

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки	09.04.04 «Программная инженерия»
Магистерская программа	Программное обеспечение вычислительной техники и автоматизированных систем

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

**Модельные представления и алгоритмы управления
метаданными в оперативных информационных системах**

Студент	_____	Панов Дмитрий Евгеньевич
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	Шибанов С.В.
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	_____	Такташкин Д.В.
	(подпись, дата)	(фамилия, инициалы)
Рецензент	_____	Васин Л.А.
(должность, место работы)	(подпись, дата)	(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой	_____	Макарычев П.П.
	(подпись)	(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК	_____	Попова Н.А.
	(подпись)	(фамилия, инициалы)

Пенза, 2020

Содержание

Введение	4
1 Актуальность применения метаданных в оперативных информационных системах	8
1.1 Понятие, свойства и роль метаданных в информационных системах	8
1.1.2 Классификация метаданных	14
1.1.3 Жизненный цикл метаданных	15
1.2 Понятие и свойства информационных систем	18
1.2.1 Классификация информационных систем	19
1.3 Обзор архитектур оперативных информационных систем основанных на метаданных	21
1.3.1 Анализ архитектур, применяемых для построения информационных систем.....	22
1.3.2 Анализ подходов к проектированию информационных систем оперативного уровня на базе многоуровневой архитектуры.....	28
Выводы	32
2 Структура метаданных и операции над метаданными в оперативных информационных системах	34
2.1 Формализованное представление реляционной базы данных оперативной информационной системы.....	34
2.2 Структура метаданных оперативной информационной системы.....	36
2.2.1 Описание компонентов структуры метаданных оперативной информационной системы	39
2.3 Построение универсальной модели данных оперативной информационной системы	43

2.4 Операции над метаданными оперативной информационной системы	50
Выводы	54
3 Программные средства для построения приложений в составе оперативно-аналитической информационной системы	55
3.1 Назначение, архитектура и принципы функционирования оперативно-аналитической информационной системы	55
3.2 Программная реализация приложений концептуальной оперативно- аналитической информационной системы	58
Выводы	65
Заключение	67
Список использованных источников	69
Приложение А	71

Введение

Метаданные начали использоваться задолго до появления компьютерных систем и введения данного термина в научно-техническую лексику. В компьютерных системах, метаданные начали использоваться уже на ранней стадии развития, но во времена создания языков программирования данного термина ещё не существовало. Однако, в исходном коде программы уже содержались описания типов, которые в свою очередь уже являлись метаданными. На основе этих типов, например, компилятор обеспечивает требуемое для этих данных распределение оперативной памяти, определяет область возможных действий, которые можно провести с каждым конкретным типом, либо их совокупностью [1].

Рождение технологий баз данных и информационно-поисковых систем, также потребовало использования метаданных. При проектировании БД, например, метаданными будут являться ограничения целостности, полномочия пользователей и т.д.

Но несмотря на то, что метаданные уже стали неотъемлемой частью множества категорий информационных систем, их интероперабельность даже в рамках одной предметной области весьма ограничена. Главным образом по причине того, что не существует чёткой согласованности метаданных, как в рамках отдельно взятой предметной области, так и, особенно, в какой-либо существующей универсальной метамодели, которая могла бы послужить основой для создания различных систем в разных предметных областях, объединенных едиными алгоритмами взаимодействия с метамodelью. Это позволило бы повысить степень интегрируемости разноплановых систем между собой для объединения их в программные комплексы, с целью наращивания функционала.

Использование универсальной метамодели в основе таких систем, помимо вышеописанных плюсов, также позволило бы снизить расходы бизнеса на переориентирование обслуживающих систем на другие отрасли (например, во

времена финансовых кризисов: когда спрос на предоставляемые услуги упал и возникает необходимость в переориентировании бизнеса на предоставление других услуг или выпуск другой продукции).

С точки зрения бизнеса, возрастает как время на разработку, так и её стоимость. Возникает необходимость в поиске и найме узкоспециализированных специалистов для создания новой информационной системы, но уже с учётом новых требований, когда можно было бы просто нанять специалистов, например, в области бухгалтерского учёта и предоставить им возможность сконфигурировать систему и интерфейс под себя без дополнительного обращения к программистам. Также, существует такая альтернатива, как покупка лицензии у сторонних компаний на использование уже готовых продуктов. Но, как уже было сказано выше – такой продукт не будет учитывать специфики работы конкретного бизнеса, а для обслуживания таких систем понадобилось бы нанимать и содержать дополнительный персонал.

Цель диссертации состоит в изучении существующих метамоделей, типов метаданных, в информационных системах оперативного уровня и применении подходов для проектирования унифицированных метамоделей. На основе изученных методов, разработать подход для создания систем с динамическим отображением семантической модели предметной области в существующие реляционные базы данных на основе универсальной метамоделей. Также, разработать подход, который позволит динамически перестраивать интерфейс системы на основе изменяющейся метамоделей.

Задачи исследования. Основная последовательность выполняемых задач:

- анализ существующих подходов к построению систем на основе метамоделей;
- изучение альтернативных систем, разработанных на основе метамоделей;

- разработка алгоритма управления метаданными для возможности изменения универсальной метамодели системы;
- разработка концептуальной модели информационной системы с использованием предложенной метамодели и алгоритмов для её модификации.

Объектом исследования был выбран широкий пласт разнородных систем оперативного уровня, с которыми взаимодействует обслуживающий персонал (оператор системы).

Предметом исследования является внедрение в архитектуру систем оперативного уровня: универсальных, независимых от предметной области метамodelей и алгоритмов взаимодействия с ними, для получения возможности переконфигурирования системы оператором с учётом своих рабочих задач без знаний программирования.

Методы исследования. В соответствии с поставленными целями исследования, работа включает в себя современные методы изучения, связанные с различного рода метамodelями, входящими в их основу метаданных и существующих подходов в управлении метаданными.

Научная новизна работы заключается в следующем:

- универсальная модель метаданных позволяет описывать некоторую абстрактную метамodelь без применения языков программирования высокого уровня, тем самым позволяет автоматизировать и упростить настройку системы под нужды конкретного заказчика;
- разработанные алгоритмы управления метаданными позволят наиболее эффективно строить собственные хранилища данных, не прибегая к непосредственному взаимодействию с исходным кодом системы;
- внедрение предложенного подхода позволит многократно сократить как временные затраты на разработку информационной системы, так и финансовые, а также кадровые;
- подобный подход, также позволяет исключить появление ошибок в непосредственно в программном коде и вывести их на новый, семантический

уровень, основанный на конкретной предметной области (ошибки конфигурирования метамодели персоналом);

- помимо разработки серверной составляющей, появляется возможность конфигурировать пользовательский интерфейс на основе предпочтений сотрудников, а также специфики конкретного бизнеса.

Практическая значимость исследования. На основе концептуальной модели информационной системы, в дальнейшем можно создать программное обеспечение, которое позволит персоналу оперативного уровня непосредственно влиять на разработку ПО, нивелируя обращения в службу поддержки конкретного программного продукта и конфигурировать информационную систему отталкиваясь только от семантики предметной области, игнорируя семантику конкретного языка программирования.

Основные положения, выносимые на защиту:

- универсальная метамодель, которая содержит в себе все базовые атрибуты для дальнейшего конфигурирования информационной системы;
- архитектура и протоколы для взаимодействия компонентов системы между собой;
- созданные алгоритмы управления метаданными для совершения манипуляций над метамodelью информационной системы;
- анализ эффективности работы информационной системы оперативного уровня на основе универсальной метамодели.

Публикации. По теме диссертационного исследования опубликована одна научная статья.

Структура и объем работы. Работа состоит из введения, трех тематических глав, заключения, списка литературы и приложения.

1 Актуальность применения метаданных в оперативных информационных системах

1.1 Понятие, свойства и роль метаданных в информационных системах

Понятие метаданных варьируется в зависимости от источника. Например, в источнике [2] вводится следующее понятие:

Метаданные (Metadata) – данные о данных: каталоги, справочники, реестры, базы метаданных, содержащие сведения о составе данных, содержании, статусе, происхождении, местонахождении, качестве, форматах и формах представления, условиях доступа, приобретения и использования, авторских, имущественных и смежных с ними правах на данные и др. [2].

Также, в источнике [3] даётся общее представление о структуре метаданных, и их предназначении:

Метаданные – слабоструктурированные данные, как правило, соответствующие согласованным моделям, обеспечивающим операционную интероперабельность в неоднородной среде [3].

Исходя из вышеописанных определений, можно сказать о том, что в общем случае к метаданным относятся:

- информация об информации;
- данные, представляющие собой характеристики описываемых сущностей для целей их идентификации, поиска, оценки, управления ими, набор допустимых структурированных описаний, которые доступны в явном виде и предназначение которых может помочь найти объект;
- данные из более общей формальной системы, описывающей заданную систему данных.

Примером метаданных, описанных в первом случае, будет являться: имя автора правки в тексте. Этот термин в широком смысле слова используется для любой информации о данных: именах таблиц, колонок в таблице в реляционных

базах данных, номер версии в файле программы (то есть как информативная часть в бинарном файле) и т.п.

Термин, описанный во втором случае, используется в контексте поиска объектов, сущностей, ресурсов.

Самой обширной областью применения метаданных являются базы данных. Метаданные, в такого рода системах позволяют организовать работу с данными, хранящимися в БД и представляют собой: таблицы, ограничения целостности, триггеры, а также позволяют отделять сегменты базы данных на основе политики прав доступа и т.д.

Метаданными текстовых публикаций в электронных библиотечных каталогах служат также их библиографические описания и аннотации, рубрики, сформированные по тематикам, к которым также относятся и содержание данных публикаций. Так, например, для описания данных, в нескольких системах, реализующих схожую бизнес-логику, могут использоваться метаданные, очень сильно различающиеся между собой, как по описываемым типам данных, так и по способам их обработки и применения, что накладывает определённого рода ограничения на повторное использование уже готовых наборов метаданных в схожих системах. Примером такого рода являются СУБД. Многие из них несовместимы между собой даже на уровне используемых типов данных, что сильно затрудняет перенос системы с одной СУБД на другую. В этом случае, создание определённого рода «прослойки» из метаданных, которая бы в ходе работы программы выполняла преобразования одних типов данных в другие.

Но несмотря на явные преимущества использования метаданных в современном мире информационных технологий существует проблема поддержки информационных систем. На данном этапе, в течение всего жизненного цикла программного обеспечения вносятся различного рода правки (такие, как: исправления ошибок или добавление нового функционала), что влечёт за собой целую цепочку, зачастую, лишних действий. Например, при

добавлении новой функции в информационную систему, приходится независимо от изменения структуры функциональной части ПО, ещё и переделывать интерфейс. Это происходит опосредовано от основного процесса наращивания функционала программного продукта и на данном этапе могут возникнуть различного рода ошибки. Причины могут быть абсолютно различные: от банальной невнимательности до неполного/неправильного описания новых функций в документации к информационной системе. Внедрение метаданных значительно упрощает этот процесс и позволяет даже генерировать интерфейс «на лету» уже на основе внесённых изменений, беря за основу некоторые ключевые описательные элементы внутри информационной системы. Также, метаданные позволяют снизить временные (а следовательно – и финансовые) издержки на разработку ПО, т.к. сокращается как само время на написания кода, так и время, отведённое на его отладку.

Таким образом, с ходом технологической эволюции, метаданные всё глубже вплетались практически во все сферы деятельности человека. Они сильно облегчили работу с данными. Вопрос о способах обработки новых типов систем, которые неизбежно генерируются в ходе технологического прогресса не утратит своей актуальности, т.к. неизбежно возникнет необходимость создания новых способов взаимодействия с данными. А метаданные являются идеальным инструментом решения подобного рода проблем.

Метаданные могут описывать как неструктурированные (текст, даты, цифры, факты и т.д.), так и структурированные данные (базы данных, адреса URI и др.).

Роль метаданных зависит от следующих ключевых факторов:

- особенностей системы;
- характера описываемых ресурсов;
- потребностей пользователя.

Ниже приведены некоторые свойства метаданных:

- относительность;

- многоуровневость;
- автономность/встроенность в описываемый ресурс;
- формализованность представляемых метаданных.

1. Относительность. Представляет собой разделение информации на данные и метаданные является относительным. В разных ситуациях и даже в рамках одной предметной области одни и те же наборы данных могут играть разные роли. Например: гипертекстовая ссылка. Она содержит в себе часть информации из основного текста, но при этом может быть указателем на другие статьи или части текста, которые могут дополнять или освещать определённые подробности той или иной статьи.

2. Многоуровневость. Известно, что метаданные по своей сути, являются некой абстракцией над основным набором данных (и является метамоделью данных). При этом, метаданные, как и данные, являются информационными ресурсами. А это значит, что для них может быть сформирован собственный набор метаданных, который будет представлять собой набор мета-метаданных по отношению к исходным объектам. Теоретически, данная иерархия ничем не ограничена и можно также создать набор мета-мета-метаданных и т.д. Примером, демонстрирующим данное свойство является – модель DOM (Document Object Model – Объектная модель документа), с помощью которой веб-браузер делает анализ структур веб-документов и составляет дерево узлов, каждый узел которого имеет в своём составе атрибут, текстовый или графический элемент. DOM универсален для всех браузеров и является системой метаданных, описывающих общее дерево узлов объектов любых HTML, XHTML, JavaScript файлов, которые в свою очередь являются метаданными для работы с файлами, каркасами страниц и т.д.

3. Автономность/встроенность в описываемый ресурс. Метаданные могут быть автономными (отчужденными) от описываемого ресурса или встроенными в него. Примеры автономных метаданных: схемы баз данных (они отчуждены от описываемого ими наполнения базы данных); DTD, описывающие тип XML-

документов и хранимые в Вебе вне конкретных экземпляров XML-документов; полнотекстовый индекс коллекции текстовых документов электронной библиотеки. Примеры встроенных метаданных: HTML-разметка веб-страницы; разметка веб-страницы средствами микроформатов; DTD, специфицированное в теле XML-документа; аннотация статьи, содержащаяся в ее тексте; глоссарий, содержащийся в тексте спецификации официального стандарта. Информационные ресурсы со встроенными описывающими их метаданными естественно называть самоописываемыми.

