`role_idrole`)
    REFERENCES `metamodels`.`role` (`idrole`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
AUTO_INCREMENT = 2
DEFAULT CHARACTER SET = utf8;
CREATE TABLE IF NOT EXISTS `metamodels`.`attr_values` (
  `idattr_values` INT NOT NULL,
  `attr_values` VARCHAR(45) NULL,
  `attribute_idattribute` INT(11) NOT NULL,
  `attribute_object_idobject` INT NOT NULL,
  PRIMARY KEY (`idattr_values`, `attribute_idattribute`, `attribute_object_idobject`),
  INDEX `fk_attr_values_attribute1_idx` (`attribute_idattribute` ASC, `attribute_object_idobject` ASC),
  CONSTRAINT `fk_attr_values_attribute1`
    FOREIGN KEY (`attribute_idattribute` , `attribute_object_idobject`)
    REFERENCES `metamodels`.`attribute` (`idattribute` , `object_idobject`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
CREATE TABLE IF NOT EXISTS `metamodels`.`event` (
  `idevent` INT NOT NULL AUTO_INCREMENT,
  `eventname` VARCHAR(45) NULL,
  `eventdesc` VARCHAR(245) NULL,
  `datetime` DATETIME NULL,
  `user_iduser` INT(11) NOT NULL,
  `attr_values_idattr_values` INT NOT NULL,
  PRIMARY KEY (`idevent`, `user_iduser`, `attr_values_idattr_values`),
  INDEX `fk_event_user1_idx` (`user_iduser` ASC),
  INDEX `fk_event_attr_values1_idx` (`attr_values_idattr_values` ASC),
  CONSTRAINT `fk_event_user1`
    FOREIGN KEY (`user_iduser`)
```

```
    REFERENCES `metamodels`.`user` (`iduser`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_event_attr_values1`
    FOREIGN KEY (`attr_values_idattr_values`)
    REFERENCES `metamodels`.`attr_values` (`idattr_values`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
CREATE TABLE IF NOT EXISTS `metamodels`.`modules` (
  `idmodules` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) NULL,
  `datestart` DATETIME NULL,
  `dateend` DATETIME NULL,
  PRIMARY KEY (`idmodules`))
ENGINE = InnoDB;
```

Таким образом, с помощью MySQLWorkbench была разработана модель базы данных для гибкой информационной системы, которая предназначена для сохранения основных сущностей информационной системы, которая включает метамодель и метасущности.

3.3 Алгоритмы функционирования системы

Для функционирования системы необходим ряд алгоритмов, которые будут отвечать за выполнение следующих функций:

- добавление сущностей (для описания таблиц в базе данных);
- редактирование сущностей предметной области;
- добавление атрибутов (для описания атрибутов для таблиц в базе данных);
- редактирование атрибутов сущностей предметной области;
- добавления связей (для описания связей между таблицами);
- редактирование связей между сущностями предметной области;
- генерация форм для ввода данных;

– введение данных в созданные таблицы;

– управление данными в базе данных.

Представленные функции позволят создавать базы данных с возможностью изменения концептуальных данных, таких как таблицы, атрибуты и связи между таблицами. Ниже представлено схематическое описание указанных функций (рис. 32 – 34).

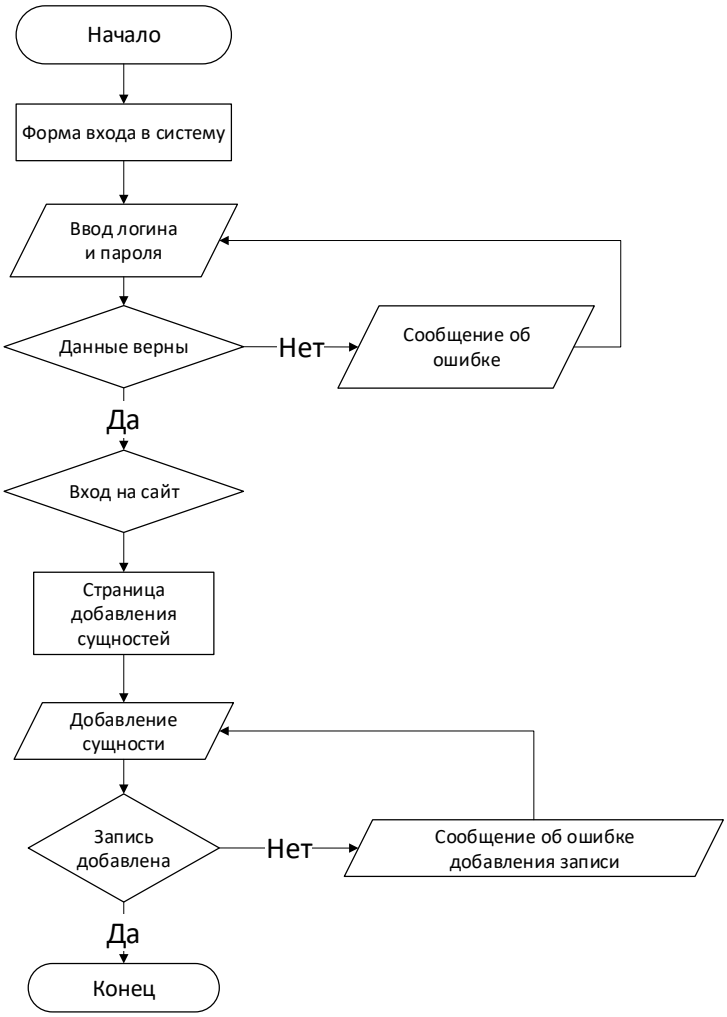

Рисунок 32 – Алгоритм добавления сущности

```
public function actionCreate()
{
   $model = new Object();
   if ($model->load(Yii::$app->request->post()) && $model->save()) {
      return $this->redirect(['view', 'id' => $model->idobject]);
```

```
    }
    return $this->render('create', [
        'model' => $model,
    ]);
}
public function actionUpdate($id)
{
    $model = $this->findModel($id);
    if ($model->load(Yii::$app->request->post()) && $model->save()) {
        return $this->redirect(['view', 'id' => $model-> idobject]);
    }
    return $this->render('update', [
        'model' => $model,
    ]);
}
```

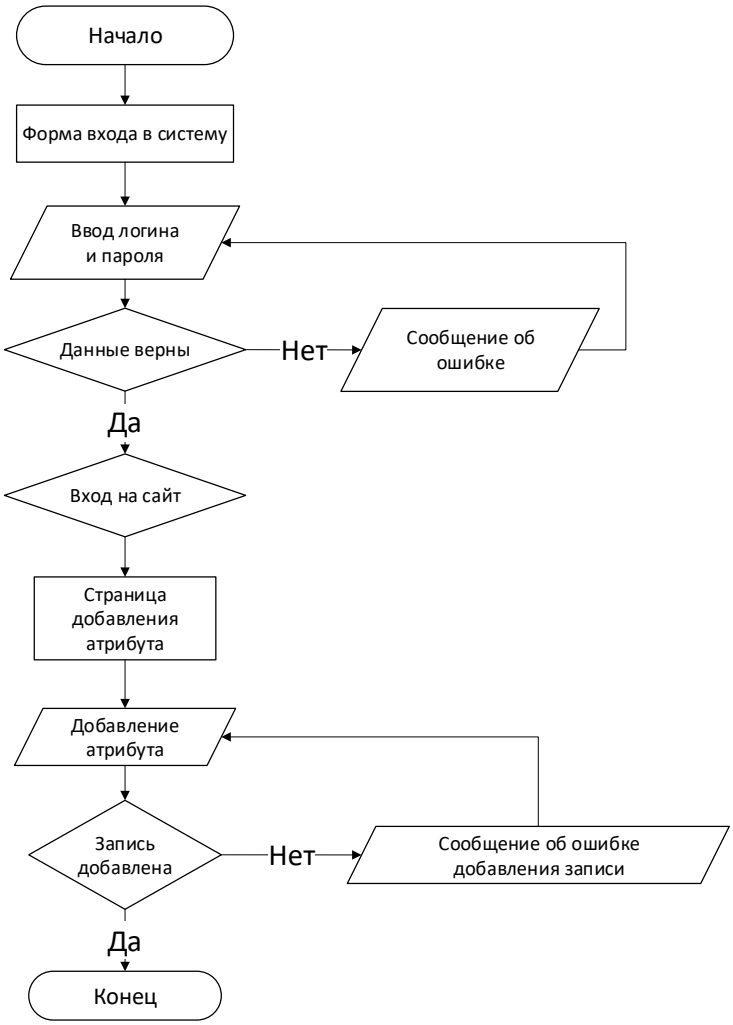

Рисунок 33 – Алгоритм добавления атрибута

