Министерство образования и науки Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Институт кибернетики
Направление подготовки 09.04.01 «Информатика и вычислительная техника»
Кафедра оптимизации систем управления

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Тема работы
Разработка библиотеки доступа к реляционным данным на языке LINQ с чтением схемы базы данных в режиме времени выполнения

УДК 004.652:004.4’2:004.428.4

Студент

Группа	ФИО	Подпись	Дата
8ВМ4В	Гаврилов Кирилл Александрович		

Руководитель

Должность	ФИО	Ученая степень, звание	Подпись	Дата
доцент каф. ОСУ	Чердынцев Е.С.	к.т.н., доцент		

КОНСУЛЬТАНТЫ:

По разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
доцент каф. менеджмента ИСГТ	Конотопский В.Ю.	к.э.н., доцент		

По разделу «Социальная ответственность»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
ассистент каф. ЭБЖ ИНК	Акулов П.А.	ассистент		

ДОПУСТИТЬ К ЗАЩИТЕ:

Зав. кафедрой	ФИО	Ученая степень, звание	Подпись	Дата
ОСУ	Иванов М.А.	к.т.н.		

Томск – 2016 г.

ПЛАНИРУЕМЫЕ РЕЗУЛЬТАТЫ ОБУЧЕНИЯ ПО ООП

Код результатов	Результат обучения (выпускник должен быть готов)
Общепрофессиональные компетенции	
Р1	Воспринимать и самостоятельно приобретать, развивать и применять математические, естественнонаучные, социально-экономические и профессиональные знания для решения нестандартных задач, в том числе в новой или незнакомой среде и в междисциплинарном контексте.
Р2	Владеть и применять методы и средства получения, хранения, переработки и трансляции информации посредством современных компьютерных технологий, в том числе в глобальных компьютерных сетях.
Р3	Демонстрировать культуру мышления, способность выстраивать логику рассуждений и высказываний, основанных на интерпретации данных, интегрированных из разных областей науки и техники, выносить суждения на основании неполных данных, анализировать профессиональную информацию, выделять в ней главное, структурировать, оформлять и представлять в виде аналитических обзоров с обоснованными выводами и рекомендациями.
Р4	Анализировать и оценивать уровни своих компетенций в сочетании со способностью и готовностью к саморегулированию дальнейшего образования и профессиональной мобильности. Владеть, по крайней мере, одним из иностранных языков на уровне социального и профессионального общения, применять специальную лексику и профессиональную терминологию языка.
Профессиональные компетенции	
Р5	Выполнять инновационные инженерные проекты по разработке аппаратных и программных средств автоматизированных систем различного назначения с использованием современных методов проектирования, систем автоматизированного проектирования, передового опыта разработки конкурентно способных изделий.
Р6	Планировать и проводить теоретические и экспериментальные исследования в области проектирования аппаратных и программных средств автоматизированных систем с использованием новейших достижений науки и техники, передового отечественного и зарубежного опыта. Критически оценивать полученные данные и делать выводы.
Р7	Осуществлять авторское сопровождение процессов проектирования, внедрения и эксплуатации аппаратных и программных средств автоматизированных систем различного назначения.
Общекультурные компетенции	
Р8	Использовать на практике умения и навыки в организации исследовательских, проектных работ и профессиональной эксплуатации современного оборудования и приборов, в управлении коллективом.

Код результатов	Результат обучения (выпускник должен быть готов)
Р9	Осуществлять коммуникации в профессиональной среде и в обществе в целом, активно владеть иностранным языком, разрабатывать документацию, презентовать и защищать результаты инновационной инженерной деятельности, в том числе на иностранном языке.
Р10	Совершенствовать и развивать свой интеллектуальный и общекультурный уровень. Проявлять инициативу, в том числе в ситуациях риска, брать на себя всю полноту ответственности.
Р11	Демонстрировать способность к самостоятельному обучению новым методам исследования, к изменению научного и научно-производственного профиля своей профессиональной деятельности, способность самостоятельно приобретать с помощью информационных технологий и использовать в практической деятельности новые знания и умения, в том числе в новых областях знаний, непосредственно не связанных со сферой деятельности, способность к педагогической деятельности.

Министерство образования и науки Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Институт кибернетики
Направление подготовки 09.04.01 «Информатика и вычислительная техника»
Кафедра оптимизации систем управления

УТВЕРЖДАЮ:
Зав. кафедрой ОСУ
_____ _____ _____М.А.Иванов___
(Подпись) (Дата) (Ф.И.О.)