4. Формализованность представляемых метаданных. Для представления метаданных могут служить в различной степени формализованные выразительные средства – естественные языки, полужформальные языки, например, набор элементов метаданных «дублинского ядра», а также формальные языки в дедуктивных базах данных описываемых средствами логики.

Исходя из рассмотренных свойств, можно сделать вывод, что список задач, решаемых с помощью метаданных очень широк, а конкретные функции зависят от технологий, на которых базируется информационная система, функциональности системы, используемых структур данных и т.д. В полной мере описать все функции метаданных не представляется возможным, т.к. границы областей их применения до сих пор не выявлены. Ниже приведён примерный перечень функций метаданных, отражающий вышеописанные свойства.

1. Описание моделей предметной области. Является основной задачей метаданных. С разработки моделей и начинается проектирования информационных систем. Для спецификации моделей используются языки концептуального моделирования различного уровня формальности: диаграммы UML или различные формальные языки представления знаний, основанные на математической логике.

2. Описание источников данных. Эти функции выполняют схемы БД, определения типа XML и JSON документов, схемы репозитория, онтологии и т.д. Метаданных такого вида представлены соответствующими языками (SQL, XML, JSON, OWL и т.д.) [3].

3. Описание семантики информационных источников/объектов. Метаданные, описывающие семантику, называются семантическими. Такие метаданные добавляются в ходе семантического аннотирования – придания уже существующим данным семантической значимости (например, микроформаты в HTML-коде, которые считываются поисковыми системами и интерпретируются для последующей обработки).

4. Описание данных на разных уровнях информационной архитектуры. Примером этой функции может послужить ранее упомянутая интерпретация HTML/JS данных сайта в дерево узлов DOM. На уровне пользователя – полноценный веб-интерфейс, на уровне передачи данных – исходный код. На уровне архитектуры браузера – дерево объектов DOM. Каждый из уровней является метаданными для вышестоящего уровня в иерархии интерпретации данных.

5. Идентификация описываемых ресурсов. Метаданные описывают: какие входящие атрибуты будут идентифицировать искомую информацию. Примером таких метаданных являются ключи в таблицах БД. Они однозначно идентифицируют эти строки или совокупности строк. Для интернет-страниц такими идентификаторами являются адреса URI/URL: они позволяют так же однозначно идентифицировать искомую страницу и обеспечить взаимодействие пользователя с элементами этой страницы.

6. Управление структурированными данными. Примером проявления этой функции метаданных являются правила взаимосвязей между таблицами, права доступа к ним для различных пользователей/групп пользователей. Это свойство обеспечивает правильное функционирование схем СУБД.

7. Верификация данных. С помощью определённых структур метаданных можно верифицировать поступающую информацию. Примером является использование препроцессора XML для проверки целостности структуры XML-документов. Также, такие структуры обеспечивают правильность хранения данных в СУБД. Например: нельзя записать в поле Integer значение String, т.к. при сравнении входных данных с ожидаемым результатом, ядро СУБД, используя определённые механизмы верификации на основе метаданных, выдаст ошибку и не позволит произвести запись неправильных значений, чтобы предотвратить нарушение структуры базы данных.

Исходя из рассмотренных свойств и функций метаданных, напрашивается закономерный вывод о целесообразности их использования при проектировании современных информационных систем. Это позволит в некоторой мере преодолеть основные архитектурные ограничения, которые, в свою очередь являются неотъемлемой частью используемых информационных систем. А также, расширить варианты использования уже накопленных данных для их интеграции в системы различных уровней.

1.1.2 Классификация метаданных

Существует множество подходов к классификации метаданных. Рассмотренные ранее свойства и функции тоже в своём роде являются классификациями по свойствам и функциям, выполняемым в системе. Но, как правило, группировок по данным признакам бывает недостаточно.

Дополнительно, метаданные можно подразделить:

- по уровням семантической абстракции;
- по уровням информационной архитектуры;
- по ролям, выполняемым в системе.

Наиболее популярной является именно классификация по ролям. В работе [4] представлена следующая классификация метаданных:

- административные метаданные;

- описательные метаданные;
- метаданные сохранения;
- технические метаданные;
- метаданные об использовании.

1. Административные метаданные. Служат для управления и администрирования электронными коллекциями и другими информационными ресурсами. Содержат в себе данные о местоположении, правах доступа.

2. Описательные метаданные. Используются для идентификации и описания свойств объектов и их коллекций. Например: сведения об авторах, версии ПО, используемых библиотеках, пользовательские индексы, аннотации.

3. Метаданные сохранения. Предназначены для управления сохранением данных. Содержат сведения о физическом расположении записываемых данных, перечне необходимых действий производимых над данными.

4. Технические метаданные. Примером таких метаданных является техническая документация к проекту, журналы логов, идентификационная информация об оборудовании.

5. Метаданные об использовании. Описывают правила использования данных. Эти правила могут зависеть от контекста использования и изменяться в зависимости от сценария использования.

1.1.3 Жизненный цикл метаданных

В последнее время, метаданные всё чаще становятся объектом для различного рода исследований. Но многим остаётся неясно: на каких этапах разработки они задействуются. И для большего понимания значимости метаданных для информационных систем, на рисунке 1 изображена схема жизненного цикла метаданных в информационных системах любого уровня.

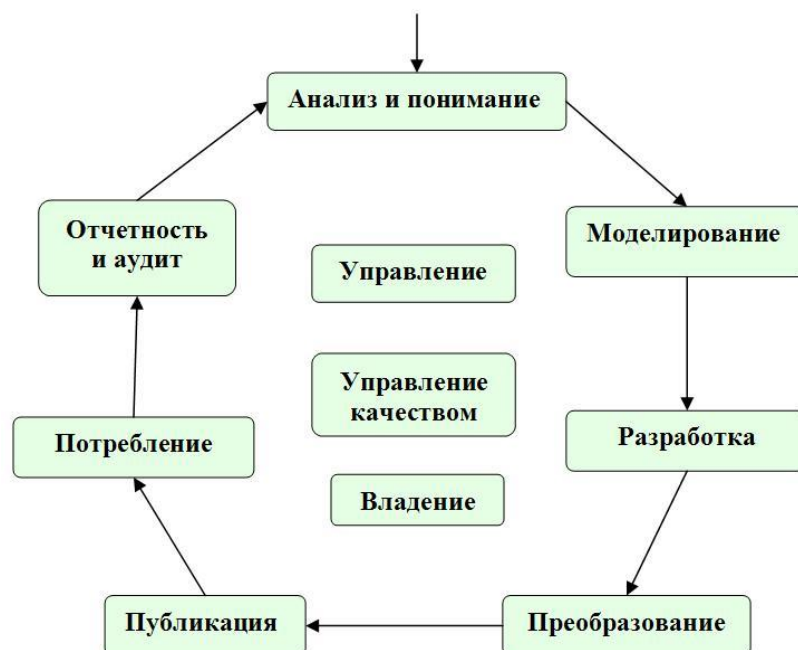


Рисунок 1 – Жизненный цикл метаданных в информационных системах

Анализ и понимание. Включают определение структур данных, понимание смысла и содержания входных данных, выявление связей между таблицами БД, анализ зависимостей и связей информации, исследование данных для их интеграции [5].

Моделирование. Подразумевает выявление схем объединения данных, выявление и отображение взаимосвязей в метаданных, моделирование структур данных и схем объединения данных, анализ влияния и синхронизации между моделями.

Разработка. Предполагает создание новых метаданных. Разработка обеспечивает коллективное создание и управление словарем параметров, поддержку функций информационных систем, усвоение потоков трансформации и доставки данных.

Преобразование. Заключается в автоматизации трансформации данных, связывании источников и БД с помощью определенных правил.

Публикация. Извещает заинтересованных пользователей о существовании искомых метаданных и их расположении. Публикация предоставляет

унифицированный механизм размещения метаданных и оповещения об обновлениях.

Владение. Позволяет определять права использования метаданных. Использование метаданных осуществляется сотрудниками или информационными системами. Управление, выполняемое руководителем, включает в себя модификацию, расширение метаданных и контроль доступа.

Потребление. Это поиск, визуальная навигация и отображение метаданных и их взаимосвязей, доступ к метаданным, их интеграция, импорт и экспорт; анализ влияния изменений.

Управление качеством. Метаданных решает задачи проверки разнородных данных в рамках их интеграции, повышения качества информационных ресурсов, мониторинга качества входных данных и позволяет устранять проблемы со структурами данных и их пригодностью до того, как они повлияют на проект.

Отчетность и аудит. Предполагают определение функций подготовки, исполнения отчетов по расписанию, форматирования результатов отчета, сохранения и просмотра версий отчета. Результаты аудита должны использоваться для анализа и понимания на следующем витке жизненного цикла метаданных.

Управление метаданными. Состоит в организации доступа к шаблонам, отчетам и результатам, навигации по объектам метаданных; определении прав и ответственности за поддержание качества и актуальности метаданных.

Большинство этапов из ЖЦ метаданных перекликаются с реально протекающими бизнес-процессами внутри крупных компаний. Из чего можно сделать вывод о том, что внедрение метаданных помогает в полной мере раскрыть все тонкости предметной области и построить программное обеспечение, которое будет максимально гибко и эффективно выполнять поставленные задачи.

1.2 Понятие и свойства информационных систем

Информационные системы призваны решать широкий спектр задач: от автоматизации документооборота, до контроля рынка биржевых котировок и моментального реагирования на его колебания. В зависимости от предназначения, ИС комплектуется необходимым набором метаданных, описывающим конкретную предметную область, для выполнения своих функций.

ИС не должна выполнять «автоматизацию ради автоматизации». Её внедрение должно быть целесообразно и экономически обосновано. В некоторых случаях, такая система может заменить целый отдел сотрудников, но иногда, для её обслуживания возникает необходимость найма дорогостоящих узкоспециализированных специалистов, что может только увеличить финансовую нагрузку на компанию.

Данные системы, обычно, являются вспомогательным инструментом для достижения определённых целей. ИС выполняет роль «дирижёра» с точки зрения манипуляций с информацией. Для этого ей необходимо достаточно эффективно взаимодействовать с серверами СУБД, иметь возможность коммуникации с другими ИС, а также обладать аналитическим функционалом.

В общем случае, информационная система должна обладать следующими свойствами:

- 1) структура ИС, её функциональное назначение должны согласовываться с предметной областью, в которой используется ИС;
- 2) ИС, в зависимости уровня, должна обладать соответствующими инструментами для агрегации, обработки и анализа информации;
- 3) интерфейс ИС должен быть интуитивно понятен, т.к. обслуживающий такую систему персонал, зачастую, не обладает квалификацией в области компьютерных технологий;
- 4) ИС использует некоторую сеть передачи данных, как для обмена данными между узлами ИС, так и для обмена ими с пользователем.

1.2.1 Классификация информационных систем

Тип информационной системы зависит от того, чьи интересы она обслуживает и на каком уровне управления.

Ниже приведён краткий обзор систем в зависимости от уровня управления:

- стратегические;
- тактические;
- оперативные.

1. Информационная система стратегического уровня – информационная система, обеспечивающая поддержку принятия решений по реализации стратегических перспективных целей развития организации. ИС данного типа помогают высшему звену управленцев решать неструктурированные задачи, осуществлять долгосрочное планирование. Они призваны создать общую среду компьютерной и телекоммуникационной поддержки решений в неожиданно возникающих ситуациях [6]. Как правило, такие системы оперируют сводными кластерами метаданных в определённой области.

2. Информационные системы тактического уровня. Задача подобного рода информационных систем – интеграция новых сведений в организацию и помощь обеспечению обработки бумажных документов. На этом этапе, можно выделить два типа ИС:

- управленческие;
- системы поддержки принятия решений.

3. Управленческие ИС. Данный класс систем имеет крайне небольшие аналитические возможности. Основное их назначение состоит в отслеживании строго структурированных сводных типовых отчётов. Информация поступает из ИС оперативного уровня.

4. Системы поддержки принятия решений. Такие системы обслуживают частично структурированные задачи, результаты которых трудно спрогнозировать заранее.

5. Информационные системы оперативного уровня. Назначение ИС на этом уровне отвечать на запросы о текущем состоянии и отслеживать поток сделок в фирме, что соответствует оперативному управлению. Чтобы с этим справляться: ИС должна быть легкодоступной, непрерывной действующей, гибкой и предоставлять точную информацию.

Задачи, цели и источники информации на оперативном уровне заранее определены и в высокой степени структурированы. Решение запрограммировано в соответствии с алгоритмом под каждую определённую задачу.

Информационная система оперативного уровня является связующим звеном между фирмой и внешней средой. Системы данного уровня – основной поставщик информации для остальных типов информационных систем в организации, так как содержит и оперативную и архивную информацию.

На рисунке 2 показан один из возможных вариантов классификации информационных систем по функциональному признаку с учётом уровней управления и уровней квалификации персонала. Также, на рисунке видно, что чем выше по значимости уровень управления, тем меньше объём работ, выполняемых специалистом и менеджером с помощью информационной системы.

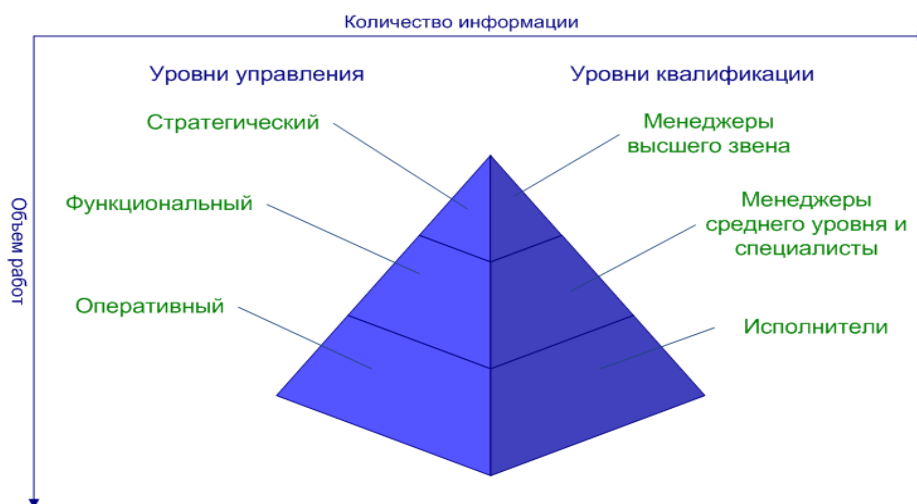


Рисунок 2 – Типы информационных систем с учётом уровней управления и квалификация персонала

Однако, при этом возрастают сложность и интеллектуальные возможности информационной системы и её роль в принятии менеджером решений. Любой уровень нуждается в информации из всех функциональных систем, но в разных объёмах и с разной степенью обобщения.