```
public function actionCreate()
{
    $model = new Attribute();

    if ($model->load(Yii::$app->request->post()) && $model->save()) {
        return $this->redirect(['view', 'id' => $model->idattribute]);
    }

    return $this->render('create', [
        'model' => $model,
    ]);
}


public function actionUpdate($id)
{
    $model = $this->findModel($id);

    if ($model->load(Yii::$app->request->post()) && $model->save()) {
        return $this->redirect(['view', 'id' => $model->idattribute]);
    }

    return $this->render('update', [
        'model' => $model,
    ]);
}


public function actionDelete($id)
{
    $this->findModel($id)->delete();

    return $this->redirect(['index']);
}
```

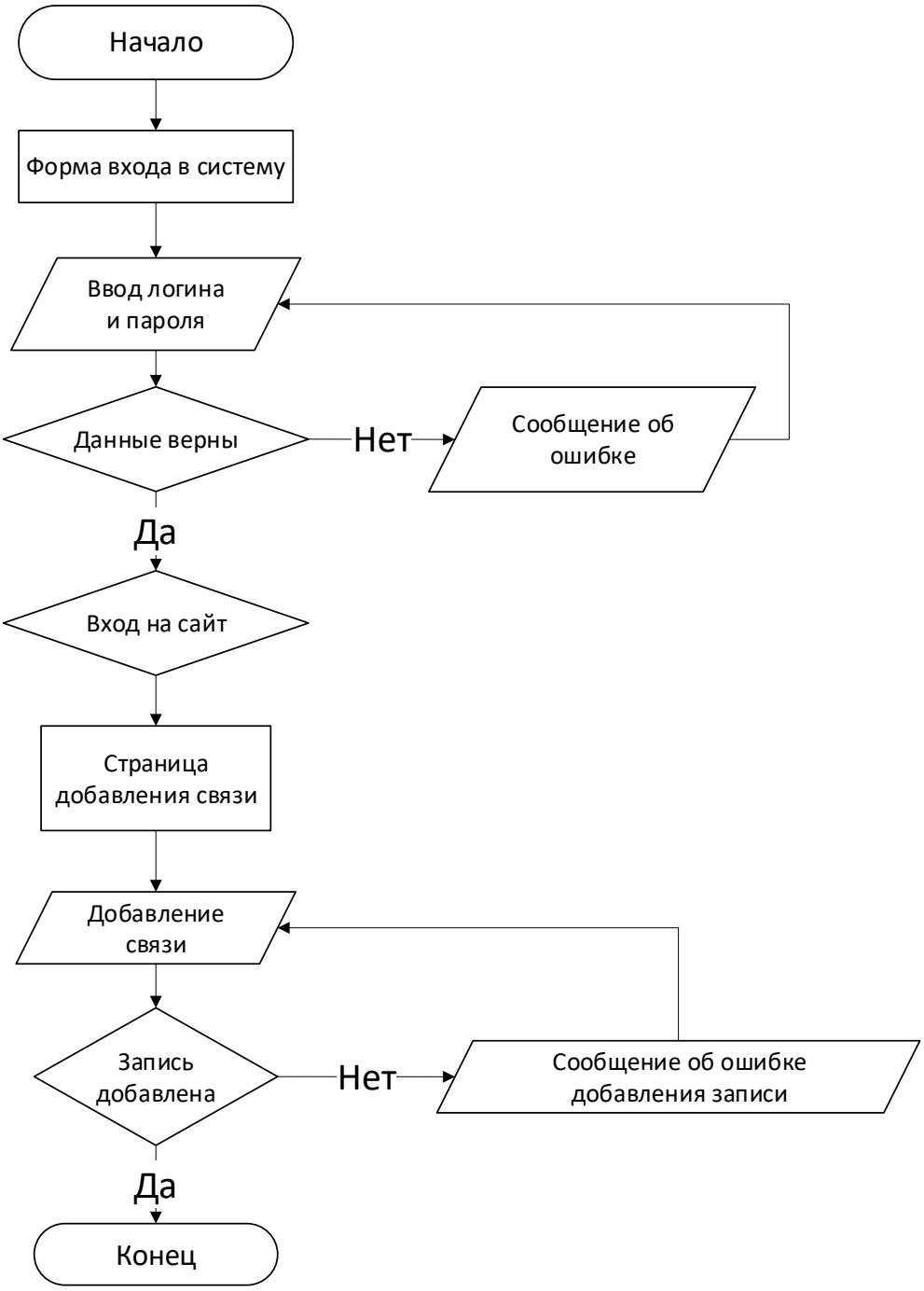

Рисунок 34 – Алгоритм добавления связи

```
public function actionCreate()
{
    $model = new Relation();

    if ($model->load(Yii::$app->request->post()) && $model->save()) {
        return $this->redirect(['view', 'id' => $model->idrelation]);
```

```
    }

    return $this->render('create', [
        'model' => $model,
    ]);
}


public function actionUpdate($id)
{
    $model = $this->findModel($id);

    if ($model->load(Yii::$app->request->post()) && $model->save()) {
        return $this->redirect(['view', 'id' => $model-> idrelation]);
    }

    return $this->render('update', [
        'model' => $model,
    ]);
}


public function actionDelete($id)
{
    $this->findModel($id)->delete();