ЗАДАНИЕ
на выполнение выпускной квалификационной работы

В форме:

магистерской диссертации

(бакалаврской работы, дипломного проекта/работы, магистерской диссертации)

Студенту:

Группа	**ФИО**
8ВМ4В	Гаврилову Кириллу Александровичу

Тема работы:

Разработка библиотеки доступа к реляционным данным на языке LINQ с чтением схемы базы данных в режиме времени выполнения	
Утверждена приказом директора (дата, номер)	

Срок сдачи студентом выполненной работы:	25.05.2016

ТЕХНИЧЕСКОЕ ЗАДАНИЕ:

Исходные данные к работе	Объектом исследования является взаимодействия пользовательского приложения с базой данных, схема которой неизвестна на этапе компиляции приложения. Цель исследования – адаптация существующей или разработка собственной библиотеки, предоставляющей возможность взаимодействия с базой данных, схема которой известна только на этапе компиляции и может меняться за время жизни приложения. Вид разрабатываемого программного обеспечения – динамически подключаемая библиотека Характер работы – периодический.

Перечень подлежащих исследованию, проектированию и разработке вопросов	1. Исследование проблемы взаимодействия объектно-ориентированной среды с реляционной системой управления базами данных 2. Обзор существующих ORM систем и их возможностей для работы со схемой базы данных, неизвестной на этапе компиляции приложения 3. Разработка библиотеки, позволяющей организовать работу с базой данных, схема которой известна только во время выполнения приложения 4. Анализ результатов
Перечень графического материала	Диаграммы взаимодействия с базой данных, архитектура решения и результаты работы

Консультанты по разделам выпускной квалификационной работы

Раздел	Консультант
Финансовый менеджмент, ресурсоэффективность и ресурсосбережение	Конотопский В.Ю.
Социальная ответственность	Акулов П.А.
Раздел на иностранном языке	Куркан Н.В.

Названия разделов, которые должны быть написаны на русском и иностранном языках:

Раздел 3. Разработка библиотеки (Development of library)

Дата выдачи задания на выполнение выпускной квалификационной работы по линейному графику	15.03.2016

Задание выдал руководитель:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент кафедры ОСУ	Чердынцев Е.С.	к.т.н., доцент		

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8ВМ4В	Гаврилов Кирилл Александрович		

Министерство образования и науки Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Институт кибернетики
Направление подготовки 09.04.01 «Информатика и вычислительная техника»
Уровень образования магистратура
Кафедра оптимизации систем управления
Период выполнения весенний семестр 2015/2016 учебного года

Форма представления работы:

магистерская диссертация

(бакалаврская работа, дипломный проект/работа, магистерская диссертация)

КАЛЕНДАРНЫЙ РЕЙТИНГ-ПЛАН
выполнения выпускной квалификационной работы

Срок сдачи студентом выполненной работы:	06 июня 2016 г.

Дата контроля	Название раздела (модуля) / вид работы (исследования)	Максимальный балл раздела (модуля)
01.04.2016	*Обзор проблемы взаимодействия объектно-ориентированной среды с реляционными данными*	*5*
15.04.2016	*Обзор возможностей готовых решений*	*10*
06.05.2016	*Проектирование и реализация библиотеки для работы с динамической схемой базы данных*	*25*
10.05.2016	*Финансовый менеджмент, ресурсоэффективность и ресурсосбережение*	*20*
13.05.2016	*Социальная ответственность*	*20*
20.05.2016	*Приложение А. Development of library*	*20*

Составил преподаватель:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент кафедры ОСУ	Чердынцев Е.С.	к.т.н., доцент		

СОГЛАСОВАНО:

Зав. кафедрой	ФИО	Ученая степень, звание	Подпись	Дата
ОСУ	М.А. Иванов	к.т.н.		

ЗАДАНИЕ ДЛЯ РАЗДЕЛА
«ФИНАНСОВЫЙ МЕНЕДЖМЕНТ, РЕСУРСОЭФФЕКТИВНОСТЬ И РЕСУРСОСБЕРЕЖЕНИЕ»

Студенту:

Группа	ФИО
8ВМ4В	Гаврилову Кириллу Александровичу

Институт	кибернетики	Кафедра	ОСУ
Уровень образования	магистр	Направление/специальность	09.04.01 «Информатика и вычислительная техника»