Основание пирамиды составляют информационные системы, с помощью которых сотрудники-исполнители занимаются операционной обработкой данных, а менеджеры низшего звена – оперативным управлением.

Наверху пирамиды, на уровне стратегического управления – информационные системы изменяют свою роль и становятся стратегическими, поддерживающими деятельность менеджеров высшего звена по принятию решений в условиях плохой структурированности поставленных задач.

1.3 Обзор архитектур оперативных информационных систем основанных на метаданных

Существует множество подходов для создания систем на оперативном уровне управления. Но отсутствуют какая-либо стандартизация, которая бы позволяла безболезненно сращивать разные типы систем, либо агрегировать данные из однотипных систем, но написанных с использованием отличающихся архитектур.

Применение определенных архитектурных подходов вкупе с метамоделями, позволяет наращивать слои абстракции и тем самым отделять компоненты системы от их реализации. Использование некоторых подходов позволяет, при наличии заранее заложенных правил интерпретации – и вовсе автоматически генерировать реализацию на нужном языке программирования для определенных компонентов системы.

1.3.1 Анализ архитектур, применяемых для построения информационных систем

Существует два основных типа архитектур:

- одноуровневая;
- многоуровневая.

Одноуровневая архитектура – тип архитектуры, который характеризуется следующими отличительными признаками:

- представление и логика приложения могут быть не отделены;
- лёгкость в построении приложения, т.к. нет отдельной серверной части и нет взаимодействия по сети;
- свобода выбора в реализации хранилища данных, т.к. приложение не взаимодействует с другими приложениями, нет и необходимости в соблюдении стандартов обмена данными (можно хранить и обрабатывать хоть в двоичном виде);
- лёгкость отладки, т.к. отсутствует необходимость в верификации данных, полученных от других систем.

Данный подход был доминирующим до начала XXI века. Большинство приложений создавались для частного использования, а взаимодействие через интернет было доступно далеко не всем. Приложения распространялись на физических носителях и не было необходимости докачивать обновления напрямую от разработчика; приходилось покупать носители со всеми нужными обновлениями.

Классическим примером приложений, построенных на данной архитектуре, являются оффлайновые клиентские приложения небольшие программы, выполняющие некоторые функции локально, без подключения к интернету. На данной архитектуре невозможно построить полноценное веб-приложение, так как исключается вынос бизнес-логики за пределы монолитной архитектуры.

Ряд недостатков, присущих данному типу архитектуры ПО, напрямую проистекает из её преимуществ:

- сложность в наращивании функционала;
- бизнес-логика жестко завязана на конкретное представление;
- сложность во внедрении метаописания для объектов из-за угрозы смешивания метамодели системы с функциональными компонентами;
- отсутствие стандартизации хранения данных: разработчик мог использовать собственные закрытые форматы данных и формировать хранилище данных по своему усмотрению, а система-конкурент даже при реализации такого же функционала могла оперировать абсолютно другим набором форматов представления и хранения данных, несовместимых с системами конкурентов.

Из-за жёсткой привязки реализации к конкретному языку программирования, архитектура перенимает некоторые особенности, присущие этому языку. Например: система хранения данных будет отличаться в каждой реализации из-за отличающихся подходов по работе с коллекциями различных ЯП. Таким образом, ядро системы будет претерпевать структурные изменения от реализации к реализации.

Внедрение метаописаний в системы, построенные на базе данной архитектуры затруднено и зачастую – нецелесообразно. В основном, из-за «плотности» компоновки бизнес-логики и данных, описанной выше, а также необходимости постоянной корректировки метаописания из-за проблемы зависимости реализации от используемых инструментов.

В настоящее время, данный тип систем можно встретить в различного рода специализированном программном обеспечении, таких как САД системы, различные текстовые, графические редакторы и т.д.

Область информационных технологий стремительно развивалась. Развивалась и компьютерная индустрия в целом. Возникла необходимость расширения функционала существующих систем и внедрения в них новых

технологических решений. К тому моменту, количество проприетарных систем было запредельно велико, а их кодовая база, уже не позволяла с такой же скоростью наращивать функционал. Помимо всего прочего, из-за повсеместного распространения интернета и увеличения автоматизации производств, возникла потребность в широком внедрении нового типа архитектур, которые назывались многоуровневыми.

Был выведен ряд требований, которым, системы на базе одноуровневой архитектуры уже не удовлетворяли:

- возможность взаимодействия через сеть;
- выделение и обобщение бизнес логики на отдельный сервер;
- введение новых видов СУБД, а также выделение таких систем на отдельные сервера;
- возможность централизованного сбора информации и управления различными компонентами системы;
- возможность объединения разнородных систем в программные комплексы, тем самым, увеличивая спектр выполняемых задач.

Общая схема построения системы на базе данной архитектуры представлена на рисунке 3.



Рисунок 3 – Общая схема построения системы на базе трехуровневой архитектуры

Как представлено на рисунке, логика приложения была отделена от представления, тем самым была ограничена их зависимость друг от друга. Снизилась затраты на оборудование рабочих станций, т.к. снизился

требования к накопителям информации и больше не требовалось большого количество дискового пространства для разворачивания подобного типа систем.

Многоуровневая архитектура значительно расширяет возможности для взаимной интеграции разнородных систем между собой, а также позволяет беспрепятственно и практически без ограничений производить масштабирование системы. Для реализации данных задач, данная архитектура была поделена ещё на 2 архитектурных подуровня:

- монолитная;
- микросервисная.

Монолитная архитектура. Практически не отличается от стандартной многоуровневой архитектуры по своему устройству и представляет собой всю ту же схему, представленную на рисунке 3. Она состоит из интерфейса, некоторой бизнес-логики, работающей на сервере, и хранилища данных, с которыми этот сервер взаимодействует.

Микросервисная архитектура. В свою очередь, значительно модифицирует стандартную многоуровневую архитектуру. В ней также отслеживается чёткое разделение функциональных компонентов, но помимо всего прочего появляется возможность их дублирования и/или физического распределения по различным серверам приложений.

Современный подход построения веб-приложений на микросервисной архитектуре является более гибким, нежели если строить систему на монолитной архитектуре. Можно выделить ряд преимуществ данной архитектуры:

- повышается отказоустойчивость системы за счёт дублирования компонентов. В случае сбоя, появляется возможность перезагрузки отдельно взятого микросервиса, а не всей ИС;
- можно атомарно обновлять отдельно взятые компоненты системы (микросервисы), не затрагивая всё остальное;

- повышается устойчивость высоконагруженных систем за счёт более точного дозирования нагрузок по однородным узлам;
- упрощается наращивание функционала для уже существующей системы.

Но у данного подхода есть и недостатки:

- сложность в разработке подобного рода систем;
- высокая зависимость быстродействия микросервисов от используемых протоколов взаимодействия (Websockets, REST, cookies, JWT и т.д.).

Сравнение этих архитектур приведено ниже, на рисунке 4.

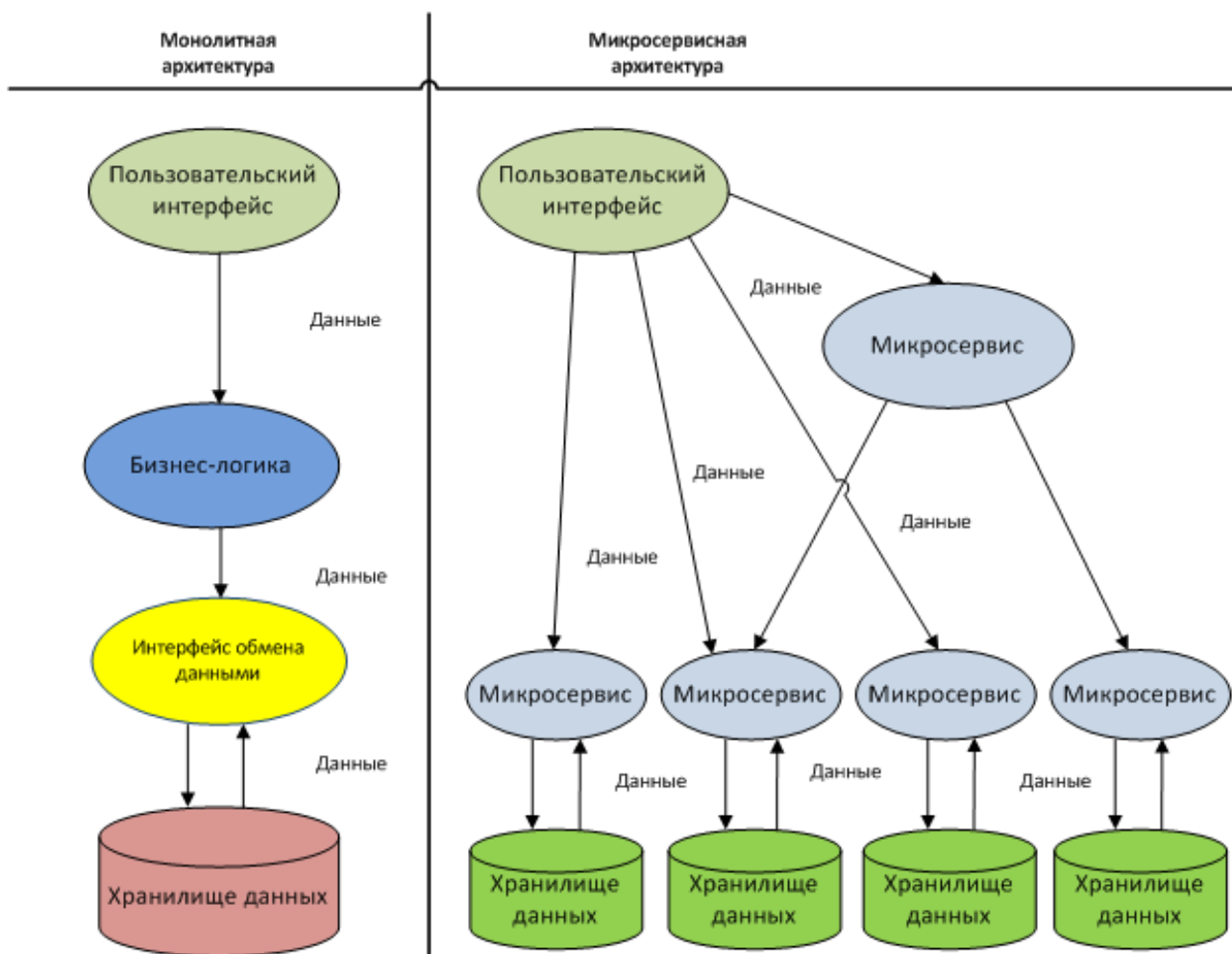


Рисунок 4 – Сравнение монолитной и микросервисной архитектур

Помимо выноса основной бизнес-логики на сервер и оптимизации работы с данными за счёт микросервисной архитектуры, также, встаёт вопрос о перераспределении нагрузки между клиентом и сервером.

Существует два варианта перераспределения:

- операции агрегации, обработки и преобразования информации, а также отрисовки компонентов интерфейса осуществляется на стороне клиента;
- клиент собирает информацию о выполнении необходимых действий по преобразованию информации в системе и лишь отрисовывает заранее и собранные сервером компоненты интерфейса.

В первом случае, следует говорить о «толстом» клиенте и «тонком» сервере. Данный подход подразумевает вынос части логики приложения на сторону клиента. Серверу же, в основном, отводится роль «менеджера» хранилища данных. В отдельных случаях, сервер занимается, например, нейросетевой обработкой, а результат отдаёт клиенту для дальнейших манипуляций с данными.

Во втором случае, говорят о «тонком» клиенте и «толстом» сервере. Данный подход подразумевает вынос всей бизнес-логики приложения на сторону сервера. Клиенту отводится роль «агрегатора» данных. Пользователь производит определённые манипуляции в интерфейсе и вводит необходимые данные. Сервер обрабатывает, полученные от клиента команды и данные, производит необходимые модификации над данными, а результат выполнения команд возвращает клиенту. С использованием данного подхода работает большая часть интернет-ресурсов или облачных информационных систем.

Таким образом, исходя из рассмотренных архитектурных подходов; их плюсов и минусов – наилучшим вариантом для построения современной системы оперативного уровня является использование микросервисной многоуровневой архитектуры с «тонким» клиентом.

1.3.2 Анализ подходов к проектированию информационных систем оперативного уровня на базе многоуровневой архитектуры

Внедрение многоуровневой архитектуры, спровоцировало появление новых подходов к проектированию информационных систем. Они конкретизировали и в значительной степени расширяли данную модель, адаптируя её для каждой конкретной системы, максимизируя производительность и оптимизируя затраты на проектирование и интеграцию с другими системами. Также, это способствовало и появлению новых протоколов коммуникации между системами (например, RESTful), стандартизации форматов данных, которыми должны были обмениваться системы.

Так как вопрос применяемых технологий уже решался, то количество систем на их базе начало неуклонно расти. Помимо всего прочего, росли и объёмы накапливаемой и обрабатываемой информации.

Существует два основных подхода к проектированию ИС на базе многоуровневой архитектуры:

- Model-Driven-Development (MDD);
- Database-Centric-Development (DCD).