    return $this->redirect(['index']);
}
```

С помощью представленных функций и алгоритмов пользователю предоставляется возможность настройки концептуальной модели базы данных для любой предметной области. Осуществляется это за счет возможности описания сущностей, атрибутов и связей в таблицах базы данных, которые являются метауровнем создаваемой базы данных. На метуровне формируется описание таблиц предметного уровня.

3.4 Математическое обеспечение решаемой задачи

Классической методикой проектирования базы данных является создание отдельной таблицы для каждой описываемой моделью данных сущности. В процессе нормализации и унификации создаются отдельные таблицы для хранения классификации атрибутов сущности (классификаторы). Такой подход хорошо работает только с относительно статичной структурой базы данных. Любое же изменение предметной области может привести к внесению изменений в структуру хранящих их таблиц, а значит, потребует время и расходы на работу программистов. Эта операция, несложная на этапе разработки, становится крайне проблематичной при больших объемах данных и отсутствии у разработчика непосредственного доступа к базе данных.

Максимальная гибкость информационной системы достигается за счет метаданных, с помощью которых предоставляется возможность описать предметную область, в которой будет использована система и которые описывают условия работы системы, характеризуют протекающие бизнес-процессы и пользователей этой системы.

Методология формируемых представлений данных тесно связана с методологией концептуального метода проектирования баз данных [48], т. е. систематический подход к созданию, применению и управлению базами данных и метаданных, обеспечивающий:

- учет изменений предметной области или ее представления;
- управление базами данных и метаданных, включая совместное использование (интегрированное) объектов данных, объектов метаданных и программ;
- унификацию представления данных, метаданных (в виде объектов данных и объектов метаданных) и языковых средств работы с ними в рамках общей вычислительной модели, допускающей ряд специализаций;
- обеспечение проектных процедур представления предметной области

путем установления связей объектов данных и объектов метаданных;

- поддержание возрастающего со временем многообразия данных, метаданных и языковых средств работы с ними;
- оптимизацию работы с базами данных и метаданных;
- поддержку работы с различными БД и схемами БД;
- разграничение прав доступа пользователей к информации (к модулям, задачам, сущностям, атрибутам, функциям, а также отдельным записям);
- поддержку целостности данных на физическом и логическом уровнях;
- логический контроль данных;
- ведение журнала изменений, происходящих с БД.

Таким образом, понятие метаданных является ключевым понятием информационной системы с настраиваемой концептуальной моделью данных. Метаданные (М) – это описание сущности, однозначно его идентифицирующее, где отражаются как его статические, так и динамические характеристики. Для описания метаданных введены общие метаструктурные сущности, которые могут быть представлены модулями, задачами, функциями, отчетами, фрагментами, сущностями, атрибутами, схемами, пользователями, группами пользователей. Отобразим математическое описание представленных элементов.

Модулем p называется набор:

$$p = \langle T, E_T, M_p \rangle, \quad (1)$$

где $p \in P$; T - множество решаемых модулем задач; $E_T \subseteq T \times T$ - отношение иерархичности задач; M_p - метаданные модуля.

Модуль, доступный на чтение:

$$p' = \langle P, G_p \rangle, \quad (2)$$

где G_p - множество групп пользователей модуля, ответственных за выполнение его задач, причем $G_p \subset G$;

Задачей t называется набор:

$$t = \langle F, C_t, R, S, M_t \rangle, \quad (3)$$

где $t \in T$; F - множество функций, необходимых для решения данной задачи; C_t - множество фрагментов данных задачи, причем $C_t \subset C$, где C — множество фрагментов данных системы; R - множество отчетов, отнесенных к задаче; S - множество пользовательских настроек; M_t - метаданные задачи.

Доступная на чтение задача описывается формулой:

$$t' = \langle T, G_t \rangle, \quad (4)$$

где G_t - множество групп исполнителей задачи, ответственных за ее выполнение, причем $G_t \subset G$;

Функцией f называется пара:

$$f = \langle o, M_f \rangle, \quad (5)$$

где $f \in F$; o - элементарное действие, выполняемое системой и возвращающее определённый результат (добавление, удаление данных, поиск и т. д.); M_f - метаданные функции.

Доступная группе пользователей функция:

$$f' = \langle F, G_f \rangle, \quad (6)$$

где G_f - множество групп пользователей, имеющих доступ к функции, причем $G_f \subset G$;

Фрагментом данных с называется набор:

$$c = \langle A, W, M_c \rangle, \quad (7)$$

где $c \in C$; A - множество атрибутов системы; W - множество условий отбора данных; M_c - метаданные фрагмента.

Доступным группе пользователей называется фрагмент данных:

$$c' = \langle C, G_c \rangle, \quad (8)$$

где G_c - множество групп пользователей фрагмента, причем $G_r \subset G$; C - все фрагменты данных системы.

Пользовательскими настройками s называется пара:

$$s = \langle u, M_s \rangle, \quad (9)$$

где $s \in S$; u - пользователь, сохранивший настройку, причем $u \in U$; M_s - метаданные пользовательской настройки.

Сущность в гибкой информационной системе представляет собой логическое понятие, его главное назначение – предоставить уникальный идентификатор и набор атрибутов, по которым она будет отличаться.

Атрибутом a сущности называется набор:

$$a = \langle FS, E_{FS}, \Theta, MD, TPE, M_a \rangle, \quad (10)$$

где $a \in A$, при этом A - множество всех атрибутов системы; FS - множество полей БД, из которых состоит атрибут; $E_{FS} \subseteq FS \times FS \times TR$ - подмножество связей полей, TR - тип связи; Θ - множество операторов получения атрибута (=,≠.<,≤,>,≥,+,-,*,/ и др.); MD - множество методов базы данных для преобразования полей (унарные операторы, встроенные функции,' пользовательские функции, агрегаты БД и т.д.); TPE – множество типов редакторов соответствующего типа поля; M_a - метаданные атрибута.

Доступный группе пользователей атрибут — это пара вида:

$$a' = \langle A, G_a \rangle, \quad (11)$$

где G_a - множество групп пользователей атрибута, причем $G_a \subset G$;

Полем fs называется пара:

$$fs = \langle TP, M_{fs} \rangle, \quad (12)$$

где $fs \in FS$; TP - множество типов полей системы; M_{fs} – метаданные поля.

Группой пользователей g называется пара:

$$g = \langle U, M_g \rangle, \quad (13)$$

где $g \in G$, а G — множество всех групп пользователей; U - множество пользователей группы, причем

$$U = \langle M_{u1}, M_{u2}, \ldots, M_{un} \rangle, \quad (14)$$

M_g - метаданные группы пользователей.

Пользователь может быть членом различных групп:

$$\exists u \left(u \in g_1 \wedge \ldots \wedge u \in g_n \right). \quad (15)$$

Бизнес-процессом *b* называется пара:

$$b = \langle P, M_b \rangle, \quad (16)$$

где P - множество модулей, задействованных в выполнении бизнес- процесса; M_b - метаданные БП.

Для всех объектов информационной модели представлено табличное описание структуры метаданных и отношений с другими объектами. Отношения основных метаэлементов системы представлены на рис. 35.

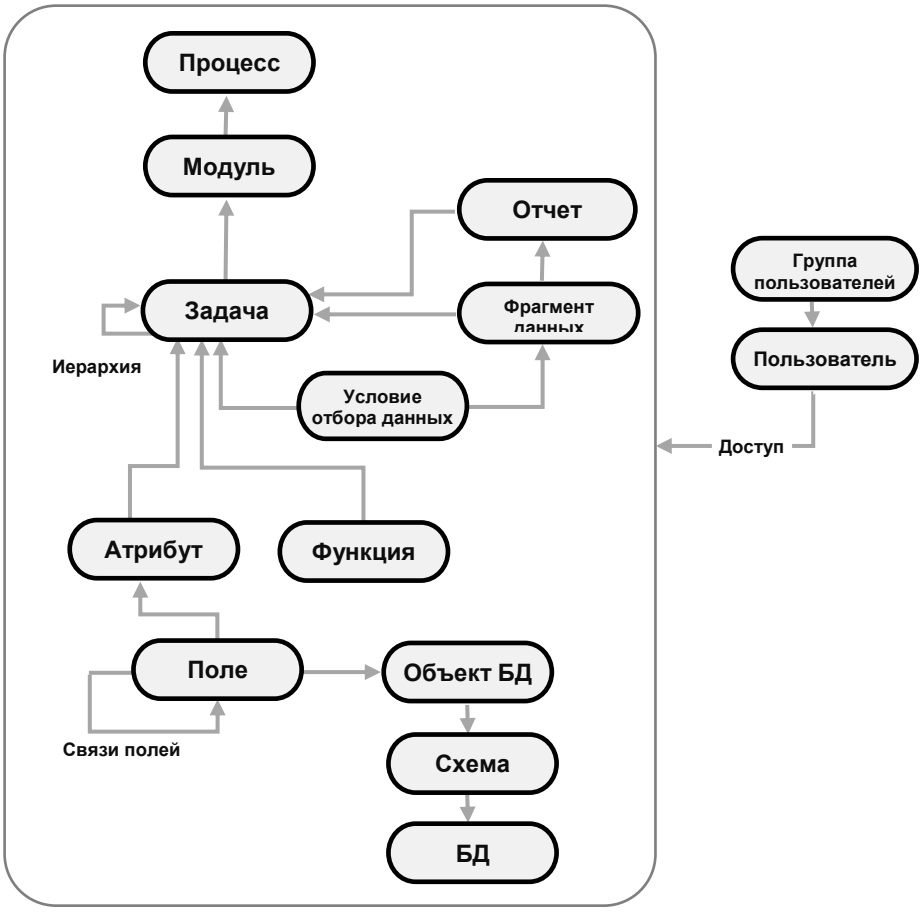

Рисунок 35 – Структура метаэлементов гибкой информационной системы

Представим основные положения проектирования и реализации представленной структуры:

1. Задача – совокупность сущностей и связей между ними, а также правила их отображения и взаимодействия.
2. Модуль – совокупность задач, объединенных в едином интерфейсе. Причем задачи могут повторяться в разных модулях:

$$\exists t\ (t \in p_1 \wedge \ldots \wedge t \in p_n). \quad (3.18)$$

3. Модули являются составными частями информационной системы. Система может дополняться новыми модулями и задачами.
4. Каждая задача имеет определенный набор *функций* доступных/недоступных различным группам пользователей. Набор функций пополняемый.
5. Каждая сущность, информация о которой хранится в БД — это *объект*, (экземпляр сущности), получаемый из одной/нескольких физических *таблиц* путем отбора и/или вычисления определенных атрибутов.
6. Каждая сущность уникальна (и вне БД) и имеет уникальный идентификатор (GUID), что позволяет «безболезненно» объединять данные одной структуры.
7. Сущность имеет *свойства* (строковые, числовые, временные, классифицируемые, константные, составные, калькуляционные и т. д.), которые описывают *атрибуты* задачи.
8. Две задачи могут иметь одинаковые атрибуты:

$$\exists A\ (A \in t_1 \wedge \ldots \wedge A \in t_n). \quad (17)$$

9. Атрибут может использоваться отдельно для фрагмента данных и отдельно для задачи:

$$\exists c \forall t \forall a\ (a \in c \wedge a \notin t) \vee \exists t \forall c \forall a\ (a \notin c \wedge a \in t). \quad (18)$$

10. Задача может обладать любым количеством атрибутов.
11. Сущности могут быть связаны между собой произвольным образом. Связь в некотором роде аналогична понятию ссылки на таблицу-справочник в традиционной реляционной модели БД и характеризуется

типом связи.

12. Описание каждой таблицы, поля и связи как минимум дублирует их структуру в БД:

$$\forall a \exists FS\,(FS \in a). \tag{19}$$

13. Представления могут содержать в себе одни и те же сущности.
14. Фрагмент или срез данных могут также становиться отдельными задачами.

Такая организация данных не зависит от предметной области и позволяет с ее помощью решать задачи практически любого бизнес-процесса.

Связь базы данных с программным приложением происходит в процессе перевода отображенной предметной области в описание базы данных – концептуальную схему. Этот процесс проходит в несколько этапов:

- Физическое хранилище. Совокупность реальных объектов БД (схем, таблиц и их связей, представлений, хранимых процедур/функций, типов, полей и).
- Логическое хранилище. Совокупность логических описаний объектов БД (таблиц, представлений, срезов, процедур/функций, типов, полей)
- Слой сопоставления. Этап сопоставления объектов логического хранилища и концептуального слоя.
- Концептуальный слой. На этом этапе на основе установленных или выработанных требований по эффективности, безопасности, ограничениям из таблиц, представлений и срезов формируются запросы для отображения сущностей и их свойств.
- Предметная область. Совокупность задач, формирующих единое информационное пространство из сущностей, их связей и операций над данными.

Глава 4 ПРЕДСТАВЛЕНИЕ ЭКСПЕРИМЕНТАЛЬНЫХ И РАСЧЕТНЫХ РЕЗУЛЬТАТОВ АПРОБАЦИИ

4.1 Технологии тестирования информационной системы

В проектах по разработке программного обеспечения (ПО) помимо основной задачи по реализации заявленной функциональности существует не менее важная задача по обеспечению качества ПО. Качество ПО (Software quality) — это совокупность характеристик программного обеспечения, относящихся к его способности удовлетворять установленные и предполагаемые потребности.

Одним из устоявшихся способов контроля качества является тестирование. Тестирование ПО (Software testing) — проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. Тестирование включает следующие этапы:

а) планирование работ (Test Management);

б) проектирование тестов путем ручной разработки или автоматической генерации (Test Design);

в) выполнение тестирования с получением результатов (Test Execution);

г) анализ полученных результатов выполнения с целью оценки качества ПО (Test Analysis).

Результат теста может сообщить разработчику, что ошибка присутствует в коде, тесте или в коде и тесте одновременно. Кроме этого, объект тестирования может выдавать большой объем выходных данных, требующий значительных усилий для анализа.

На каждом этапе жизненного цикла должны выполняться верификация и валидация проекта. Верификация (Verification) — это процесс оценки системы или её компонентов с целью определения удовлетворяют ли результаты текущего этапа разработки условиям, сформированным в начале этого этапа. Валидация (Validation) — это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе.

Тестирование как инструмент верификации и валидации является постоянным процессом и проводится на всех этапах жизненного цикла проекта. В ходе тестирования необходимо достичь следующих целей:

1. Повысить вероятность того, что разрабатываемое ПО будет работать правильно при любых обстоятельствах.

2. Повысить вероятность того, что разрабатываемое ПО будет соответствовать всем описанным требованиям.

3. Предоставить актуальную информацию о состоянии продукта на данный момент.

На этапе тестирования выполняется итеративный запуск тестовых наборов. Итерация тестирования запускается при изменении тестируемого объекта. Как правило, итерация состоит из следующих шагов.

1. Обновление тестовых наборов. Изменения тестируемого объекта (появление новой, изменение/исправление существующей или удаление устаревшей функциональности) требуют изменений тестовых наборов: добавление новых тестов, коррекция существующих или исключение устаревших тестов.

2. Приемочные испытания. Выполняется проверка работоспособности объекта тестирования. Как правило, на приемочных испытаниях используют не большое число позитивных тестов, проверяющих выполнение объектом своего предназначения. В случае неудачи объект возвращается на доработку, а итерация завершается.

3. Запуск основного набора тестов. Как правило, основной набор состоит из большого числа тестов, что приводит к значительным временным затратам на выполнение тестов. По результатам запуска формируется сводная ведомость с указанием перечня тестов, завершившихся с ошибкой.

4. Анализ результатов тестирования. Выполняется классификация найденных ошибок, общая оценка тестируемого объекта. По результатам анализа может быть

принято решение о доработке объекта тестирования и запуска следующей итерации или завершении этапа тестирования.

Так как этап тестирования является итеративным, количество итераций ограниченно доступными временными ресурсами (сроками сдачи проекта) и используемыми стандартами качества. Тестирование может быть закончено после выполнения итерации без ошибок, при достижении требуемого объема выполненных тестов без обнаружения ошибок или по истечении выделенных временных ресурсов. В случае наличия не исправленных ошибок в момент завершения этапа формируется итоговый список.

Выявление программных ошибок является сложной задачей. Программная ошибка (Software error) может не приводить к наблюдаемому сбою, а, например, порождать другую программную ошибку или переводить процесс работы в некорректное состояние. Сбой (Software failure) порождается наличием одного или нескольких дефектов (Software defect) — недостатков в компоненте или системе. Для выявления программных ошибок используются тестовые случаи или тесты.

Тестовым случаем (TestCase) называют документ, который описывает конкретные шаги, условия и параметры, необходимые для анализа реализации тестируемой функции. Каждый тест содержит три базовые части.

1. Предусловия (PreConditions) — шаги, которые переводят систему в состояние, пригодном для проведения проверки.

2. Описание теста (Description) — шаги, которые переводят систему из стояния в состояние. На основании полученного результата делается вывод о соответствии реализации заявленным требованиям.

3. Постусловия (PostConditions) — шаги, которые переводят систему в изначальное положение.

Проверка результата работы объекта тестирования выполняется на основе определения корректного состояния или эталонной модели результата. Эталонная модель определяется используемыми стандартами, спецификациями или

ожиданиями пользователя. Эталонная модель может быть представлена множеством различных способов:

а) неформальное представление того, «как ПО должно работать»;

б) формальная техническая спецификация;

в) набор тестовых примеров;

г) корректные результаты работы программы;

д) другая (априори корректная) реализация той же исходной спецификации.

Для проявления некорректных состояний необходимо создавать тесты, обладающие следующими характеристиками:

а) достижение (Reachibility) — тест должен выполнить место в исходном коде, где присутствует программная ошибка;

б) повреждение (Corruption) — при выполнении ошибки состояние программы должно испортиться с появлением сбоя;

в) распространение (Propagation) — сбой должен распространиться даль-ше и вызвать неудачу в работе ПО.

Проектирование и создание тестов, которые соответствуют определенным ранее критериям качества и целям тестирования, называется Тест дизайном (Test Design).

Для разработки тестов используются следующие методы:

а) на основе внутренней структуры объекта тестирования (белый ящик);

б) на основе требуемой функциональности (черный ящик).

Разработка тестов белым ящиком предполагает непосредственное участие автора-разработчика, наличие достаточных знаний в понимании логики программной реализации и может быть применена к небольшим объектам. Разработка тестов черным ящиком может быть выполнена без участия автора кода на основе имеющихся спецификаций, а тестированию может быть подвержен большой и сложный объект. Методы не являются взаимозаменяемыми, а дополняют друг друга для проведения качественного тестирования.

Существует возможность комбинации методов разработки тестов (серый ящик, Gray box testing). В этом случае создатель теста знает частично или полностью внутреннее устройство тестируемого объекта, но находится на уровне пользователя.

Серый ящик особенно удобен для проверки интерфейсов взаимодействия объектов (интеграционное тестирование).