Исходные данные к разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»:	
1. *Стоимость ресурсов научного исследования (НИ): материально-технических, энергетических, финансовых, информационных и человеческих*	Стоимость персонального компьютера, стоимость лазерного принтера и расходных материалов к нему, стоимость канцелярских принадлежностей.
2. *Нормы и нормативы расходования ресурсов*	Нормальный срок полезного использования вычислительной техники, оргтехники.
3. *Используемая система налогообложения, ставки налогов, отчислений, дисконтирования и кредитования*	Проценты отчислений в Пенсионный фонд РФ, Фонд обязательного медицинского страхования РФ, Фонд социального страхования РФ.
Перечень вопросов, подлежащих исследованию, проектированию и разработке:	
1. *Оценка коммерческого и инновационного потенциала НТИ*	
2. *Разработка устава научно-технического проекта*	
3. *Планирование процесса управления НТИ: структура и график проведения, бюджет, риски и организация закупок*	
4. *Определение ресурсной, финансовой, экономической эффективности*	
Перечень графического материала *(с точным указанием обязательных чертежей):*	
1. Доли участия исполнителей проекта в запланированных работах 2. Распределение трудоемкости запланированных работ по исполнителям 3. Линейный график запланированных работ 4. Изменение степени готовности проекта в процессе выполнения запланированных работ 5. Структура цены проекта 6. Оценка научно-технического уровня проекта	

Дата выдачи задания для раздела по линейному графику	14.03.2016

Задание выдал консультант:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
доцент каф. менеджмента ИСГТ	Конотопский В.Ю.	к.э.н., доцент		

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8ВМ4В	Гаврилов Кирилл Александрович		

ЗАДАНИЕ ДЛЯ РАЗДЕЛА
«СОЦИАЛЬНАЯ ОТВЕТСТВЕННОСТЬ»

Студенту:

Группа	ФИО
8ВМ4В	Гаврилову Кириллу Александровичу

Институт	кибернетики	Кафедра	ОСУ
Уровень образования	магистр	Направление/специальность	09.04.01 «Информатика и вычислительная техника»

Исходные данные к разделу «Социальная ответственность»:	
Характеристика объекта исследования (технология) и области его применения.	*Разработка библиотеки доступа к реляционным данным на языке LINQ с чтением схемы базы данных в режиме времени выполнения. Данная разработка будет применяться при написании новых приложений.*
Перечень вопросов, подлежащих исследованию, проектированию и разработке:	
1 Профессиональная социальная ответственность. *1.1 Анализ вредных и опасных факторов, которые может создать объект исследования.* *1.2 Анализ вредных и опасных факторов, которые могут возникнуть на производстве при внедрении объекта исследования.* *1.3 Обоснование мероприятий по защите персонала предприятия от действия опасных и вредных факторов.*	*1.1 Описание вредных факторов производства при работе с ПЭВМ.* *1.2 Описание опасных факторов: опасность поражения электрическим током и опасность возникновения пожара* *1.3 Применение коллективных и индивидуальных средств защиты, а также мероприятий, снижающих воздействие вредных и опасных производственных факторов согласно нормативным документам.*
2 Экологическая безопасность. *2.1 Анализ влияния объекта исследования на окружающую среду.* *2.2 Анализ влияния процесса эксплуатации объекта на окружающую среду.* *2.3 Обоснование мероприятий по защите окружающей среды.*	*2.1 Объект исследования не оказывает воздействия на окружающую среду, но негативное влияние оказывает ПЭВМ, используемая человеком для работы с объектом исследования.* *2.2 Описание влияния на окружающую среду плат и компонентов ПЭВМ.* *2.3 Описание организационных мер по утилизации плат и компонентов ПЭВМ согласно нормативным документам.*
3 Безопасность в чрезвычайных ситуациях. *3.1 Анализ вероятных ЧС, которые может инициировать объект исследований.*	*3.1 Объект исследований не может инициировать ЧС, но находится в рабочей среде офисных помещений.*

3.2 Анализ вероятных ЧС, которые могут возникнуть на производстве при внедрении объекта исследований. *3.3 Обоснование мероприятий по предотвращению ЧС и разработка порядка действия в случае возникновения ЧС.*	*3.2 Описание наиболее вероятной ЧС (пожар), которая может возникнуть при внедрении и в процессе эксплуатации объекта исследований.* *3.3 Описание организационных мер по предотвращению пожара, приведение порядка действий при возникновении пожара.*
4 Правовые и организационные вопросы обеспечения безопасности. *4.1 Специальные (характерные для проектируемой рабочей зоны) правовые нормы трудового законодательства.* *4.2 Организационные мероприятия при компоновке рабочей зоны.*	*4.1 Описание правовых норм для работ, связанных с взаимодействием с ПЭВМ согласно ТК РФ.* *4.2 Описание влияния правильно организованной рабочей зоны на увеличение степени безопасности и удобства работы и повышение производительности труда.*

Дата выдачи задания для раздела по линейному графику	31.03.2016

Задание выдал консультант:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
ассистент каф. ЭБЖ	Акулов П.А.			