Подход MDD (Разработка, управляемая моделями) декларирует такой стиль разработки, при котором модели являются основными объектами разработки. Из данных моделей, при необходимости генерируется код и другие объекты. Модель служит нотацией и метамоделью.

Нотация представляет собой совокупность графических элементов, которые должны интерпретировать люди для лучшего понимания устройства компонентов системы.

Метамодель представляет собой понятия, которыми оперирует модель и фиксирует информацию в виде метаданных, которые обрабатываются отдельными соответствующими инструментами [7].

В данном случае, модель – абстрактное описание ПО, которое скрывает информацию о некоторых аспектах с целью предоставления упрощенного

описания. Также, модель может выступать в качестве исходного объекта для разработки, если она фиксирует информацию о форме, с помощью которой люди интерпретируют модель, либо производится обработка инструментами [8].

Примером реализации данного подхода является концепция Domain Models – это такой подход к построению системы, когда объекты из реальной предметной области являются метаописанием объектов информационной системы. Использование данной концепции позволяет приблизить описание объектов на языке программирования к реальным объектам предметной области не теряя семантической значимости, а также более обобщённо манипулировать данными объектами в рамках бизнес-логики приложения.

Подход DCD (разработка, основанная на метаданных). При использовании данного подхода, базы данных играют решающую роль. В этом подходе имеет место предпочтение построения архитектуры данных, нежели архитектуры приложения. Например, характеристика архитектуры как "ориентированной на базу данных" может означать любую комбинацию представленных ниже параметров:

- использование стандартной, универсальной реляционной системы управления базами данных;
- использование динамической логики, управляемой таблицами;
- использование хранимых процедур, выполняемых на серверах баз данных;
- использование общей базы данных в качестве основы для связи между параллельными процессами в распределенных вычислительных приложениях;
- общая корпоративная архитектура.

1. Использование стандартной, универсальной реляционной системы управления базами данных, вместо настроенных в памяти или основанных на файлах структур данных и методов доступа. С развитием сложного

программного обеспечения СУБД, большая часть которого либо бесплатна, либо входит в состав операционной системы, разработчики приложений стали все больше полагаться на стандартные инструменты баз данных, особенно для быстрой разработки приложений.

2. Использование динамической логики, управляемой таблицами, вместо логики, воплощенной в ранее скомпилированных программах. Использование табличной структуры, то есть поведения, которое в значительной степени диктуется содержимым базы данных, позволяет программам быть более простыми и гибкими. Эта возможность является центральной особенностью динамических языков программирования.

3. Использование хранимых процедур, выполняемых на серверах баз данных. Такой подход позволяет снизить зависимость от логики, выполняемой на серверах приложений среднего уровня в многоуровневой архитектуре. Вопрос о распределении логики приложения между бэкендом или уровнем СУБД, является предметом постоянных дискуссий. Например, Toon Korpelaars представляет подробный анализ альтернативных архитектур на базе Oracle, которые различаются по размещению бизнес-логики. Автор предполагает, что подход, ориентированный на базу данных, имеет практические преимущества с точки зрения простоты разработки и дальнейшего сопровождения информационной системы [9].

4. Использование общей базы данных в качестве основы для связи между параллельными процессами в распределенных вычислительных приложениях. Вместо прямой межпроцессной связи с помощью функций передачи сообщений и ориентированного на сообщения промежуточного программного обеспечения. Потенциальное преимущество архитектуры, ориентированной на базы данных в распределенных приложениях, заключается в том, что она упрощает проектирование, используя обработку транзакций и индексирование, предоставляемые СУБД, для достижения высокой степени надежности, производительности и емкости. Например, корпорация Base One, в своей

работе [10] описывает ориентированную на базу данных распределенную вычислительную архитектуру для кластерных вычислений и объясняет, как эта конструкция обеспечивает повышенную безопасность, отказоустойчивость и масштабируемость [10].

5. Общая корпоративная архитектура. Данный параметр отдает предпочтение общим моделям данных [11] и не позволяет каждому приложению иметь свою собственную, индивидуальную модель данных.

Характерным примером реализации данного подхода является концепция Object-Relational Mapping (ORM), которая предоставляет объектный интерфейс для доступа к реляционной базе данных. ORM фреймворк может быть написан на каком-либо объектно-ориентированном языке (PHP, Python, Ruby) и представлять обертку над некой реляционной базой данных. Классы будут соответствовать таблицам в базе, а экземпляры этих классов – конкретным строкам таблицы. Общая схема взаимодействия объектов с таблицами с использованием ORM-фреймворка представлена на рисунке 5.

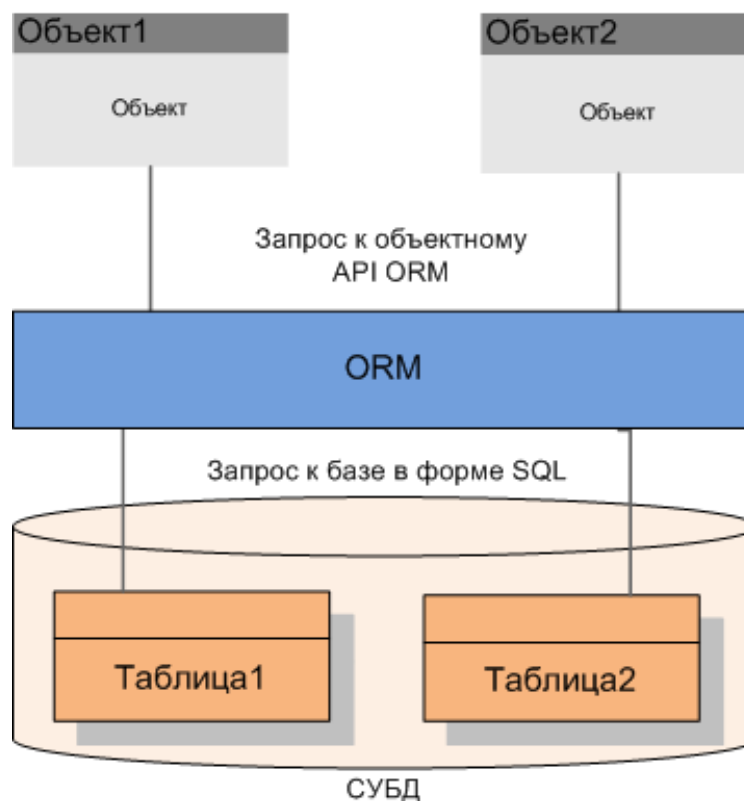


Рисунок 5 – Схема взаимодействия объектов ЯП с таблицами РСУБД

В данном случае, объекты ORM API будут являться объектным метаописанием таблиц базы данных. На основе метаописания, библиотека ORM, вне зависимости от языка программирования будет транслировать запросы к API со стороны пользовательского кода в SQL-код. Данный код, в свою очередь, будет интерпретирован СУБД, после чего необходимые данные будут изменены.

Рассмотренные выше подходы к проектированию информационных систем в той или иной степени базируются на метаописаниях предметных областей или отдельных функциональных компонентов системы. При этом, метаописание может задаваться как в явном виде (объекты ORM), так и в неявном, когда программист перекладывает предметную область на какую-либо технологию (например, язык программирования).

Таким образом, исходя из описанных свойств, рассмотренных подходов к проектированию информационных систем оперативного уровня на базе многоуровневой архитектуры, можно сделать вывод, что наиболее предпочтительным является именно подход, основанный на управлении моделями. Так как это позволит построить одну универсальную метамодель, описывающую широкий спектр предметных областей. Наиболее эффективно такой подход возможно будет применить в случае необходимости внедрения оперативных информационных систем.

Выводы

Анализ подходов показал, что метаданные в том или ином виде присутствуют в каждом из описанных видов систем. Но существует проблема интеграции этих систем, т.к. между ними отсутствует слой метаданных, обеспечивающий интерфейс, посредством которого системы смогли бы преобразовывать данные в универсальный формат и представлять средства для агрегированного анализа данных.

В главе были рассмотрены различные варианты архитектур построения систем оперативного уровня, а также подходы, реализующие и дополняющие представленные архитектуры.

Архитектуры, рассмотренные в пункте 1.3.1, являются базовыми. На их основе реализуются основные подходы для проектирования ИС. Также, в пункте 1.3.2 были рассмотрены подходы к проектированию информационных систем оперативного уровня на базе многоуровневой архитектуры. Исходя из проведённого анализа, был выбран подход, основанный на управлении моделями, т.к. это позволяет спроектировать одну универсальную метамодель, на базе которой можно было бы конфигурировать широкий спектр информационных систем оперативного уровня под любую предметную область с минимальными затратами.

Для построения такой универсальной модели данных и описывающей её метамодели, требуется разработать эффективные алгоритмы преобразования универсальной модели данных.

2 Структура метаданных и операции над метаданными в оперативных информационных системах

2.1 Формализованное представление реляционной базы данных оперативной информационной системы

Современные оперативные информационные системы, для эффективного обеспечения рабочего процесса, должны быть совместимы с различными типами баз данных вне зависимости от используемых моделей данных. Главной проблемой является разработка совместимых со всеми существующими БД, универсальных структур представления данных.

В работе [12], предлагается использовать именно реляционные модели, т.к. они логически просты и не несут в себе высокой структурной нагрузки. Как правило, данные в таких таблицах объединены между собой отношениями. А уже отношение состоит из атрибутов и данных (столбцов). При этом, атрибуты по одному соединяются в кортежи (строки) данных. Данная структура достаточно проста и понятна, хотя и имеет за собой достаточно прочную теоретическую основу.

Структура РСУБД задаётся реляционной схемой, содержащей в себе одну или несколько схем отношений. В таком случае, схема отношения будет задаваться типом отношения, именами и типами атрибутов, принадлежащих к конкретной предметной области. Каждый такой атрибут будет иметь собственное значение.

Данная структура имеет следующее формализованное описание:

$$A = \{a_1, a_2, \dots, a_e, \dots, a_E\}; \quad (1)$$

$$\forall a_e \in A \exists B_e = \{b_{e1}, b_{e2}, \dots, b_{eu_e}, \dots, b_{eU_e}\}; \quad (2)$$

$$\forall b_{eu_e} \in B_e \exists C_{eu_e} = \{c_{eu_e1}, c_{eu_e2}, \dots, c_{eu_e g_{eu_e}}, \dots, c_{eu_e G_{eu_e}}\}, \quad (3)$$

где A – множество имен отношений;

$e = \overline{1, E}$; $U_e = \overline{1, U_e}$; B_e – множество имен атрибутов отношения a_i ,
 C_{eu_e} – множество значений атрибута b_{eu_e} отношения a_e .

$$Z = \{z_1, z_2, \dots, z_1, \dots, z_L\}; \quad (4)$$

$$\forall z_l \in Z \quad \exists Y_l = \{y_{l1}, y_{l2}, \dots, y_{lm_l}, \dots, y_{lM_l}\}; \quad (5)$$

$$\forall z_l \in Z \quad \exists X_l = \{x_{l1}, x_{l2}, \dots, x_{ln_l}, \dots, x_{lN_l}\}, \quad (6)$$

где $l = \overline{1, L}$; $m_l = \overline{1, M}$; $n_l = \overline{1, N}$; и Y_l ; – множество реализаций записей-владельцев набора-связи типа z_l ;

X_l – множество реализаций записей набора-связи типа z_l . При этом:

$$\forall z_l \in Z \quad \exists \left\{ \langle y_{lm_l}, x_{ln_l} \rangle \mid y_{lm_l} \in Y_l, x_{ln_l} \in X_l \right\}. \quad (7)$$

Для формирования единой структуры БД, необходимо также определиться с размером самой крупной структуры. Так как оперативная ИС должна взаимодействовать с широким спектром реляционных баз данных, возникла необходимость в выделении внешнего уровня над множеством отношений $A - DB$, т. е.

$$DB = \{db_1, db_2, \dots, db_d, \dots, db_D\}; \quad (8)$$

где, db_d отдельная универсальная модель данных (УМД).

Таким образом, выражения (1)(3) видоизменяются:

$$\forall db_d \in DB \quad \exists A_d = \{a_{d1}, a_{d2}, \dots, a_{de_d}, \dots, a_{dE_d}\}; \quad (9)$$

$$\forall a_{de} \in A_d \quad \exists B_{de} = \{b_{de1}, b_{de2}, \dots, b_{deu}, \dots, b_{deU}\}; \quad (10)$$

$$\forall b_{deu} \in B_{de} \quad \exists C_{deu} = \{c_{deu1}, c_{deu2}, \dots, c_{deug}, \dots, c_{deuG}\}. \quad (11)$$

Таким образом, введение дополнительного уровня УМД, позволяет решить проблему интеграции системы с различными базами данных. А реляционная структура, позволяет повысить отзывчивость системы до необходимого уровня.

2.2 Структура метаданных оперативной информационной системы

В оперативных информационных системах, универсальность модели данных зависит, в первую очередь, от выбранной структуры метаданных. Правильно спроектированная структура, позволяет повысить удобство обработки данных, при этом, несильно усложняя систему. Обычно, понятия и правила выбранной предметной области, бывает достаточно тяжело отобразить в какую-либо заранее спроектированную модель метаданных. Для преодоления данной трудности, необходимо выделить некоторую совокупность метаклассов, которые содержали бы в себе сведения об описываемых объектах, таких как: простой тип, сложный (составной) тип, атрибут и связи. Архитектура метаданных в общем виде, представлена на рисунке 6.

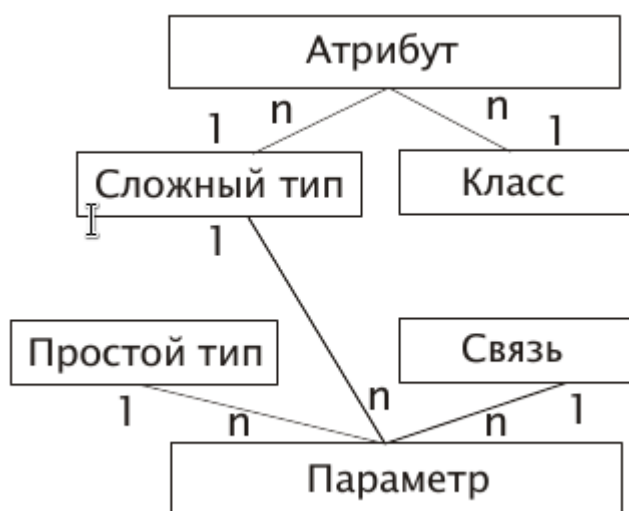


Рисунок 6 – Архитектура метаданных

Как можно видеть, на рисунке указана иерархия связей между метаклассами, где объект – это инкапсулированная сущность, обладающая некоторыми свойствами и методами.