Главной проблемой тестирования является определение того, достаточно

ли текущего количества тестов для вывода о правильности реализации системы, а также нахождения такого множества тестов, которые обладают таким свойством.

Для решения этой проблемы используют следующие методики:

а) эквивалентное разделение (Equivalence Partitioning);

б) анализ граничных значений (Boundary Value Analysis);

в) причина / следствие (Cause/Effect);

г) предугадывание ошибки (Error Guessing);

д) исчерпывающее тестирование (Exhaustive Testing). Число возможных комбинаций входных параметров может быть очень большим. Также число возможных путей следования внутри тестируемого объекта может быть очень большим. В силу ограниченности временных ресурсов, выделенных на проект в целом и этап тестирования в частности, выполнение полного тестирования невозможно. В таких случаях тесты объединяют в группы при выполнении следующих условий:

а) тесты, которые предназначены для тестирования одной и той же ошибки;

б) при выявлении ошибки в одном из тестов другие тесты, вероятнее всего, тоже это сделают;

в) при отсутствии ошибки в одном из тестов в других тестах, вероятнее всего, она также будет отсутствовать.

Тесты, объединенные в группу, называются эквивалентными, а сама группа — классом эквивалентности. Для проведения тестирования достаточно выбрать по одному представителю из классов эквивалентности.

Изменение поведения тестируемого объекта происходит при достижении внешних или внутренних граничных значений. Граничные значения принадлежат одному или нескольким классам эквивалентности или образуют собственный класс эквивалентности. В связи с этим тесты с участием граничных значений являются наилучшими кандидатами.

В ряде случаев необходимо проверить реализованные причинно-следственные связи, например, условия(причин) для получения ответа от ПО (следствие). Использование этой методики построения тестов позволяет проверить работу ПО в динамике (потоки данных, передача управления и т. д.). Использование накопленного опыта в области разработки и тестирования ПО позволяет проектировщику тестов предугадать места появления ошибок.

Исчерпывающее тестирование — это крайний случай. В пределах этой методики проверяются всевозможные комбинации входных значений.

Модульные тесты запускаются с использованием специальной программы-драйвера, выполняющего следующие функции в следующем порядке:

а) чтение входных параметров теста;

б) подготовка окружения модуля: заглушек и других вспомогательных инструментов;

в) запуск тестируемого модуля;

г) чтение результатов работы модуля.

Интеграционное тестирование (Integration testing) направлено на проверку взаимодействия между частями (модулями) приложения.

На этапе интеграционного тестирования выполняется поиск ошибок, связанных с трактовкой данных, реализацией интерфейса взаимодействия и совместимостью компонент приложения. Как правило, для интеграционного

тестирования применяется метод серого ящика: известны все характеристики взаимосвязей между модулями, но модули закрыты для анализа.

В ходе интеграционного тестирования выполняется объединение модулей в блоки. Существуют два основных подхода к проведению интеграции:

а) восходящая интеграция (Bottom Up Integration);

б) нисходящая интеграция (Top Down Integration).

Восходящая интеграция предполагает объединение модулей низкого уровня в группы, группы с группами или модулями и с получением в итоге целого приложения. Нисходящая интеграция предполагает последовательное присоединение модулей к группе, содержащей управляющий модуль.

Системное тестирование завершает проверку реализации приложения. В ходе системного тестирования проводится функциональное тестирование, а также характеристики разработанного ПО, в том числе устойчивость, производительность, надежность и безопасность.

Для системного тестирования применяется подход черного ящика: приложение рассматривается как единое целое, на вход подаются реальные данные, работа приложения анализируется по полученным результатам. На этапе системного тестирования выявляются ошибки, связанные с неправильной реализацией функций ПО, неправильным взаимодействием с другими системами, аппаратным обеспечением, неправильным распределением памяти, отсутствием корректного освобождения ресурсов и т. п. Источником данных выступают техническое задание на разработку приложения, спецификации на компоненты приложения и его окружения и используемые стандарты.

План тестирования (Test plan) — это основной документ этапа тестирования, который описывает работы по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.

Основные вопросы, которые план тестирования должен раскрыть:

1. Что необходимо тестировать: включает в себя абсолютно все аспекты ПО, которые могут быть протестированы.

2. Что будет тестироваться: подмножество пунктов из первого вопроса, которые будут протестированы. Из первого вопроса исключаются аспекты, исходя из анализа сроков, бюджета, приоритетов и пр. В идеальном случае множество первого и второго вопросов совпадают.

3. Как будет проходить тестирование: выбор стратегии тестирования (ручное тестирование, написание автоматических тестов, подбор групп пользователей для тестирования и др.).

4. Когда будет проходить тестирование: сроки тестирования для каждого компонента ПО.

5. Критерии окончания тестирования: результат, который должен быть получен в результате тестирования (отчет, список ошибок, каким способом представлены эти документы и др.).

План тестирования может создаваться либо как полноценный продукт, либо как инструмент. В первом случае требуются значительные ресурсы для его создания, но затем он может использоваться вне команды разработчиков. Обычно выполняется по какому-либо стандарту. Во втором случае в тестовый план включается только то, что помогает в организации процесса тестирования и выявлении ошибок. Все, что не отвечает этим задачам, избыточно.

Необходимость создания плана тестирования заключается в повышение качества продукта. Для этого ставятся следующие цели:

а) облегчение тестирования (контроль полноты тестирования и его эффективности, отсутствие повторяющихся тестов, поиск наилучших методов для тестирования);

б) организация взаимодействия между участниками команды, отвечающих за проведение тестирования;

в) удобная структура для организации, планирования и управления.

Процесс тестирования выделяется в отдельный процесс, поскольку он имеет свои задачи. Тестированию подлежат все разрабатываемые продукты: документация, программное обеспечение. Тестирование ПО включает в себя как тестирование качественных характеристик ПО, так и тестирование дистрибутива системы на различных платформах (на которых будет использоваться система), тестировании документации.

Процесс тестирования начинается раньше, чем процесс проектирования. После того, как в процессе анализа создаются функциональные и технические требования, и еще до того, как эти требования будут полностью согласованы, создается первая версия плана тестирования. Это необходимо для того, чтобы учесть затраты времени на тестирование при составлении плана жизненного цикла программного обеспечения, согласования даты выпуска. Кроме того, в некоторых случаях может потребоваться приобретение дополнительных инструментальных средств тестирования или разработка специализированного ПО.

В тестировании используются метрики для предсказания времени тестирования, ожидаемой продолжительности тестирования, оценки рисков. Оптимизация тестирования и отладки приложений представляется одной из наиболее актуальных задач. В отличие от этапа проектирования и разработки, длительность которых определяется требованиями к конечному продукту и производительностью, длительность отладки может быть установлена произвольно. То есть, может быть задано время отладки или условие (условия), при котором отладка заканчивается. При этом существующие модели строятся на основе предположения, что между временем отладки и количеством необнаруженных ошибок существует обратная зависимость.

Необходимо различать первичные и вторичные проявления ошибки. Первичные ошибки обнаруживаются в тексте программ и подлежат корректировке. Вторичные ошибки представляют собой искажение выходных результатов

исполнения программы, именно с вторичными ошибками сталкивается конечный пользователь на этапе эксплуатации программы. Следовательно, именно прогнозируемое количество вторичных ошибок должно быть критерием при принятии решения о продолжении или прекращении отладки. При этом ожидаемое количество вторичных ошибок определяется ожидаемым количеством первичных ошибок и структурой приложения.

Качество исходного кода может определяться разными критериями, одни из которых важны с точки зрения человека, другие со стороны техники. Форматирование исходного кода определяет удобство поддержки программы и привлечение к работе над ней других специалистов.