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8ВМ4В	Гаврилов Кирилл Александрович		

РЕФЕРАТ

Выпускная квалификационная работа 124 с., 23 рис., 26 табл., 45 источников, 3 прил.

Ключевые слова: язык запросов LINQ, работа с динамической схемой базы данных, серверная фильтрация данных

Объектом исследования является взаимодействия .NET приложения с базой данных, схема которой неизвестна на этапе компиляции приложения и может быть получена только во время выполнения.

Цель работы – разработка библиотеки доступа к реляционным данным на языке LINQ с чтением схемы базы данных в режиме времени выполнения.

В процессе исследования проводились исследования возможностей, существующих ORM на предмет поддержки работы с динамической схемой базы данных, проектирование архитектуры библиотеки, реализация алгоритма динамического построения LINQ запросов с поддержкой серверной фильтрации данных.

В результате исследования разработана библиотека доступа к реляционным данным на языке LINQ с чтением схемы базы данных в режиме времени выполнения.

Область применения: разработанная библиотека может применяться разработчиками для работы с базой данных, схема которой неизвестна на этапе проектирования и реализации клиентского приложения.

Экономическая эффективность и значимость работы заключаются в том, что при умеренные изменения, вносимые в базу данных, никак не скажутся на её взаимодействии с клиентом и не потребуют написания дополнительного кода или перекомпиляции.

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ, СОКРАЩЕНИЯ, НОРМАТИВНЫЕ ССЫЛКИ

В настоящей работе использованы ссылки на следующие стандарты:

1. ГОСТ 12.0.003 – 74 Система стандартов безопасности труда. Опасные и вредные производственные факторы. Классификация.
2. ГОСТ Р 12.1.019 – 2009 ССБТ Электробезопасность. Общие требования и номенклатура видов защиты.
3. ГОСТ 19.101 – 77 Единая система программной документации. Виды программ и программных документов.
4. ГОСТ 19.401 – 78 Единая система программной документации. Текст программы. Требования к содержанию и оформлению.
5. ГОСТ 19.402 – 78 Единая система программной документации. Описание программы.
6. ГОСТ 19.404 – 79 Единая система программной документации. Пояснительная записка.
7. ГОСТ 19.502 – 78 Единая система программной документации. Описание применения. Требования к описанию и оформлению.
8. ГОСТ 19.504 – 79 Единая система программной документации. Руководство программиста. Требования к содержанию и оформлению.
9. ГОСТ 28388 – 89 Система обработки информации. Документы на магнитных носителях данных. Порядок выполнения и обращения.

В данной работе применены следующие термины с соответствующими определениями:

.NET Framework: Технология, которая поддерживает создание и выполнение нового поколения приложений и веб-служб XML.

Application Programming Interface (API): Интерфейс программирования приложений (иногда интерфейс прикладного программирования).

C#: Объектно-ориентированный язык программирования.

Common Intermediate Language (CIL): «Высокоуровневый» ассемблер, промежуточный язык виртуальной машины среды .NET.

Common Language Runtime (CLR): Общеязыковая исполняющая среда для байт-кода CIL.

Database Schema: Схема базы данных, включающая в себя описание содержания, структуры и ограничений целостности базы данных.

Dynamic Link Library (DLL): Динамическая библиотека, использование которой возможно различными программными приложениями.

eXtensible Application Markup Language (XAML): Расширяемый язык разметки для приложений, основанный на XML, предназначен для декларативного программирования приложений.

eXtensible Markup Language (XML): Расширяемы язык разметки, который предназначен для удобного создания и обработки документов программами, обладающий простым и легко читаемым синтаксисом.

Graphical User Interface (GUI): Разновидность пользовательского интерфейса, в котором элементы интерфейса, представленные пользователю на дисплее, исполнены в виде графических изображений.

Integrated development environment (IDE): Интегрированная среда разработки, комплекс средств, необходимых для разработки программного обеспечения.

IntelliSense: Технология автодополнения, используемая в IDE Microsoft Visual Studio, использующаяся также для доступа к документации и устранения неоднозначности в именах классов, методов, свойств и переменных.

Language-Integrated Query (LINQ): Набор шаблонных функций для выполнения запросов и обновления данных.

Object-Relational Mapping (ORM): Технология программирования, связывающая концепции реляционных баз данных с концепциями объектно-ориентированных языков программирования посредством создания виртуальной объектной базы данных.

Structured Query Language (SQL): Структурированный язык запросов, непроцедурный язык, применяемый для управления данными реляционной базы данных.