Класс является определенным типом объектов, который задается при помощи описания, включающего атрибуты, методы и интерфейсы доступа к объектам класса.

Атрибут – это некоторое значение, сохраняемое объектом и задающее ту или иную его характеристику. При помощи атрибута также можно однозначно идентифицировать объект [13].

Параметры позволяют задать ограничения на значения атрибутов класса.

Простым типом называется элементарный, встроенный тип данных, принятый в используемом языке программирования или каком-либо ином инструментальном средстве проектирования программного обеспечения. С точки зрения проектирования БД, данный параметр будет являться хранимым типом данных в поле таблицы.

Сложный тип – класс, рассматриваемый как новый тип данных.

Связь – описывает связь между некоторыми классами.

Также, существует некоторое подмножество, называемое контуром – это подмножество, выделенное из всего множества атрибутов и описывающее атрибут с одной, определённой точки зрения [13].

Очевидным плюсом использования данной архитектуры является то, что её возможно воплотить, как в форме классов языка программирования, так и в форме таблиц реляционной базы данных. Если отображать данную архитектуру в РСУБД, то можно без особых ограничений, отобразить каждый описанный выше класс, как отдельную таблицу и задать необходимые связи в получившейся иерархии. Но при отображении на область объектно-ориентированных языков программирования, могут возникнуть некоторые сложности, которые будут описаны ниже. В общем случае, архитектура примет следующий вид, представленный на рисунке 7.

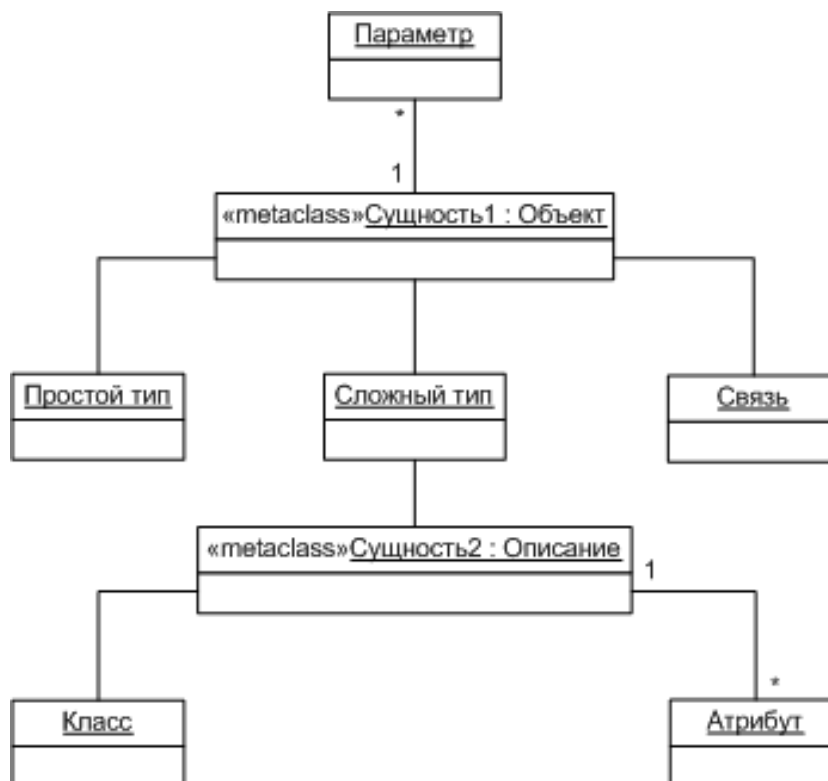


Рисунок 7 – Диаграмма классов в концепции ООП

Так как атрибут может относиться только к классу или сложному типу, появляется необходимость во введении дополнительных абстрактных метаклассов: «Сущность1» и «Сущность2». Единственная сложность, которая может возникнуть: это то, что многие языки программирования не поддерживают множественное наследование даже от абстрактных классов, но посредством внедрения в код композиций, появляется возможность обойти данное ограничение [13].

В описанной архитектуре метаданных, на первый взгляд, отсутствует связь между метаклассами «связь» и «класс». Так как в контексте РСУБД возможно однозначно определить только связь «многие к одному» или «один к одному», было принято решение исключить связь между этими метаклассами, т.к. метакласс «связь» описывает некоторую связь между двумя классами, что предполагает введение новых таблиц для разрешения данной коллизии. В данном случае: одной записи «связь», соответствуют две записи «класс» (т.к.

«класс» может быть связан сам с собой), а одной записи «класс», может быть поставлено в соответствие множество «связей» [14].

2.2.1 Описание компонентов структуры метаданных оперативной информационной системы

Метакласс может содержать информацию о других классах или таблицах. Ниже приведён список атрибутов метакласса, при этом, многие из атрибутов могут и вовсе не использоваться. Всё зависит от конкретной предметной области.

1. Наименования класса/Сложного типа. Представляет собой имя, под которым объект или таблица будет записан в БД. С помощью данного атрибута будет производиться доступ к объекту.

2. Текстовое описание класса/Сложного типа. Содержит в себе информацию описательного типа для дальнейшей интерпретации, например, в интерфейсе.

3. Наименование базового класса/Сложного типа. Понадобится, если архитектура построена с применением объектного подхода, а также хранит информацию о наследовании.

4. Поддержка журналирования данных. Данный атрибут позволяет включать или выключать журналирование отдельно взятых классов. С точки зрения оперативных систем, данный параметр может быть ключевым, т.к. такие системы могут проектироваться именно для этих целей.

5. Поддержка удаления класса. Описывает признак существования класса или таблицы, т.к. объект (даже если был удалён), может понадобиться в будущем.

6. Имя пользователя, который произвёл модификацию компонента.

7. Информационная связь между описаниями атрибутов класса / сложного типа – реализация данной связи может зависеть от выбранного подхода проектирования: реляционного или объектного.

8. Информационная связь, описывающая методы класса. Содержит в себе описания методов класса, объекта или сложного типа.

9. Информационная связь с описаниями состояний. Содержит в себе описание состояний таблиц или объектов класса, т.к. он является сущностью предметной области и несёт в себе некоторую семантическую нагрузку.

10. Модель хранения объектов сложного типа. Содержит в себе ссылку на таблицу/объект в базе данных, и ссылку на некоторый алгоритм преобразования данных.

Каждый метакласс, помимо описания, может содержать определённые атрибуты. Тип и семантическое наполнение которых может зависеть как от конкретной предметной области, так и от предпочтений программиста. Ниже представлено описание основных метаатрибутов, которые являются необходимым минимумом для организации классов метамодели [15].

1. Имя атрибута.

2. Текстовое описание атрибута.

3. Тип атрибута. Указывает: является ли атрибут коллекцией.

4. Модификатор доступа атрибута. Является ограничителем доступа к атрибуту, по аналогии с модификаторами доступа для полей класса в объектно-ориентированных языках программирования.

5. Флаг обязательности атрибута. Является аналогом флага NOT_NULL в РСУБД и используется в основном при построении системы с применением объектного подхода. Также указывает: обязателен ли атрибут или его можно не указывать при заполнении объекта.

6. Флаг уникальности атрибута. Является аналогом ограничения UNIQUE в РСУБД и используется в основном при построении системы с применением объектного подхода. Также указывает: можно ли сохранять объект со значением атрибута, который совпадает с уже ранее сохранённым атрибутом другого объекта.

7. Тип значения атрибута. Может обозначать, как имя встроенного типа (String, Integer и т.д.), так и имя какого-либо сложного типа.

8. Флаг вхождения в семантическое ядро класса. Указывает: формирует ли данный атрибут семантическое ядро класса и если флаг активен, то система не позволит удалить данный атрибут.

9. Видимость значения атрибута. Позволяет регулировать видимость атрибута при выполнении разнородных запросов. Например: нет необходимости в документе отображать идентификатор пользователя в системе, поэтому данный атрибут следует скрыть.

10. Флаг существования атрибута. Является некоторым компонентом темпоральности жизни объекта и указывает: существует ли атрибут на данный момент или был удален из определения класса.

11. Принадлежность атрибута к контурам. Указывает: к каким контурам относится атрибут (аналог облака тегов для объекта). Например: параметр «Производительность» может принадлежать к экономическому и технологическому контурам.

12. Флаг журналирования изменения значения атрибута. Является одним из ключевых параметров оперативной системы и регулирует факт записи изменения состояний объекта в системный журнал.

13. Ссылка на данные. Содержит в себе ссылку на какие-либо конкретные данные (на справочник или документ), которые не могут быть отображены в текущей модели хранения данных.

14. Флаг индексирования. Указывает: является ли атрибут индексом.

Для предотвращения ситуации, когда пользователь может задать заведомо бессмысленную связь между объектами классов, необходимо строго формализовать и описать в метаданных все вариации типов связей. Ниже, описаны типы связей, в которых с каждой стороны участвует по одному объекту класса. Примером, в данном случае будет являться факт владения человека недвижимостью (например, домом) [15].

1. Наименование класса-инициатора, объекты которого инициируют связь, например «Человек».

2. Наименование класса-приемника, объекты которого являются зависимым от инициатора связи, например «Дом».

3. Описание связи для класса-инициатора. При несимметричных связях может принимать форму «Начальник-подчиненный», а при симметричных (например: родственных) – «Брат-Брат». В данном случае – связь «Собственник».

4. Описание связи для класса-приемника. В данном случае принимает форму «Собственность».

5. Контур или контуры, в которые входит данная связь. В данном случае для класса «Человек», это может быть контур с примерным названием «Имущественный».

6. Контейнеризация связи. Указывает на то, входит ли один объект в состав другого (актуально для связей между классом и сложными типом). В данном примере, связь контейнерной не является, т.к дом может существовать отдельно от человека.

7. Видимость связи. Указывает на то, существует ли описываемый тип связи в настоящее время или был удален. В противном случае он резервируется для работы с ним системного программиста.

8. Ограничение на количество связей. Задаётся, если используется объектная модель (в реляционной, подобные ограничения задаются на уровне СУБД). Примером данного параметра, может служить количество «домов» в собственности «человека». И наоборот: что одним «домом» может владеть только один «человек».

9. Дополнительная уточняющая метасвязь. Примером такой метасвязи может являться стоимость приобретения дома.

Реализация предложенной архитектуры с использованием предложенных выше компонентов позволит организовать универсальную метамоделю для

описания практически любой предметной области. Причём реализовать это, как было сказано ранее, представляется возможным, как на основе объектной модели, так и на основе реляционной.

2.3 Построение универсальной модели данных оперативной информационной системы

На основе описанной в пункте 2.2 архитектуры, следует построить некоторую универсальную модель данных (УМД), которая бы позволяла хранить созданную метамодель и позволять с ней эффективно взаимодействовать [16].

Пусть mObj (metadata of objects) – таблица из двух столбцов: уникального идентификатора и названия класса. Данная таблица содержит в себе все имеющиеся классы. Также, опишем таблицу mAttr (metadata of attributes) из трех столбцов: уникальный идентификатор, название и внешняя ссылка на родительский класс в mObj. Данная таблица содержит все атрибуты всех классов. Введём таблицу dObj (data of objects), в которой хранятся все имеющиеся объекты классов, а сама таблица будет состоять из двух столбцов: уникальный идентификатор и внешняя ссылка на таблицу mObj, которая указывает на родительский класс. Последняя таблица dAttr (data of attributes), содержит в себе описание всей модели базы метаданных. Одному кортежу данной таблицы, ставится в соответствие один атрибут одного объекта. Состоит данная таблица из трёх столбцов: ссылка, указывающая на объект в dObj; ссылка, указывающая на атрибут объекта mAttr и, собственно, само значение.

Следует понимать, что представленная архитектура является базовой и в каждом конкретном случае может всячески дополняться и видоизменяться. Представленная модель логически непротиворечива и позволяет хранить данные о метамодели и взаимодействовать с ней. Ниже, на рисунке 8, представлена структура БД, описывающая взаимосвязи между таблицами данной метамодели.