Кроме того, следует различать ошибки и недоработки. Недоработки не приводят к существенному ухудшению работы программы, и, в худшем случае, снижают эстетическое восприятие. Время, необходимое для обнаружения и исправления всех недоработок, может быть сопоставимо со временем разработки приложения.

Процесс тестирования позволить определить скорость загрузки контента, качество адаптивности верстки, отказоустойчивость ресурсов под нагрузкой, метрику кода и корректность работы информационной системы.

Test Cases — это набор условий, при которых тестировщик будет определять, удовлетворяется ли заранее определённое требование. Чтобы определить, что требование полностью выполняется, может потребоваться много вариантов тестирования. Часто варианты тестирования группируют в тестовые наборы.

UnitTest - процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы. Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

4.2 Тестирование функций разработанной системы

В процессе тестирования программного были проведены следующие виды тестов:

- функциональные;
- конфигурационные;
- тест сборки.

Функциональное тестирование представляет собой проверку того, выполняет ли программа те функции, которые предусмотрены техническим задание, а также степень точности выполнения проверяемых функций.

При конфигурационном тестировании проверяется, может ли созданный сайт работать на различных компьютерах и с различным установленным программным обеспечением.

Тест сборки является копией «дымового» тестирования, которое выполняется для подтверждения того, что после старта приложения, оно выполняет все свои основные функции. Данное тестирование выполняется программистами, после создания программы, и если программа не проходит эти тесты, то в дальнейшем тестировании нет смысла, так как эти тесты указывают на то, что программе не работает.

Примером того, что программе не прошла «smoke testing» является то, что она не запустилась, также этот тест не проходится по причине ошибки подключения к базе данных.

Тестовый случай представляет собой совокупность шагов, конкретных условий и параметров, которые необходимы для проверки разработанной системы. Для удобного представления тестовых сценарием он будут размещены в таблицах, которые представлены ниже.

Таблица 5 - Тест авторизации

Название:	Тест авторизации	
Функция:	Вход в программу	
Действие	Ожидаемый результат	Результат теста:
Предусловие:		
Откройте программу	Открытое главное окно программы	пройден
Шаги теста:		
Введите логин и пароль пользователя	Поля заполнены	пройден
Нажмите кнопку «Вход»	Открытие главного окна программы	пройден

Таблица 6 - Тест главного меню приложения

Название:	Тест меню программы	
Функция:	Переход в другие окна приложения	
Действие	Ожидаемый результат	Результат теста:
Предусловие:		
Откройте программу	Открытое главное окно программы	пройден
Шаги теста:		
Выберите пункты из предоставленного меню	Модули для открытия выбраны	пройден
Нажмите кнопку перехода в другой модуль	Переход на выбранный модуль осуществлен	пройден

Таблица 7 - Тест создания записи в модуле «Сущности»

Название:	Тест создания записи в модуле «Документооборот»	
Функция:	Запись в базу данных информации о задаче	
Действие	Ожидаемый результат	Результат теста:
Предусловие:		
Откройте программу	Открытое главное окно программы	пройден
Перейдите в пункт «Сущности»	Открытие подменю модуля «Сущности»	пройден
Шаги теста:		
Нажать пункт меню «Создать»	Открыта форма для создания сущности	пройден
Ввод данных в форму создания задачи	Данные добавлены	пройден
Нажать кнопку «Сохранить»	Задача сохранена	пройден

Таблица 8 - Тест удаления записи в модуле «Сущности»

Название:	Тест удаления задачи в модуле «Сущности»	
Функция:	Удаление задачи	
Действие	Ожидаемый результат	Результат теста:
Предусловие:		
Откройте программу	Открытое главное окно программы	пройден
Перейдите в модуль «Сущности»	Открытие окна «Сущности»	пройден
Шаги теста:		
Выберите задачу	Подсвечивание выбранной записи	пройден
Нажмите на кнопку «Удалить»	Удаление задачи	пройден

При конфигурационном тестировании определено, что созданная программа успешно функционирует на компьютерах под управлением разных операционных систем (табл. 9).

Таблица 9 - Работа приложения в разных версиях ОС семейства Windows

Параметр	Windows 7	Windows 8	Windows 10
Открытие главного модуля программы	+	+	+
Отображение меню программы	+	+	+
Выбор модуля программы	+	+	+
Отображение списка элементов из выбранного модуля	+	+	+
Добавление записей в выбранном модуле	+	+	+
Редактирование записей в выбранном модуле	+	+	+
Удаление записей в выбранном модуле	+	+	+

При проверке удобства пользования, определено, что программа понятна и удобна пользователя, при этом отвечает следующим параметрам:

- быстрое открытие модулей программы;
- подходящее стилевой оформление;
- удобное и понятное расположение пунктов навигации по системе;
- однотипные шрифты во всех модулях системы;
- отсутствие отвлекающих блоков и информации;
- удобное управление концептуальной структурой базы данных в модулях системы.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы были выполнены все поставленные в работе задачи:

- проведен анализ технологий разработки информационных систем с настраиваемой концептуальной моделью базы данных;
- рассмотрены особенности математического обеспечения информационных систем с настраиваемой концептуальной моделью данных;
- разработан проект информационной системы с настраиваемой концептуальной моделью данных;
- проведена экспериментальную проверку математического и программного обеспечения информационной системы с настраиваемой концептуальной моделью данных.

На современном этапе развития технологий проектирования и создания информационных систем популярность набирают новые гибки информационные системы, которые построены на базе предметно-независимых моделей данных. Основой таких систем, как и традиционных является база данных, но с внедрением уровня метаданных, которые описывают сущности предметного уровня, которые формируют концептуальных уровень базы данных. Подобные системы посредством возможностей изменения концептуальной модели базы данных могут приобретать разные конфигурации и за счет этого расширять свой функционал и возможности конечного назначения для разных предметных областей.

Для информационных систем с настраиваемой концептуальной моделью данных наиболее удобным вариантом является создание в внедрение динамических баз данных, которые обладают физической структурой данных, которая является независимой относительно хранимой информации – структурно независимые базы данных. Динамические базы данных позволяют вносить изменения в пользовательские метаданные, которые содержаться внутри базы данных, в

отличии от классического подхода, когда этим данные находятся на системном уровне. Минимальным и достаточным набором элементов, которые могут характеризовать новый объект являются: сущность, атрибут, тип данных, типовые связи. Уровень реализации структурно-независимой базы данных включает указанные элементы, при этом каждому из них соответствует отдельная реляционная таблица, в которую посредством разработанных инструментов вносятся изменения, таким образом управляя структурой данных информационной системы.

Проектируемая система является клиент-серверным приложением, для построения которого необходим серверный язык программирования. Существует множество серверных языков, которые можно использовать для программирования веб-приложений, наиболее подходящим для реализации информационной системы является PHP, который позволяет реализовать высоко функциональные веб-приложения, в том числе и приложения с настраиваемой концептуальной моделью данных.

Как СУБД для гибкой информационной системы выбрана MySQL. Средой разработки базы данных послужит программа MySQL Workbench, как среда создания исходного кода выбрана программа PHPStorm. Функция веб-сервера для разработки будет реализована с помощью решения – OpenServer. Созданная информационная система с настраиваемой концептуальной моделью данных позволит пользователям удобно подстраивать информационную систему под новые требования предметной области в связи с изменениями бизнес-процессов.

В качестве архитектурного решения для разработки информационной системы с настраиваемой концептуальной моделью данных выбран шаблон проектирования Модель-Представление-Контроллер (MVC). Данный шаблон построен на основе сохранения представления данных отдельно от методов, которые взаимодействуют с данными. Подобная схема приложений позволяет

реализовать функционал за счет модулей, что определяет возможность быстрого изменения функционала программы.