Windows Presentation Foundation (WPF): Система для построения пользовательских приложений операционной системы Windows с широкими возможностями настройки графического пользовательского интерфейса, использующая язык XAML.

виртуализация: Процесс обработки только такого объема информации, который загружается в видимую область интерфейса, с целью снижения общего объема загружаемых данных и повышения скорости работы.

маппинг или маэппинг: Процесс сопоставления взаимодействия двух и более технологий.

В настоящей работе использованы ссылки на следующие обозначения и скоращения:

БД – база данных;

ООП – объектно-ориентированное программирование;

ПК – персональный компьютер;

ПО – программное обеспечение;

ПОбл – предметная область;

СУБД – система управления базами данных.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Взаимодействие с базой данных является неотъемлемой частью современных приложений. При использовании платформы.NET Framework для разработки пользовательского приложения перед разработчиками возникает проблема перехода от реляционных данных СУБД к объектно-ориентированной модели. Для облегчения этой задачи существует LINQ to SQL – API-интерфейс, позволяющий формировать запросы на языках .NET для обращения к СУБД.

При этом традиционным является подход, в котором связи между таблицами базы и классами приложения известны на этапе разработки и компиляции, что позволяет установить однозначное соответствие. Однако существует класс задач, для которых полный набор таблиц СУБД не может быть известен на этапе компиляции и может меняться за время жизни приложения.

Объектом исследования является взаимодействия .NET приложения с базой данных, схема которой неизвестна на этапе компиляции приложения и может быть получена только во время выполнения. Цель работы – разработка библиотеки доступа к реляционным данным на языке LINQ с чтением схемы базы данных в режиме времени выполнения.

Новизна разработки заключается в том, что на данный момент не существует каких-либо готовых решений, которые позволяют достаточно просто организовать такое взаимодействие.

Данная разработка помогает снизить зависимость приложения от структуры СУБД и исключить необходимость перекомпиляции для актуализации клиента. При этом следует отметить, что благодаря динамическому формированию запроса, в него можно делать вставки, позволяющие выполнять фильтрацию на SQL сервере СУБД, что значительно повышает скорость работы приложения.

Данное решения полностью реализовано и апробировано на тестовом проекте, показав при этом высокую стабильность и скорость работы. В дальнейшем планируется использовать разработку при реализации проектов

компании Rubius. Промежуточные результаты исследования представлены на всероссийских и международных научных конференциях.

1 ВЗАИМОДЕЙСТВИЕ ОБЪЕКТНО-ОРИЕНТИРОВАННОЙ СРЕДЫ И РЕЛЯЦИОННЫХ ДАННЫХ

Взаимодействие клиентского приложения с базой данных является неотъемлемой частью современных приложений. Для правильной организации такого взаимодействия требуется рассмотреть основы концепций реляционных баз данных и объектно-ориентированных языков программирования и описать существующие проблемы.

1.1 Проблема связи между реляционной и объектно-ориентированной концепциями

В методологии объектно-ориентированного программирования (ООП) классы, описывающие свойства и поведение сущностей, являются абстрактным описанием объектов реального мира. Например, если рассматривать упрощённый вариант журнала успеваемости по дисциплине, то все объекты класса «Студент» будут содержать поля ФИО и массив оценок (баллов). При этом время жизни объектов данного класса ограничивается временем работы приложения. Поэтому возникает вопрос хранения таких объектов: объекты должны храниться в файлах или базах данных, быть легко извлекаемыми, с сохранением своих свойств и отношений с другими объектами.

Решением проблемы хранения данных таким образом является использованием реляционных систем управления базами данных (СУБД). Однако использование реляционной СУБД для хранения данных объектно-ориентированной среды приводит к нарушению семантики (так называемому семантическому разрыву) [1], в результате чего проходиться разрабатывать программное обеспечение (ПО), которое должно как поддерживать работу с программными объектами, так и уметь сохранять данные этих объектов в реляционной форме. Постоянная необходимость преобразования формы данных снижает скорость работы решения и оказывает дополнительную нагрузку на разработчиков.

В основе работы реляционных СУБД лежит набор таблиц, содержащих простые данные. Связанная информация хранится в других таблицах, а связь осуществляется по внешнему ключу. Если объект достаточно сложен, то для его хранения может использоваться более десятка таблиц, а для получения всей информации об объекте требуется применять множество операций JOIN. Это требует от разработчика глубоких знаний не только принципов ООП, но и концепций СУБД, иначе выполнение множества запросов может быть крайне неоптимальным и достаточно затратным по времени. Поддержка такого кода из клиента также является нетривиальной задачей, так как обычно сам запрос храниться в строковых переменных, и его правильность никак не проверяется компилятором, а значит является потенциальным местом множества ошибок [2].