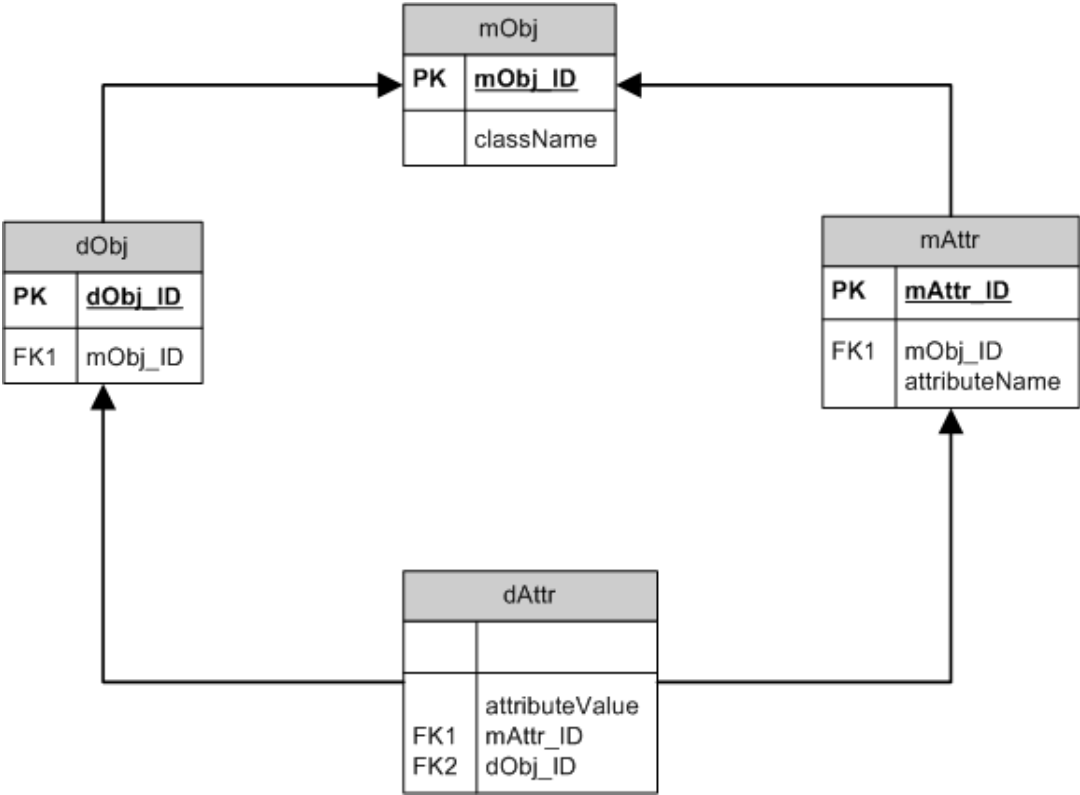


Рисунок 8 – Обобщённая базовая структура УМД

Для описания логической модели базы данных, распишем пример, приведённый в пункте 2.2, где рассматривается связь «Собственник» между «Человек» и «Дом». Данная связь устанавливается, если человек приобрёл в собственность дом. Связь имеет тип «один ко многим», т.к. в собственности у одного человека может находиться множество домов. Концептуальная модель данных, в данном случае будет иметь такие сущности, как: «Человек», «Дом», «Застройщик» и сложный тип «Адрес». Данные сущности обладают следующим перечнем свойств, описанным ниже в таблице 1.

Таблица 1 – Таблица сущностей и соответствующих свойств

	Сущность			
	Человек	Дом	Застройщик	Адрес
	фамилия	адрес	адрес	страна
	Имя	дата постройки	наименование организации	Почтовый индекс

Продолжение таблицы 1

Свойство	отчество	количество подъездов	реквизиты	область
	дата рождения	охрана	тип организации	город
	ИНН	парковка	номер лицензии	район
	номер паспорта	застройщик	год основания	улица
	пол			номер дома
	гражданст во	владелец (человек)	завершенные проекты	
	адрес			

Для более чёткого понимания, данная таблица сущностей и свойств отображена в контексте объектно-ориентированного языка Golang. В листинге 1, представлен код, описывающий взаимосвязи между сущностями.

Листинг 1 – Пример концептуальной модели, описанной с применением объектного подхода

```

type Address struct {
    addressId      int           // уникальный идентификатор
    state          string        // страна
    postIndex      string        // почтовый индекс
    region         string        // область
    city           string        // город
    district       string        // район
    street         string        // улица
    building       int           // номер дома
}

type Person struct {
    personId      int           // уникальный идентификатор
    firstName     string        // имя
    lastname      string        // фамилия
    patronymic    string        // отчество
    birthDate     time.Time    // дата рождения
    inn           string        // ИНН
    passport      string        // номер паспорта
    sex           string        // пол
    citizen       string        // гражданство
    address       Address       // адрес проживания
    house         []House      // связь с сущностью Дом в форме списка
}

```

```

type Builder struct {
    builderId      int           // уникальный идентификатор
    name           string        // наименование организации
    requisites     string        // реквизиты
    typeOrg        string        // тип организации
    license        string        // номер лицензии
    foundation     time.Time     // год основания
    address        Address       // адрес организации
    completedProjects []House      // связь с сущностью Дом в форме списка
}

type House struct {
    houseId      int           // уникальный идентификатор
    address      Address       // адрес
    dateOfBuilding time.Time     // дата постройки
    entrancesNumber int         // количество подъездов
    security     boolean      // наличие охраны
    parking      Boolean      // наличие парковки
    managementCompany Builder     // управляющая компания
    owner        Person      // владелец
}

```

В листинге, поля структур соответствуют соответствующим полям таблиц БД. Также, в коде указаны отдельные поля, которые объявлены как списки с типами других структур. Так было решено показать связь «один ко многим». Также, некоторые классы сущностей, в зависимости от рассмотрения, играют роль самостоятельной сущности или сложного типа данных [16].

Ниже, в листинге 2, представлен вариант реляционной интерпретации структуры.

Листинг 2 – Пример концептуальной модели, описанной с применением языка SQL

```

CREATE TYPE Address AS OBJECT (
    Id          CHAR(10),
    State       CHAR(20),
    postIndex   CHAR(20),
    city        CHAR(20),
    district    CHAR(20),
    street      CHAR(20),
    building    INT,
    PRIMARY KEY(id)
);

CREATE TABLE Person
(
    Id          INT,
    firstName   CHAR(15),
    lastName    CHAR(15),

```

```

        patronymic      CHAR(15),
        birthDate       DATE,
        INN             CHAR(12),
        Passport        CHAR(15),
        Sex             CHAR(5),
        Citizen         CHAR (10),
        Address         ADDRESS,
        PRIMARY KEY(id),
        FOREIGN KEY(house) REFERENCES House
    );

CREATE TABLE Builder
(
    id                INT,
    name             CHAR(20),
    requisites       CHAR(50),
    typeOrg          CHAR(15),
    license          CHAR(15),
    address          ADDRESS,
    foundation        DATE,
    PRIMARY KEY(id),
    FOREIGN KEY (completedProjects) REFERENCES House
);

CREATE TABLE House
(
    Id                INT,
    Address           ADDRESS,
    dateOfBuilding    DATE,
    entrancesNumber   INT,
    security          BOOLEAN,
    parking           BOOLEAN,
    managementCompany Builder,
    owner            Person
    PRIMARY KEY(id),
);

```

Универсальная модель данных, отображающая описанную выше архитектуру, имеет более сложную структуру, нежели у концептуальной модели. Так как единого, общепринятого стандарта УМД не существует, то не существует и регламента по подразделению моделей сущностей по компонентам метамодели. Следовательно, архитектор системы волен самостоятельно подразделять атрибуты по таблицам [16]. В листинге 3 представлен код для генерации сервисных таблиц базы данных, удовлетворяющий описанной выше структуре метамодели.

Листинг 3 – Пример структуры базы метаданных, описанной с применением языка SQL

```

/* Список типов */
CREATE TABLE TYPE_LIST
(
    ID_NUM            INTEGER      NOT NULL,
    NAME_TYPE         VARCHAR(40),
    PRIMARY KEY (ID_NUM);
);

/* Список свойств */
CREATE TABLE PROP_LIST /
(
    ID_NUM            INTEGER      NOT NULL,
    ID_TYPE            INTEGER      NOT NULL,
    TYPE_PROP         SMALLINT,
    NAME_PROP         VARCHAR(40),
    PRIMARY KEY (ID_NUM),
    FOREIGN KEY (ID_TYPE) REFERENCES TYPE_LIST(ID_NUM);
);

/* Данные */
CREATE TABLE DATA_LIST
(
    ID_NUM            INTEGER      NOT NULL,
    ID_TYPE            INTEGER      NOT NULL,
    PRIMARY KEY (ID_NUM);
    FOREIGN KEY (ID_TYPE) REFERENCES TYPE_LIST(ID_NUM);
);

/* Целые числа */
CREATE TABLE VAL_INT
(
    ID_NUM            INTEGER      NOT NULL,
    ID_PROP            INTEGER      NOT NULL,
    VAL               INTEGER,
    PRIMARY KEY (ID_NUM, ID_PROP);
    FOREIGN KEY (ID_NUM) REFERENCES DATA_LIST(ID_NUM);
    FOREIGN KEY (ID_PROP) REFERENCES PROP_LIST(ID_NUM);
);

/* Остальные таблицы низкоуровневых форматов данных */
CREATE TABLE VAL_STR /* Строки*/
(
    ID_NUM            INTEGER      NOT NULL,
    ID_PROP            INTEGER      NOT NULL,
    VAL               VARCHAR(128),
    PRIMARY KEY (ID_NUM, ID_PROP);
    FOREIGN KEY (ID_NUM) REFERENCES DATA_LIST(ID_NUM);
    FOREIGN KEY (ID_PROP) REFERENCES PROP_LIST(ID_NUM);
);

```

Таблицы TYPE_LIST, PROP_LIST, DATA_LIST являются аналогами таблиц mObj, mAttr, dObj соответственно. Новое поле TYPE_PROP из таблицы

PROP_LIST хранит тип атрибута. Таким образом, таблица dAttr подразделяется на несколько таблиц – по количеству низкоуровневых типов данных: VAL_INT, VAL_STR и т.д. Для каждой из этих таблиц тип поля хранящего значение атрибута, уже становится определенным и может быть назначен в CREATE TABLE.

Если запись является типом из другой таблицы, то необходимо создать дополнительную таблицу, содержащую ссылки на другие записи. Пример такой таблицы представлен в листинге 4.

Листинг 4 – Пример таблицы, содержащей ссылки на записи с типами из других таблиц

```
CREATE TABLE VAL_POINT
(
    ID_NUM          INTEGER      NOT NULL,
    ID_PROP         INTEGER      NOT NULL,
    VAL             INTEGER      NOT NULL,
    PRIMARY KEY (ID_NUM, ID_PROP);
    FOREIGN KEY (ID_NUM) REFERENCES DATA_LIST (ID_NUM);
    FOREIGN KEY (ID_PROP) REFERENCES PROP_LIST (ID_NUM);
    FOREIGN KEY (VAL) REFERENCES DATA_LIST (ID_NUM);
);
```

Поля таблицы содержат два внешних ключа: на таблицу объектов DATA_LIST. ID_NUM содержит ссылку на объект атрибута. VAL – на объект, являющийся значением атрибута.

Произведя данные манипуляции, структура становится более специфицированной, а связи между таблицами базы данных – более явными. Появляется возможность интеграции данной УМД с другими компонентами базы данных [17].

Таким образом, обобщённая базовая структура УМД, проиллюстрированная на рисунке 8, трансформируется в специфицированную базовую структуру УМД. Данная структура изображена на рисунке 9.

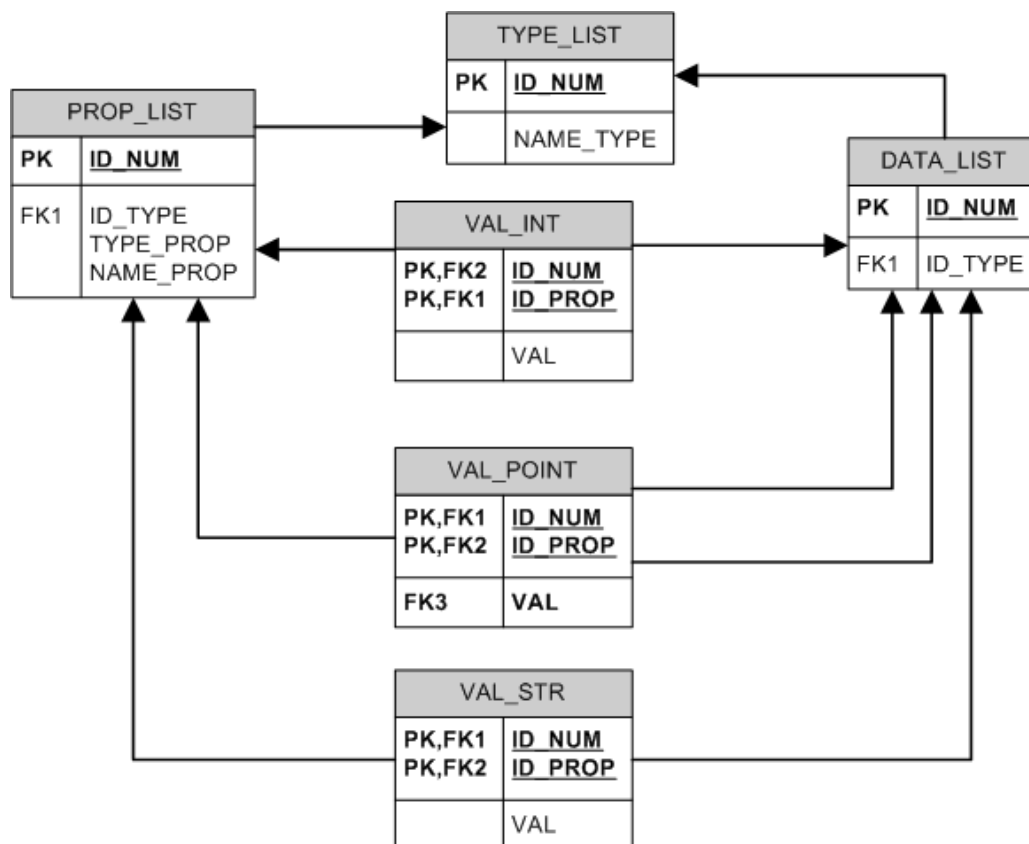


Рисунок 9 – Обобщённая базовая структура УМД

Предложенная модель позволяет повысить отзывчивость системы при клиент-серверных взаимодействиях между интерфейсом и бизнес-логикой, а также сократить количество обращений к БД, что является ключевой характеристикой при проектировании оперативной информационной системы.

2.4 Операции над метаданными оперативной информационной системы

Модель предметной области находится в постоянном развитии. Какие-то компоненты добавляются, а какие-то удаляются. Соответственно, меняется и логическая модель базы данных. Данные модификации возможно производить без последствий до тех пор, пока не приходится нарушить согласованность между существующими компонентами схемы отношений. Если подобное всё же произошло, то база данных перестанет выполнять свои функции, ведь в ней

будет нарушена целостность структуры и семантическая связь между классами или таблицами.

Как было описано ранее, существует множество компонентов метамодели и информационная система, помимо организованного хранения метаданных, должна позволять определённым образом ими манипулировать. Следует уточнить, что производимые изменения в данных, производятся над компонентами, описанными в пунктах 2.2 и подпункте 2.2.1 , а именно над:

- сложными типами;
- классами (таблицами);
- простыми типами;
- атрибутами;
- метасвязями между компонентами.