Для реализации приложения с настраиваемой концептуальной моделью данных в качестве архитектурного основания выбран фреймворк Yii2, который реализует паттерн MVC. Yii2 представляет собой высокопроизводительный компонентный PHP фреймворк, предназначенный для быстрой разработки современных веб приложений. Для организации кода Yii2 использует архитектурный паттерн MVC (Model-View-Controller).

В процессе проектирования базы данных для приложения была разработана модель, представляющую собой, мета уровень, который описывает таблицы, атрибуты и связи баз данных. Созданная модель включает ряд таблиц, которые позволяют описать сущности разных предметных областей. После создания сущностей посредством специально разработанного приложения осуществляется взаимодействие с таблицами в базе данных.

В процессе разработки алгоритмов были созданы следующие: регистрация пользователей, авторизация пользователей, добавление сущностей, добавление атрибутов, добавление связей. Указанные алгоритмы являются основой функционирования программы и обеспечивают ее функционал, который включает: регистрацию и авторизацию, добавление таблиц, добавление атрибутов, добавление связей, генерацию форм для ввода данных, введение данных в созданные таблицы, управление данными в базе данных. При обосновании математического обеспечения для решения задачи определено математическое описание метаданных информационной системы, в перечень которых могут входить такие элементы: модули, задачи, функции, отчеты, фрагменты данных, сущности, атрибуты, схемы, пользователи, группы пользователей. Описание представленных элементов позволяет проводить отображение предметной области в виде настраиваемой концептуальной модели, которая управляется специально разработанным программных обеспечением.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Басангова Г. Ю. Модели описания данных / Г. Ю. Басангова // Межвузовский сб. научных статей. Вып. 3 / Под ред. А.А. Кабанова. – СПб.: СПб ун-т МВД России, 2008. – С. 21.

2. Вайсфельд М. Объектно-ориентированное мышление / М. Вайсфельд. - СПб.: Питер, 2014. - 304 с.

3. Вересников Ю. К. О надежности систем обработки информации / Ю. К. Вересников //Актуальные проблемы современной науки. - 2011. - № 2 (58). - С. 193-195.

4. Власенко А. А. Разработка адаптивной системы дистанционного обучения в сфере информационных технологий / А. А. Власенко // Автореферат диссертации на соискание ученой степени кандидата технических наук. Воронеж. - 2014. – 14 с.

5. Все о фреймворке Kohana [Электронный ресурс]. - Режим доступа: http://kohanaframework.su/ (дата обращения 14.05.2017)

6. Горбаченко В. И. Проектирование информационных систем с CAERwin Modeling Suite 7.3: учебное пособие / В. И. Горбаченко, Г. Ф. Убиенных, Г. В. Бобрышева – Пенза: Изд-во ПГУ, 2012. – 154 с

7. Городилов А. А. Математическая модель динамических структур данных автоматизированной информационной системы / А. А. Городилов // Вестник Сибирского государственного аэрокосмического университета имени академика М. Ф. Решетнева. – 2009. – С. 90 – 95.

8. Дронов, В. А. PHP, MySQL, HTML5 и CSS3. Разработка современных динамических Web–сайтов / В. А. Дронов. – СПб.: BHV, 2016. – 688 с.

9. Дюбуа П. MySQL. Сборник рецептов / Дюбуа П.; Пер. с англ. – СПб: Символ–Плюс, 2006. – 1056 с.

10. Жуков Р. А. СУБД с открытым исходным кодом: возможность применения алгоритмов распараллеливания / Р. А. Жуков // Технические науки. – 2015. – № 1–2 (35–36). – С. 20 – 21.

11. Завдстра М. PHP: объекты, шаблоны и методики программирования, 4-е изд. Пер. с англ. / М. Завдстра - М.: ООО "И.Д. Вильяме", 2015. - 576 с.

12. Завозкин С.Ю. Информационное обеспечение интеграции информационных систем на основе системы электронного документооборота. Кемеровский государственный университет. Диссертация на соискание ученой степени кандидата технических наук Кемерово. 2007. 218 с.

13. Зудилова, Т. В. Web-программирование HTML / Т. В. Зудилова, М. Л. Буркова - СПб: НИУ ИТМО, 2012.– 70 с.

14. Изобретая информационные системы будущего. Теория и практика. — Сургут, 2017. — 192 с.

15. Когаловский М.Р. Перспективные технологии информационных систем. - М.: ДМК Пресс, 2003. - 288 С.

16. Кондраков Д.Ю., Игнатович Р.В. Объектно-ориентированная модель учётных баз данных. Сборник научных трудов СевКавГТУ. Серия "Экономические науки" № 1. 2000.

17. Коцюба И. Ю. Основы проектирования информационных систем. Учебное пособие / И. Ю. Коцюба, А. В. Чунаев, А. Н. Шиков // СПб: Университет ИТМО, 2015. – 206 с.

18. Кошколов Д. В. Решения для адаптивной верстки WEB-интерфейсов [Электронный ресурс] / Д. В. Кошколов. - VIII Международная студенческая электронная научная конференция. «Студенческий научный форум» - 2016. – Режим доступа: https://www.scienceforum.ru/2016/1411/22804 (дата обращения: 20.01.2019).

19. Кузнецов, М. В. PHP на примерах / М. В. Кузнецов, И. В. Симдянов. – СПб.: BHV, 2012. – 400 с.

20. Курзыбова Я. В. Средства создания динамических web-сайтов: учеб. пособие / Я. В. Курзыбова. – Иркутск: Изд-во ИГУ, 2011. – 121 с.

21. Макаров А. С. Yii. Сборник рецептов. - М.: ДМК Пресс, 2013. – 18 с.

22. Маклаков С. В. Моделирование бизнес-процессов с AIIFusion Process Modeler. - Диалог-МИФИ, 2004. - 240 с.

23. Маклафлин Б. PHP и MySQL. Исчерпывающее руководство [Текст] / Б. Маклафлин. – СПб.: Питер, 2014. – 544 с.

24. Маторин С. И. Информационные системы: Учебно-практическое пособие / С. И. Маторин, О. А. Зимовец–Белгород: Изд-во НИУ БелГУ, 2012. – 231 с.

25. Мейерт Д. О. Небольшая книга о HTML/CSS фреймворках [Текст] / Д. О. Мейерт // Орейли. – 2015. – 30 с.

26. Мельниченко Д .В. Исследование логических проблем юзабилити сайтов и анализ существующих решений [Электронный ресурс] / Д.В. Мельниченко, О.Ф. Абрамова // Современная техника и технологии. - 2015. - № 1. - С. Режим http://technology.snauka.ru/2015/01/5360.

27. Мирошниченко Г. А. Реляционные базы данных. Практические приемы оптимальных решений / Г. А. Мирошниченко. - "БХВ-Петербург", 2005. – 400 с.

28. Михайлов П. В. База данных: цель создания, требования / П. В. Михайлов // Межвузовский сб. научных статей. Вып. 3 / Под ред. А.А. Кабанова. – СПб.: СПб ун-т МВД России, 2008. – С. 6.

29. Официальный сайт CSS-фреймворка Bootstrap. Режим доступа: http://getbootstrap.com (дата обращения: 20.01.2019).

30. Охотникова Е. С. Математическое моделирование, алгоритмизация и программная реализация адаптивных информационных систем (на примере систем электронного обучения) / Е. С. Охотникова. Диссертация на соискание ученой степени кандидата технических наук. Тюмень. - 2012. - 187 с.

31. Охотникова Е. С. Математическое моделирование, алгоритмизация и программная реализация адаптивных информационных систем: автореф. дис. ... канд. техн. наук: 05.13.18 / Е. С. Охотникова. – Тюмень, 2012. – 22 с.

32. Павлов А. И. Инструментальное средство для создания гибких информационно-аналитических систем / А. И. Павлов // Автореферат диссертации на соискание ученой степени кандидата технических наук. Иркутск. - 2005. – 20 с.

33. Петров В. Н. Информационные системы. - СПб.: Питер, 2003. - 688С.

34. Приступа В. В. Подход к выбору технологии создания динамического сайта [Электронный ресурс] / В. В. Приступа. - Режим доступа: http://www.repository.hneu.edu.ua/jspui/bitstream/ 123456789/6097/29/sect3_Prystupa.pdf (дата обращения 8.01.2019).

35. Рогозов Ю. И. Архитектура информационных систем: учебное пособие / Ю. И. Рогозов, А. С. Свиридов, С. А. Кучеров. – Ростов-на-Дону: Изд-во ЮФУ, 2014. – 117 с.

36. Рогозов Ю. И. Понятие метасистемы как методологической основы создания системы / Ю. И. Рогозов // Промышленные АСУ и контроллеры. – 2013. – No 2. – С. 39 – 47.

37. Рогозов Ю. И. Построение классификации характеристик программного обеспечения с целью идентификации понятий предметной области как характеристик / Ю. И. Рогозов, А. С. Свиридов, А. А. Дегтярев // Информатизация и связь. – 2011. – No3. – С. 80 - 83.

38. Рогозов Ю. И. Применение оптимизационных моделей информационных потоков для построения CASE-средств / Ю. И. Рогозов, С. А. Бутенков, Д. С. Бутенков и др. // Известия Таганрогского государственного радиотехнического университета. – 2006.– No 3. – С. 54.

39. Рогозов Ю. И. Систематизация моделей жизненного цикла информационных систем в рамках схемы J.Zakhman / Ю. И. Рогозов, С. А. Бутенков,

Н. С. Горбань, А. С. Свиридов // Известия Южного федерального университета. Технические науки. – 2008. – Т. 78, – No 1. – С. 68-72.

40. Рогозов Ю. И. Системный подход к созданию метода разработки информационных объектов на основе метамоделей / Ю. И. Рогозов // Информатизация и связь. – 2011. – No7. – С. 57-62.

41. Рогозов Ю. И., Свиридов А.С., Кучеров С.А. Архитектура информационных систем: учебное пособие. – Ростов-на-Дону: Изд-во ЮФУ, 2014. – 117 с.

42. Росс В. С. Создание сайтов: HTML, CSS, PHP, MySQL [Текст]. Учебное пособие, ч. 2 / В. С. Росс — МГДД(Ю)Т, М. - 2011 – 68 с.