Перед тем как рассмотреть возможные варианты решения данной проблемы, необходимо определить перечень технологий, на которых будет базироваться проект. На сегодняшний день существует огромное количество платформ, технологий и языков программирования, поэтому одну и ту же задачу можно решить несколькими способами. Однако по тем или иным причинам (операционная система, уже установленное программное обеспечение, на которое оплачены дорогостоящие лицензии и выполнена настройка, личные предпочтения пользователей) еще до начала исследования существует ограниченный набор технологий, которые ложатся в основу разработки.

Основными технологиями, на которых будет базироваться решение являются: платформа .NET Framework версии 4.0 (для совместимости с операционной системой Windows XP), объектно-ориентированный язык программирования C#. В серверной части используется система управления базами данных MySQL, клиентское приложение взаимодействует с БД посредством языка выражений LINQ, для отображения данных используется пользовательский элемент управления Telerik RadGridView.

1.2 Обзор требуемых технологий

.NET Framework – это программная платформа, выпущенная компанией Microsoft в 2002 году. Особенностью данной платформы является то, что код, написанный на любом языке программирования платформы, сперва компилируется в единый промежуточный байт-код, называемый Common Intermediate Language (CIL), в результате компиляции создается сборка (assembly). Этот код исполняется виртуальной машиной Common Language Runtime (CLR), либо транслируется утилитой NGen.exe в исполняемый код для конкретного целевого процессора. [3]. При этом предпочтительнее использовать виртуальную машину, поскольку при таком подходе с разработчиков снимаются потенциальные проблемы из-за особенностей аппаратной части, заботы об управлении памятью, базовой безопасности и системе исключений. В этом случае JIT-компилятор, встроенный в CLR «на лету» преобразуется байт-код в машинные команды нужного процессора. Данное решение является достаточно быстрым, учитывая современной развитие технологий динамической компиляции.

Архитектура .NET Framework описана и опубликована в спецификации Common Language Infrastructure (CLI), разработанной Microsoft и утверждённой ISO и ECMA. В CLI описаны типы данных .NET, формат метаданных о структуре программы, система исполнения байт-кода и многое другое.

Объектные классы .NET, доступные для всех поддерживаемых языков программирования, содержатся в библиотеке Framework Class Library (FCL). В FCL входят классы Windows Forms, ADO.NET, ASP.NET, Language Integrated Query, Windows Presentation Foundation, Windows Communication Foundation и другие. Ядро FCL называется Base Class Library (BCL) [3].

C# – объектно-ориентированный язык программирования, созданный корпорацией Microsoft для разработки приложений для платформы .NET Framework. C# разрабатывался как язык программирования прикладного уровня для CLR, поэтому выразительные особенности языка в первую очередь зависят от CLR и определяются тем, может ли языковая особенность быть транслирована

в конструкции CLR. При этом CLR предоставляет всем .NET-ориентированным языкам ряд возможностей, такие как поддержка ООП, сборка мусора, безопасность типов, структурная обработка исключений. Несмотря на то, что C# предназначен для генерации кода, выполняемого средой .NET, это полноценный язык программирования.

MySQL – это система управления реляционными базами данных. БД представляет собой структурированную совокупность данных. Для записи, выборки и обработки данных, хранящихся в компьютерной базе данных, необходима система управления базой данных. В реляционной базе данные хранятся в отдельных таблицах, благодаря чему достигается выигрыш в скорости и гибкости. Таблицы связываются между собой при помощи отношений, благодаря чему обеспечивается возможность объединять при выполнении запроса данные из нескольких таблиц.

LINQ – это набор шаблонных функций для выполнения запросов и обновления данных. Для взаимодействия с базой данных используется LINQ to SQL –API-интерфейс для работы с базами данных SQL Server.

1.3 ORM как средство связи приложения и БД

Как уже было сказано выше – использование двух концепций оказывает дополнительную нагрузку на разработчиков, так как повышается сложность кода для работы с СУБД, увеличивается риск появления ошибок. Для решения такой проблемы появились ORM фрэймворки.

ORM – это технология разработки, которая связывает базы данных с концепциями объектно-ориентированных языков программирования [4]. Ключевой особенностью ORM является отображение, которое используется для привязки объекта к его данным в БД. ORM как бы создает «виртуальную» схему базы данных в памяти и позволяет манипулировать данными уже на уровне объектов. Отображение показывает, как объект и его свойства связанны с одной или несколькими таблицами и их полями в базе данных. ORM использует информацию этого отображения для управления процессом преобразования

данных между базой и формами объектов, а также для создания SQL-запросов для вставки, обновления и удаления данных в ответ на изменения, которые приложение вносит в эти объекты.