Все операции над данными компонентами можно подразделить на следующие классы:

- удаление (мягкое или жесткое);
- создание;
- редактирование;
- восстановление.

Для полноценного понимания, что представляют из себя данные операции – ниже представлено более подробное описание.

В общем случае, операции создания новых компонентов являются достаточно простыми и не требуют дополнительного описания. Объект последовательно записывается в соответствующие таблицы mObj, dObj, mAttr, dAttr и становится доступен для дальнейшего использования. Операции редактирования, также являются простыми и подразумевают каскадные изменения во всех дочерних объектах (взаимосвязанных таблицах).

Операцию удаления следует рассмотреть подробнее, т.к. она неоднородна и различается в зависимости от удаляемого компонента. Удаление компонентов метамодели может производиться двумя способами: «мягким» или «жестким».

В первом случае, следует говорить об изменении атрибута видимости объекта Active, класса mAttr на значение 0 (либо false, в зависимости от используемого типа данных). Если пользователь выбирает мягкое удаление, то с этого момента для пользователя атрибут перестает существовать, а связи становятся неактивными. Данный атрибут, хоть и не пропадает для пользователя, но остается полноправным участником схемы данных. Вся информация, хранящаяся в данном атрибуте, остается в БД в неизменном виде, но пользователь не сможет с ней взаимодействовать без восстановления удаленного объекта.

Для проведения жесткого удаления, система должна выполнить некоторую последовательность действий:

- 1) удалить объект класса mAttr (в объектной реализации архитектуры) или удалить запись в таблице существующих объектов (для реляционной схемы);
- 2) на уровне бизнес-логики, произвести удаление атрибута из метаописания класса;
- 3) если атрибут сложного типа, то каскадно удалить из таблиц все ссылки на данные (или все его экземпляры, в случае объектной реализации);
- 4) если атрибут является индексом, то удалить из таблицы индексов (снять флаг индексирования с объекта).

При произведённом мягком удалении и дальнейшей записи нового атрибута с тем же именем, но измененной сигнатурой, может произойти коллизия имён: когда имя удаленного и добавленного атрибута совпадают. У данной ситуации есть выход: когда система, при добавлении нового атрибута, будет предлагать его жесткое удаление из модели данных с последующей записью нового атрибута на его место. Таким же образом, можно решить проблему коллизии имён и для мягкого удаления метаданных.

В конечном итоге, производить манипуляции с данными нужно только при жестком удалении компонентов. В противном случае, можно просто

деактивировать его связи и изменить область видимости объекта для пользователя.

При жестком удалении класса, необходимо произвести удаление следующих компонентов в указанном ниже порядке:

- 1) объект из класса `mObj` (запись из одноименной таблицы в случае с реляционной реализацией);
- 2) метаописания атрибутов в классе (таблице) `mAttr`;
- 3) метаописания связей в классе (таблице) `mRelation`;
- 4) все объекты класса (или записи в таблице объектов) из УМД;
- 5) все связи в классе `dRelation`, в которых участвуют объекты удаленного класса.

При удалении простого или сложного типа, могут возникнуть осложнения, в случае, если у класса присутствуют атрибуты удаляемого типа. Существует два варианта решения проблемы:

- уведомить пользователя о наличии атрибутов и отменить удаление, предложив пользователю предварительно удалить атрибуты этого типа;
- уведомить о наличии атрибутов и в случае подтверждения произвести удаление.

При удалении всех атрибутов класса, необходимо произвести удаления в следующем порядке:

- 1) объекта класса (или записи в таблице объектов) `mTypeSimple` или `mTypeComplex`;
- 2) объекты в `mParam` (или кортеж записей в таблице);
- 3) дополнительный объект в классе `mAttr`, если тип сложный.

При мягком удалении и последующем восстановлении атрибута, необходимо сначала восстановить тип и уже следом атрибут.

Операции восстановления, также не требуют дополнительного описания, т.к. при мягком удалении компонента следует лишь изменить его область видимости для пользователя, тем самым восстановить удаленные связи. При

восстановлении жестко удаленных компонентов, следует использовать механизм отката транзакции в БД, либо заново создать все необходимые компоненты и связи между ними.

Выводы

Разработанная метамодель оперативной информационной системы позволяет создать независимую от предметной области модель данных. Такой подход позволяет создать систему, которая сможет дополняться различными объектами предметной области без значительной модификации базы данных. Появляется возможность динамической генерации интерфейсов на основе метамодели.

Благодаря изложенным операциям над метаданными, появляется возможность переустройства модели данных системы без накладных расходов. Возможность «мягкого» удаления данных, позволяет снизить, например: задержки на каскадное удаление записей из таблиц БД. Также, появляется возможность восстановления данных путём изменения их области видимости для пользователя. Это может послужить дополнительным упрощением при отображении данных в интерфейс пользователя.

3 Программные средства для построения приложений в составе оперативно-аналитической информационной системы

3.1 Назначение, архитектура и принципы функционирования оперативно-аналитической информационной системы

Оперативно-аналитическая система, представляет собой аппаратно-программный комплекс разнородных компонентов, расположенных на разных уровнях иерархии в соответствии с выбранным для проектирования типом архитектуры.

В главе 1 была рассмотрена классификация систем по уровням управления. В диаграмме, представленной на рисунке 2, также была продемонстрирована зависимость: чем более стратегические задачи стоят перед персоналом, тем меньшим объёмом информации они должны манипулировать. В основном, это сводные, обработанные данные, которые могут быть менее структурированы, в отличие от систем тактического уровня. Так как данных немного, они сводные и неструктурированные, то системы, проектирующиеся для решения задач стратегического уровня, должны позволять получать знания из таких данных, путём их преобразования через классификацию, агрегацию и декомпозицию. Системы на таком уровне управления, позволяют, на основе имеющихся данных анализировать текущую обстановку в какой-либо предметной области, а на основе проведённого анализа прогнозировать исход каких-либо принятых решений.

Таким системам не требуется минимальное время отклика, т.к. при их проектировании делается упор на качество проводимого анализа, а не на скорость обработки. Они обладают сложной бизнес-логикой, применение которой исключает оперативное вмешательство в структуру данных и может не лучшим образом повлиять на качество итогового анализа системы.

При движении вниз по уровням управления, возрастает количество и степень структурированности данных. Максимальное количество которых,

достигается как раз на уровне операционного персонала. Данным специалистам требуется такая система, которая автоматизировала бы большую часть рутинной работы и обладала максимальной отзывчивостью. И всё это с сохранением возможности манипуляций огромными объёмами высокоструктурированных данных. В данном случае, следует говорить о применении оперативных систем. Они способны быстро осуществлять доступ сразу к большому количеству малых по объёму объектов и производить над ними необходимые преобразования. Благодаря такому подходу повышается скорость обработки небольших порций данных, что является существенным показателем для систем на тактическом уровне организации производства.

Различия между данными типами систем состоят не только в их предназначении, но и в используемом подходе хранения данных. Для проектирования аналитических ИС, обычно используется концепция OLAP: она позволяет повысить эффективность проводимого системой анализа, а также обеспечивает качественное хранение данных в СУБД. Для проектирования оперативных информационных систем, обычно, применяется концепция OLTP: она позволяет сократить время отклика системы на манипуляцию небольшими порциями структурированных данных, но, в отличие от OLAP не предлагает свою структуру организации данных.

Также, существуют и различные вариации гибридов данных систем. В таком случае, обычно говорят об оперативно-аналитических системах. В них сочетаются преимущества обоих типов ИС, но многократно возрастает сложность и, как следствие – стоимость проектирования подобного рода ИС.

Целью данной работы, как говорилось ранее – создать концепцию универсальной системы, которая обладала бы всеми преимуществами аналитических и оперативных ИС и при этом позволяла бы достаточно легко производить переконфигурацию системы под нужды предприятия, с минимальными трудозатратами. Добиться такого результата помогут, рассмотренные в разделе 2 метамодели и универсальные модели данных.

Архитектура предлагаемой системы базируется на основе микросервисов. Данный подход зарекомендовал себя с точки зрения скорости, отказоустойчивости и масштабируемости. Предлагаемая структура развертывания системы изображена на рисунке 10.

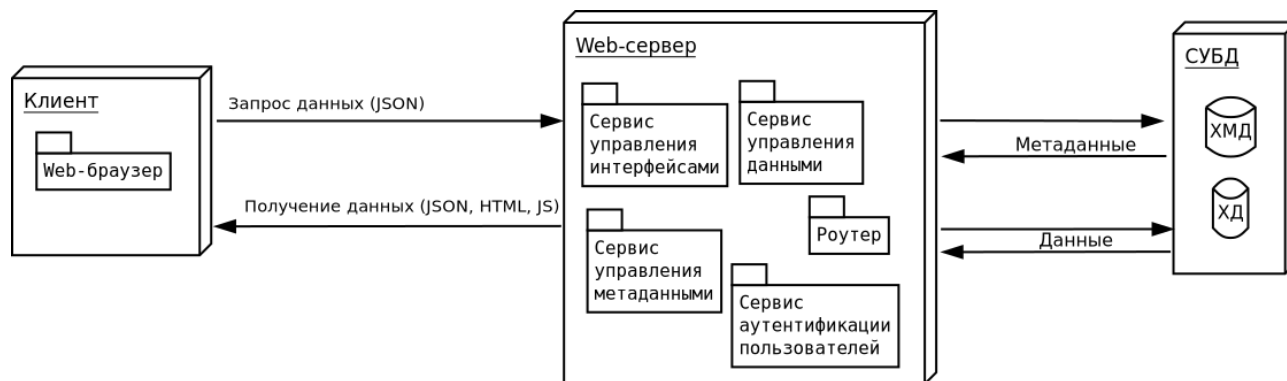


Рисунок 10 – Диаграмма развертывания для системы оперативного уровня

При этом, неважно, на одном web-сервере располагаются микросервисы, или это более распределённая архитектура, имеющая в своём составе множество объединенных веб-серверов.

1. Сервис управления интерфейсами. Отвечает за сопоставление метамодели объектов предметной области с реальными данными из УМД и динамически генерирует пользовательский интерфейс.

2. Сервис управления метаданными. Отвечает за извлечение и модификацию метамодели: проведение операций над метаданными, изложенными в подразделе 2.4.

3. Сервис управления данными. Производит выборку и модификацию данных в БД на основе метамодели.

4. Сервис аутентификации пользователей. Отвечает за выдачу прав пользователям, зарегистрированным в системе. Может базироваться как на метамодели, так и на ролях базы данных.

5. Роутер. Адресует запросы пользователя по компонентам системы.

Для взаимодействия компонентов системы между собой, будет использоваться протокол REST архитектурный стиль взаимодействия

компонентов распределённого приложения в сети [18]. Данный протокол позволяет согласовать набор ограничений, учитываемых при проектировании системы.

3.2 Программная реализация приложений концептуальной оперативно-аналитической информационной системы

Программная реализация оперативной ИС не отличается от реализации web-приложений на основе микросервисной архитектуры. Так как главной задачей данной научно-исследовательской работы является представить концептуальную модель оперативной ИС построенной с применением метамодели и на базе универсальной модели данных, то далее будет продемонстрирована пошаговая работа с данными в УМД внутри хранилища данных оперативной ИС.

Логика приложения подразделяется на компоненты, представленные на рисунке 11.

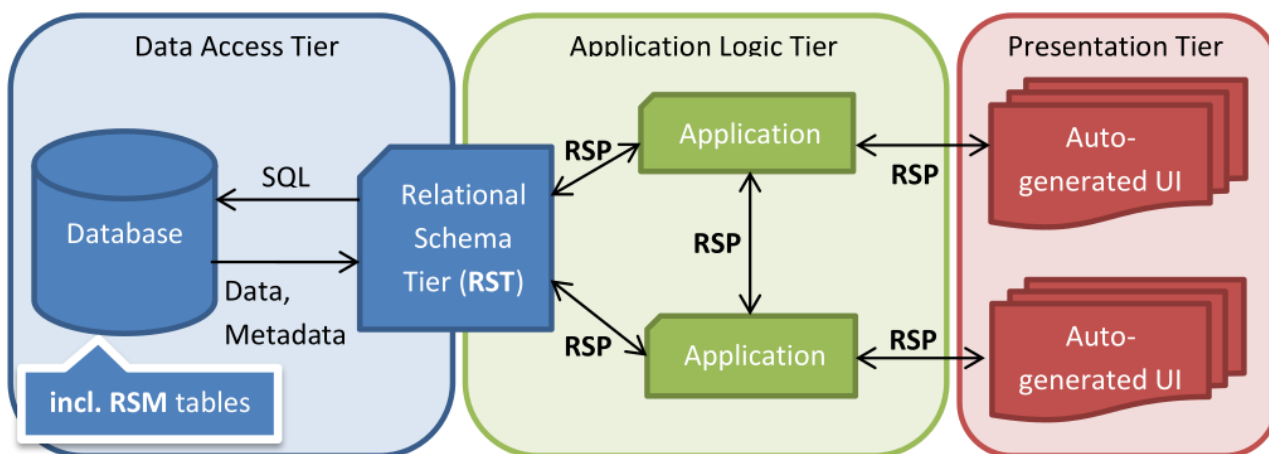


Рисунок 11 – Архитектура концептуального приложения

На рисунке выше, изображена архитектура web-приложения, подразделённая на 3 уровня:

- уровень доступа к данным (сервер СУБД);
- уровень логики приложения ;
- слой доступа к БД (ORM компонент);

- уровень представления (web-интерфейс).

Компоненты, распределённые по данным уровням, обмениваются через сетевой протокол HTTP, используя архитектурный стиль взаимодействия REST. Внутри REST, реализован собственный протокол RSP для обмена реляционными данными и метаданными. Целью этого протокола является замена зависимых от схемы операторов SQL на независимые от схемы удаленные вызовы процедур. Он определяет удаленные операции (процедуры) для исследования реляционных схем, обмена данными и метаданными и эффективного вычисления агрегатных функций, а также новую схему сериализации для общих реляционных данных и метаданных [19].