43. Рудаков А. В. Технология разработки программных продуктов: учеб. пособие для студ. / А. В. Рудаков. - 5-е изд., стер. - М.: Издательский центр «Академия», 2010.-208с.

44. Русскоязычное сообщество Yii - YiiFramework.ru [Электронный ресурс]. - Режим доступа: http://yiiframework.ru/ (дата обращения 14. 01.2019).

45. Самарев Р. С. Создание простейших HTML-страниц, валидаторы кода. Каскадные таблицы стилей CSS: методические указания к выполнению практикума № 1 и лабораторной работы № 1 по дисциплинам «Языки интернет-программирования» и «Практикум по интернет-программированию» / Р. С. Самарев. — Москва: Издательство МГТУ им. Н. Э. Баумана, 2015. — 39 с.

46. Тенцер А.Н. База данных - хранилище объектов. КомпьютерПресс 8'2001 г. 14 с.

47. Титков А. В. Создание веб-приложений: учебное пособие / А. В. Титков, С. А. Черепанов. — Томск: Эль Контент, 2014. — 72 с.

48. Устимов А. И. Функциональные возможности современных CSS-фреймворков [Электронный ресурс] / А. И. Устимов // Молодежный научно-технический вестник. – Режим доступа: http://sntbul.bmstu.ru/doc/758306.html (дата обращения: 20. 01.2019).

49. Филлипов С. А. Основы современного веб-программирования: Учебное пособие / С. А. Филлипов. – М.: НИЯУ МИФИ, 2011. – 160 с.

50. Флэнаган Д. JavaScript. Подробное руководство, 6-е издание: пер. с англ. СПб.: Символ-Плюс, 2012. 1080 с. [Flanagan D. JavaScript: The Definitive Guide, 6th ed. O'Reilly Media, Inc., 2011. 1078 p.].

51. Целуйко Д. С. Математическое и программное обеспечение информационных систем с настраиваемой концептуальной моделью данных / Д. С. Целуйко. Диссертация на соискание ученой степени кандидата технических наук– Москва. – 2018. – 128 с.

52. Целуйко Д. С. Математическое и программное обеспечение информационных систем с настраиваемой концептуальной моделью данных: диссертация ... кандидата технических наук: 05.13.11 / Целуйко Дмитрий Сергеевич; [Место защиты: Моск. гос. ун-т приборостроения и информатики].- Москва, 2011.- 136 с.

53. Чебоксаров В. А. Анализ фреймворков для создания сайта кафедры автоматики и компьютерных системы / В. А. Чебоксаров, И. И. Савенко // XII Международная научно-практической конференция студентов, аспирантов и молодых ученых. - Томск, 12-14 ноября 2014 г. – С. 92 – 93.

54. Чертовский В. Д. Математическое описание и компьютерная реализация модели адаптивной автоматизированной системы управления производством / В. Д. Чертовский // Информационно-управляющие системы. - №1. – 2017. – С. 106 – 113.

55. Чиркин Е. С. Некоторые проблемы автоматизированного извлечения данных из веб-страниц / Е. С. Чиркин // Интернет и современное общество: тр. XVI Всерос. объедин. конф. "Интернет и соврем. о-во", 9 – 11 окт. 2013 г., Санкт-Петербург / С.-Петерб. нац. исслед. ун-т информ. технологий, механики и оптики. – СПб., 2013. – С. 291–294.

56. Шалаев А. А. Модель и алгоритмы самомодификации адаптивных информационных систем / А. А. Шалаев // Автореферат диссертации на соискание ученой степени кандидата технических наук. Пенза. - 2016. – 22 с.

57. Шемсединов Т. Г. Введение мета-уровня [Электронный ресурс] / Т. Г. Шемсединов // Режим доступа: http://blog.meta- systems.com.ua/2011/01/blog-post.html (дата обращения 30.09.2018).

58. Шинкевич А. В. Современные технологии создания web-сайтов / А. В. Шинкевич // Проблемы и перспективы современной науки: сб. ст. участников V Респ. науч.-практ. семинара молодых ученых, Минск, 28 нояб. 2014 г. / редколл: В. В. Гедранович [и др.]; Минский ун-т управления. – Минск: Минский университет управления, 2015. – С. 110–114.

59. Шпаков М.В. Разработка интеллектуальных геоинформационных систем на основе настраиваемой объектной модели предметной области. Диссертация на соискание ученой степени кандидата технических наук Санкт-Петербургский институт информатики и автоматизации РАН. 2004

60. Янк К. PHP и MySQL. От новичка к профессионалу / К. Янк. - М.: Эксмо, 2013. - 384 с.

61. Agrawal R. Papakonstantinou, H. Garcia-Molina, and J. Widom. / R. Agrawal, N. Gehani, J. Srinivasan // Object Exchange Across Heterogeneous Information Sources. Proceedings of the Eleventh International Conference on Data Engineering. – Taipei, Taiwan. - 1995. – P. 251-260.

62. Alrifai M. Distributed management of concurrent web service transactions / M. Alrifai, P. Dolog, W.-T. Balke // IEEE Transactions on Services Computing (TSC). – 2009. – V. 2, No 4. – P. 289-302.

63. AntiPatterns / W. Brown, R. Malveau, H. I. McCormick, T. Mowbray. – N. Y.: John Wiley, 1998. - 156 pp.

64. Atom [Электронный ресурс] / Режим доступа: http://soft.mydiv.net/win/download-Atom.html (дата обращения - 16.10.2018).

65. Chen P. P., The Entity-Relationship Model: Toward a Unified View of Data. ACM Trans. on Database Systems, 1976, Vol. 1, – P 9-36.

66. Cox B. J. Object oriented programming: An Evolutionary Approach / B. J. Cox // ill Reading, Mass.: Addison-Wesley Pub. Co., 1. – 1986 – 274 pp.

67. Date C. J. Database Design and Relational Theory: Normal Forms and All That Jazz (Theory in Practice) / C. J. Date // O'Reilly Media, 2012. – 276 pp.

68. Douglas C. Schmidt. Why Software Reuse has Failed and How to Make It Work for You [Электронный ресурс] / C. Douglas // C++ Report magazine, January, 1999. - Режим доступа: https://www.dre.vanderbilt.edu /~schmidt/reuse-lessons.html (дата обращения 30.09.2018).

69. Fekete A. Allocating isolation levels to transac- tions / A. Fekete // Proc. of PODS. – 2005. – P. 206-215.

70. Fisher A. S. CASE: Using Software Development Tools / A. S. Fisher. – N.Y.: J. Willey and Sons Inc., 1988. – 287 pp.

71. ISO 9126 (ГОСТ Р ИСО / МЭК 9126-93) Информационная техноло- гия. Оценка программного продукта. Характеристики качества и руководство по их применению. Введ. 1994-06-30. – М.: Стандартинформ, 1994. – 12 с.

72. MagentoTM creates huge success with enterprise e-commerce platform & community built on Zend Framework [Электронный ресурс] / Режим доступа: http://framework.zend.com/casestudies/ZFCaseStudy-Magento.pdf (дата обращения 01.10.2018).

73. Medvidovic N. Round-Trip Software Engineering Using UML: From Architecture to Design and Back. Proceedings of the 2nd Workshop on Object-Oriented Reengineering (WOOR) / N. Medvidovic, A. Egyed, D. Rosenblum // Toulouse, Francею - September 1999. - P. 1-8,

74. Microsoft Dynamic CRM [Электронный ресурс] / Режим доступа: http://www.microsoft.com/rus/dynamics/crm/ (дата обращения 01.10.2018).

75. Nadkarni. P.: An Introduction to Entity-Attribute-Value Design for Generic Clinical Study Data Management Systems / P. Nadkarni // Center for Medical Informatics, Yale University Medical School. Режим доступа: http://med.yale.edu/ (дата обращения 01.10.2018).

76. Notepad++ - бесплатный редактор Html, PHP и другого кода с подсветкой синтаксиса, а также обзор его плагинов и возможностей [Электронный ресурс] / Режим доступа: http://ktonanovenkogo.ru/vokrug-da-okolo/programs/notepad-plus-plus-tekstovyj-redaktor-podsvetkoj-sintaksisa-skachat-ustanovit-nastroit.
html (дата обращения 23.10.2018).

77. Open Server. Лучший инструмент для разработки под Windows [Электронный ресурс] / Режим доступа: http://open-server.ru/ (21.10.2018).

78. Pattern-Oriented Software Architecture. Volume 1: A System of Patterns / F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M.Stal. – N.Y.: John Wiley, 1996. – 476 p.

79. Pattern-Oriented Software Architecture. Volume 2: Patterns For Concurrent And Networked Objects / D. Schmidt, M. Stal, H. Rohnert, F. Buschmann. – N. Y.: John Wiley, 2000. – 633 p.

80. PhpStorm – интегрированная среда разработки на PHP [Электронный ресурс] / Режим доступа: http://jetbrains.ru/products/phpstorm/ (Дата обращения: 15.10.2018).

81. Saito Y. Optimistic replication / Y. Saito, M. Shapiro // ACM Computing Surveys. – 2005. – V. 37, No 1. – P. 42-81.

82. Simon A. The Integrated CASE Tools Handbook / A. Simon. – Van Nostrand Reinhold/Intertext 1993 – 330 p.

83. Stead W.W. A Chartless Record–Is It Adequate? / W. W. Stead, W. E. Hammond, M. J. Straube. - Proceedings of the Annual Symposium on Computer Application in Medical Care 7 (1982-11-02). - P. 89–94.

84. Wegner P. Research Directions in Software Technology. Proceedings Of The 3rd International Conference On Software Engineering. 1978. - P. 12 – 18.

85. Weikum G. Transactional informa- tion systems: theory, algorithms, and the practice of concurrency control and recovery / G. Weikum, G. Vossen. – San Fran- cisco: Morgan Kaufmann Publishers, 2001. – 880 p.

86. Ye. Y. Designing for Participation in Socio-Technical Soft-ware Systems / Y. Ye, G. Fisher // In: Stephanidis, C. (ed.) Proceedings of 4th International Conference on Universal Access in Human-Computer Interaction, Bei-jing, China. – P. 312–321.