Из всего этого следует, что для использования ORM .NET разработчику не требуется знаний языка SQL для работы с реляционными данными, так как все изменения сущностей объектно-ориентированной среды ORM самостоятельно будет заносить в нужные таблицы БД.

1.4 Эффект от использования ORM

Внедрение любой технологии в проект привносит ряд изменений в скорость работы приложения, сложность написания и поддержки кода. В данном разделе рассмотрен результат внедрения ORM Entity Framework версии 6.1.3 [5].

Для того чтобы проанализировать изменения в коде, достаточно сравнить код, приведенный на рисунках Рисунок 1.1 и Рисунок 1.2. На рисунке Рисунок 1.1 показан запрос сущности из базы данных напрямую из кода, а на рисунке Рисунок 1.2 – с использование ORM.

```
public static Customer LoadCustomer(int id)
{
    const string sql = @"SELECT * FROM Customers WHERE ID = @ID";
    using (var connection = new SqlConnection())
    {
        connection.ConnectionString =
            @"Data Source=.\sqlexpress;Initial Catalog=mydbname;User ID=myuser;Password=mypwd";
        connection.Open();
        using (var command = new SqlCommand(sql, connection))
        {
            command.Parameters.AddWithValue("ID", id);
            using (var reader = command.ExecuteReader())
            {
                var customer = new Customer
                {
                    Id = id,
                    Name = reader.GetString((reader.GetOrdinal("Name")))
                };
                return customer;
            }
        }
    }
}
```

Рисунок 1.1 – Запрос сущности из БД напрямую

```
public static Customer LoadCustomer(int id)
{
    using (var context = new CustomerEntities())
    {
        return context.Customers.FirstOrDefault(t => t.Id == id);
    }
}
```

Рисунок 1.2 – Запрос сущности из БД с помощью ORM

Основные преимущества использования ORM [4, 6]:

– для реализации абсолютно одинакового функционала требуется написать в разы меньше кода;
– не нужно писать SQL запросы;
– в продвинутых ORM не нужно самому создавать объекты и базу, возможно сгенерировать код для объектов, имея готовую БД, или наоборот, создать базу данных по набору готовых классов сущностей;
– приложения, использующие ORM легче поддерживать.

В итоге использование ORM влияет на разработку следующим образом:

– сокращает время разработки – для работы с базой, содержащей 25-30 таблиц, необходимо завести 50-80 объектов, а это примерно от 7000 до 10000 строк кода, которые надо написать и протестировать. С ORM то же можно сделать за 1-2 дня;
– позволяет создать универсальный код – разные люди в команде могут создать свой уникальный код для преобразования объектов из/в БД, опираясь исключительно на свой опыт. ORM использует шаблоны кода, которые имеют отличный дизайн;
– сокращает время тестирования – программисту нет необходимости вручную преобразовывать объекты из/в базу данных, а код, который делает такое преобразование, написан и уже протестирован;
– упрощает сопровождение – жизнь программы не заканчивается выпуском релиза. Она живёт и развивается, и изменение кода в случае с ORM менее затратное занятие, так как написано меньше кода [6].

Однако использование ORM оказывает негативное влияние на скорость работы приложения. Например, получение большего набора данных (около 1000 объектов) может выполняться более 5 секунд, а добавление в таблицу 500 записей занимает порядка 30 секунд. Слой транзакций (выполняющий преобразования) недостаточно эффективен, поэтому при обработке большого количества данных работа приложения будет медленнее, чем при использовании чистого SQL. Но большинство запросов, не требующих обработки больших объёмов данных, выполняется через ORM за вполне приемлемое время, а учитывая все плюсы – использование ORM может стать очень выгодным и разумным решением.

Данные утверждения применимы не только для Entity Framework, а для любой ORM, с той лишь разницей, что чем мощнее ORM, чем дальше она позволяет абстрагироваться от БД и предоставляет программисту больше удобных инструментов, тем ниже скорость ее работы. Еще одним эффектов внедрения ORM является смещение ошибок в коде: программист пишет меньше кода сам, этот код достаточно типизирован, и допустить ошибку в таком коде сложнее, однако код ORM тоже содержит ошибки, о которых программист даже не подозревает. Для крупных и распространенных ORM этот недостаток некритичен, так как очевидные ошибки обнаруживаются сразу большим количеством людей и быстро правятся разработчиками. Если же проблема находится в архитектуре ORM, то создаются дополнительные библиотеки, позволяющие частично нивелировать такие недостатки.