На уровне доступа к БД реализованы таблицы RSM (Relational Scheme Model). Они позволяют хранить метаданные (правила интерпретации, ссылки на типы данных и т.д.), как для генерации интерфейса, так и для обратного процесса – модификации существующей модели данных. Данные таблицы можно сконфигурировать на основе предложенной универсальной модели данных.

Для иллюстрации распределения данных внутри УМД (внутри слоя RSM), был взят за основу пример «Человек-Собственность». Из-за введённого разбиения обобщенной архитектуры, состав таблиц БД в УМД видоизменился и ниже будут представлены нормализованные таблицы БД, которые соответствуют mObj, mAttr, dObj и dAddr соответственно. Изменённые элементы для таблицы TYPE_LIST, представлены ниже, в таблице 2.

Таблица 2 – Логическая модель для таблицы TYPE_LIST

ID_NUM	NAME_TYPE
1	Address
2	Person
3	Builder
4	House

Изменённые элементы для таблицы PROP_LIST, представлены ниже, в таблице 3.

Таблица 3 – Логическая модель для таблицы PROP_LIST

ID_NUM	ID_TYPE	TYPE_PROP	NAME_PROP
1	1	Int	addressId
2	1	String	state
3	1	String	postIndex
4	1	String	region
5	1	String	city
6	1	String	District
7	1	String	Street
8	1	Int	building
9	2	Int	personId
10	2	String	firstName
11	2	String	lastname
12	2	String	patronymic
13	2	Date	birthDate
14	2	String	inn
15	2	String	passport
16	2	String	sex
17	2	String	citizen
18	2	Address (ref)	address
19	2	House (ref)	house
20	3	Int	builderId
21	3	String	name
22	3	String	requisites
23	3	String	typeOrg

Продолжение таблицы 3

24	3	String	license
25	3	Date	foundation
26	3	Address (ref)	address
27	3	House (ref)	completedProjects
28	4	Int	houseId
29	4	Address	address
30	4	Date	dateOfBuilding
31	4	Int	entrancesNumber
32	4	Boolean	security
33	4	Boolean	parking
34	4	Builder (ref)	managementCompany
35	4	Person (ref)	owner

Изменённые элементы для таблицы DATA_LIST, представлены ниже, в таблице 4.

Таблица 4 – Логическая модель для таблицы DATA_LIST

ID_NUM	ID_TYPE
1	1
2	1
3	1
4	1
5	1
6	1
7	2
8	2
9	3

Продолжение таблицы 4

10	3
11	4
12	4

Изменённые элементы для таблицы VAL_STR, представлены ниже, в таблице 5.

Таблица 5 – Логическая модель для таблицы VAL_STR

ID_NUM	ID_PROP	VAL
2	10	Иван
3	12	Иванович
4	11	Иванов
5	14	277155279
6	15	1243 765890
7	16	Мужской
10	17	РФ
11	2	Россия
12	3	440000
14	4	Пензенская область
15	6	Первомайский
16	7	Ивановская
17	21	Термомом
21	24	2736286197647026917
22	23	ООО
23	22	1423 3559 1763 0198
24	5	Пенза

Изменённые элементы для таблицы VAL_INT, представлены ниже, в таблице 6.

Таблица 6 – Логическая модель для таблицы VAL_INT

ID_NUM	ID_PROP	VAL
2	1	1
3	8	15
4	9	1
5	20	1
6	28	1
7	31	5

Изменённые элементы для таблицы VAL_DATE, представлены ниже, в таблице 7.

Таблица 7 – Логическая модель для таблицы VAL_DATE

ID_NUM	ID_PROP	VAL
2	13	01.01.1970
3	25	01.05.2000
4	30	10.02.2020

Изменённые элементы для таблицы VAL_POINT, представлены ниже, в таблице 8.

Таблица 8 – Логическая модель для таблицы VAL_POINT

ID_NUM	ID_PROP	VAL
2	18	2
3	19	2

Продолжение таблицы 8

4	26	3
5	27	3
6	29	4

Описанные выше таблицы содержат нормализованные данные, для хранения которых, в обычной БД пришлось бы выделять отдельные сущности для каждого объекта предметной области. В данном случае, объекты предметной области хранятся в таблице TYPE_LIST и при необходимости, предметную область можно безболезненно расширить или сократить, при этом, не изменяя структуру УМД.

Для обмена данными с пользователем, система, посредством REST-запросов и через протокол RSP может осуществлять передачу сериализованных данных в определённом формате (например, в формате XML). Чтобы определить схему сериализации, необходимо задать определённые объекты на уровне бизнес-логики, в соответствии с диаграммой, представленной на рисунке 12.

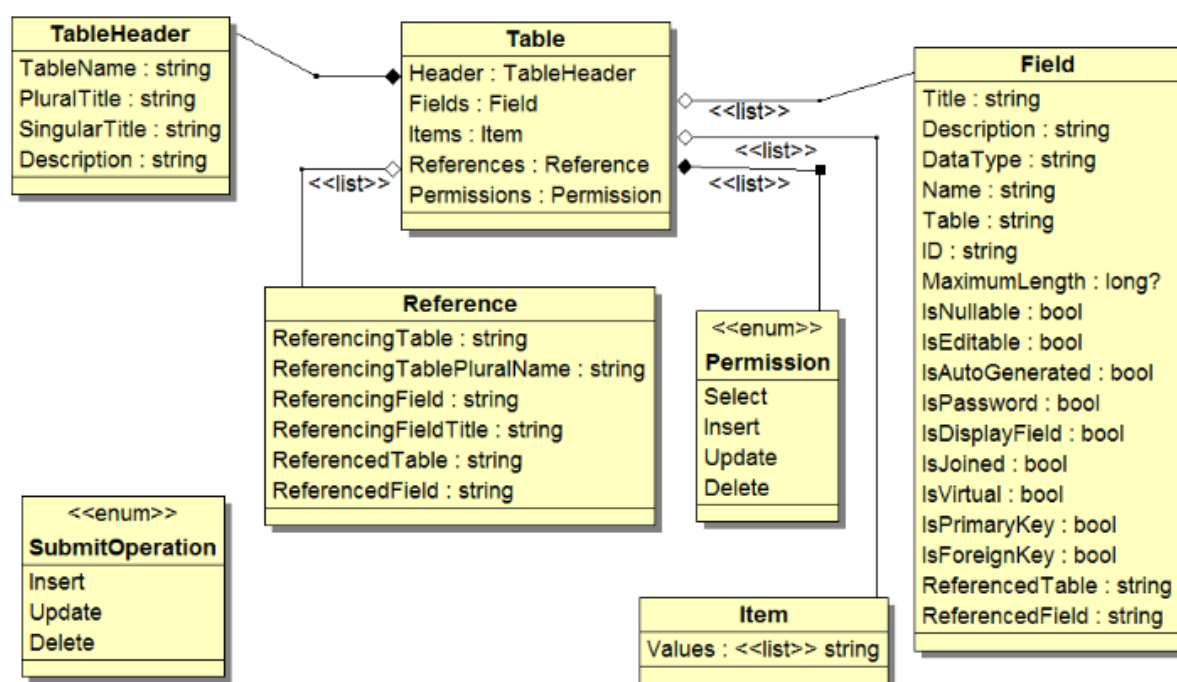


Рисунок 12 – Схема сериализации протокола реляционной схемы (RSP)

Протокол RSP, определяет операции для управления РСУБД, которые описаны в таблице 9.

Таблица 9 – Перечень реализуемых операций в протоколе RSP

Наименование метода	Выполняемое действие
ReadTableHeaders	Исследование схемы
ReadTable	Извлечение данных из метаданных
Submit	Манипуляция данными
ReadScalar	Чтение скалярного значения одного значения данных
ReadCount	Подсчёт числа элементов
ReadSum	Вычисление суммы

Таким образом, применение предложенных архитектурных решений, позволяет нормализовать данные внутри хранилища данных и тем самым повысить скорость обработки данных, что является критическим фактором для оперативных информационных систем.

Выводы

Описаны основные этапы разработки оперативной информационной системы, а также различные архитектурные решения, для оптимизации процесса разработки оперативной ИС. Также, описанный подход позволяет системам, независимо от разработчика генерировать различные вспомогательные компоненты системы (например, графический интерфейс). Для реализации данного функционала был предложен протокол, который позволяет вести обмен данными между компонентами системы. А построенная на основе УМД, схема сериализации данного протокола позволяет многократно расширить область применения данного подхода (например, в обратном

направлении изменять модифицировать сущности предметной области без изменения в УМД).

Заключение

В работе проведен анализ архитектур оперативных информационных систем. В ходе анализа установлено, что при создании такого рода систем возникает ряд трудностей. Были рассмотрены стандартные подходы проектирования оперативных ИС и выявлены их ключевые недостатки. К ним относятся: высокая трудоемкость создания хранилищ метаданных; отсутствие стандартизации построения универсальных моделей данных; трудности с обеспечением целостности информации в организации, использующей в составе своей системы метамодели; необходимость модернизации хранилищ при изменениях в бизнес процессах.

Для решения поставленной задачи, было решено разработать универсальную модель данных, которая бы позволила без лишних трудозатрат производить модификацию модели предметной области в информационной системе. Для описания УМД, были приведены логические модели и схемы, а также операции для модификации метамодели. Описаны ключевые сущности, на которых базируется УМД.

В ходе работы были представлены:

- универсальная модель данных, на основе которой можно строить различные компоненты системы, в т.ч. и хранилища данных;
- алгоритм модификации метамодели;
- концептуальная информационная система, демонстрирующая преимущества предложенного подхода;

Данная работа является актуальной, так как возрастает необходимость в системах, способных с высокой скоростью исполнять небольшие транзакции. Такие системы, зачастую, внедряются для обеспечения отказоустойчивости целых программных комплексов. Результаты приведенных исследований могут применяться для разработки оперативных информационных систем или даже целого комплекса оперативно-аналитических систем, тем самым, позволяя

объединить преимущества обоих типов систем без привязки какой-либо предметной области.

Список использованных источников

1. Метаданные, их свойства, функции, классификация и средства представления / М. Когаловский. – Москва. : Институт проблем рынка РАН, 2012. – 14 с.
2. Metadata for the Web / C. Lagose. – New York. : Cornell University, 2005 – 431 p.
3. Метаданные (Metadata) - Базы данных: основные понятия // Базы данных Access [Электронный ресурс]. – URL: http://access.avorut.ru/publ/bazy_dannykh_osnovnye_ponjatija/m/metadannye_metad ata/41-1-0-134 (дата обращения 27.11.2019).
4. Introduction to metadata. Online Edition, Version 3.0 / T. Gill, A. G. Gilliland, M. Whalen, M.S. Woodley. – Los Angeles. : Getty Research Institute, 2008 – 96 p.
5. Место метаданных на различных этапах обработки данных // Studfiles [Электронный ресурс]. – URL: <https://studfiles.net/preview/4641542/page:2/> (дата обращения 15.01.2020).
6. Информационные технологии в экономике / М.В. Головицына – Москва. : Национальный открытый университет ИНТУИТ, 2016. – 589 с.
7. Poole, John D., Model-Driven Architecture: Vision, Standards And Emerging Technologies/ John D. Poole. ECOOP, 2001. 15 с.
8. Stahl T., Völter M., Efftinge S. Modellgetriebene Softwareentwicklung. Techniken, Engineering, Management/. Heidelberg, Dpunkt-Verlag, 2007.
9. A database-centric approach to J2EE application development // WayBack Machine [Электронный ресурс]. – URL: https://web.archive.org/web/20060525094651/http://www.oracle.com/technology/pub /articles/odtug_award.pdf (дата обращения 07.02.2020).

10. Database-Centric Grid and Cluster Computing // Base One International Corp. [Электронный ресурс]. – URL: <http://www.boic.com/dbgrid.htm> (дата обращения 07.02.2020).
11. The Data-Centric Revolution // The Data Administration Newsletter [Электронный ресурс]. – URL: <http://www.boic.com/dbgrid.htm> (дата обращения 10.02.2020).
12. Универсальные объектно-ориентированные базы данных на реляционной платформе / И.А. Микляев. – Архангельск. ИД САФУ, 2014. – 226 с.
13. СУБД Caché: работа с объектами / И.И. Труб. – Москва. Диалог-МИФИ, 2006. – 471 с.
14. Программирование на SQL для профессионалов / Джо Селко. – Москва. Лори, 2009. – 442 с.
15. СУБД Caché: объектно-ориентированная разработка приложений / В. Кирстен, М. Ирингер, Б. Рериг, П. Шульте. – Санкт-Петербург. Питер, 2001. – 384 с.
16. Проектирование объектно-ориентированных баз данных / Д. Харрингтон. – Москва. ДМК-Пресс, 2001. – 272 с.
17. База данных – хранилище объектов / А. Тенцер. – Москва. Компьютер-Пресс. 2001. – 150 с.
18. Технология Web-сервисов платформы Java / Т.С. Машнин. – Санкт-Петербург. БХВ-Петербург, 2012. – 560 с.
19. A Metadata-Driven Approach to Relational Database Management / Vojtěch Přehnal. – Brno. Masaryk University, 2012 – 23 с.

Приложение А
ГЛОССАРИЙ
(справочное)

Программное обеспечение (ПО) – совокупность всех программ и соответствующей документации, обеспечивающая использование ЭВМ в интересах каждого ее пользователя.

Автоматизированная информационная система (АИС) – это комплекс программных, технических, информационных, организационно-технологических средств, предназначенный для решения задач справочно-информационного обслуживания или информационного обеспечения пользователей.

Язык программирования (ЯП) – формальная знаковая система, предназначенная для записи компьютерных программ. Язык программирования определяет набор лексических, синтаксических и семантических правил, определяющих внешний вид программы и действия, которые выполнит исполнитель (обычно - ЭВМ) под её управлением.

База данных (БД) – организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей.

Система управления базами данных (СУБД) – совокупность языковых и программных средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных одним или несколькими пользователями.

Объектно-ориентированное программирование (ООП) – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.