1.5 Обзор литературы

Перед тем как приступить к решению проблемы взаимодействия клиентского приложения с БД, необходимо рассмотреть способы решения схожих задач. Это позволит определить ряд проблем, которые могут возникнуть в процессе проектирования и разработки, а именно:

- организация взаимодействия с SQL сервером;
- получение актуальной схемы БД во время выполнения приложения;

- применения серверной фильтрации при использовании LINQ запросов;
- генерация классов объектов ООП во время выполнения;
- использование ORM в проектах.

Основы программирования на языке C# описаны в [7-9]. В данных источниках достаточно подробно описана механика работы платформы .NET Framework версии 4.0 и 4.5, а также синтаксис языка C# с примерами.

В [10] приводятся примеры достижения максимального быстродействия при реализации алгоритмов, циклов, асинхронной работы и управлению ресурсами при программировании на C#.

В [11-16] описана работа с БД. В [11] кратко рассмотрены ключевые особенности языка LINQ to SQL, а именно – работа с объектами типа IQuaryable<Type> вместо IEnumerable<T>. Так как запросы LINQ to SQL возвращают последовательность типа IQueryable<T>, они не компилируются в код промежуточного языка .NET, как это делают обычные запросы LINQ. В отличие от запросов LINQ, которые выполняются в памяти локальной машины запросы LINQ to SQL преобразуются в деревья выражений, что позволяет им вычисляться как единое целое и транслироваться в соответствующие и оптимальные конструкции SQL. В [12] рассмотрены особенности синтаксиса при написании запросов LINQ to SQL и примеры компиляции нетранслируемого кода запросов, в которых ошибки проявляют себя только при попытке получить результат запроса (вызов неподдерживаемых методов на столбцах таблицы, а не на параметрах).

В [13] рассматриваются языки LINQ и Dynamic LINQ, динамическое формирование LINQ запросов, парсинг лямбда выражений, сферы применения и варианты использования Dynamic LINQ.

В [14-16] подробно рассмотрен интерфейс IDataReader. В [14] описан сам интерфейс. В [15] подробнее рассмотрен вызов метода GetSchemaTable для получения данных о таблицах БД. В [16] рассмотрен вариант реализации интерфейса IDataProvider для чтения данных из БД.

В [17-19] рассмотрена кодогенерация, а именно примеры динамической компиляции кода из файлов и строк, содержащих, в том числе, LINQ операторы, в DLL. В [19] код для генерации записывается прямо в текстовое поле, что позволяет проверить его корректность перед компиляцией, однако исключает возможность полностью автоматической генерации.

В [6, 20-21] рассмотрено использование ORM в проекте. В [6] приведено обзорное сравнение LINQ to SQL, Entity Framework и nHibernate. В [20-21] выполнен сравнительный анализ быстродействия платных и бесплатных ORM.

По результатам обзора текущего развития .NET технологий очевидно, что в основе решения должны быть следующие компоненты: интерфейс IDataReader для получения информации о структуре таблицы БД, автоматическая динамическая кодогенерация для класса сущности с последующей компиляцией, язык LINQ to SQL для возможности обработки запросов на стороне сервера. В дальнейшей предстоит проанализировать, существуют ли готовые или частично готовые средства для реализации требуемого функционала, или его необходимо разрабатывать.

2 АНАЛИЗ ВОЗМОЖНОСТЕЙ СУЩЕСТВУЮЩИХ ORM ДЛЯ РАБОТЫ С ДИНАМИЧЕСКОЙ СХЕМОЙ БАЗЫ ДАННЫХ

Схема базы данных – это её структура, описанная на формальном языке, поддерживаемом СУБД. В реляционных базах данных схема определяет таблицы, поля в каждой таблице (обычно с указанием их названия, типа, обязательности), и ограничения целостности (первичный, потенциальные и внешние ключи и другие ограничения [22].

Обычно при разработке приложений схема базы данных определяется на этапе проектирования, незначительно корректируется на этапе разработки и неизменна во время выполнения приложения. Традиционным способом взаимодействия клиентского приложения с базой является создание сущностей, являющихся отражением таблиц и представлений реляционной СУБД в объектно-ориентированной среде. Этот способ очень удобен, так как существует большое количество ORM фрэймворков, автоматически преобразующих данные их объектно-ориентированной среды в сущности таблиц БД. Однако любое изменение в СУБД потребует, как минимум, изменения в коде сущностей, связанных с этой таблицей, а потенциально – переписывание большого количества кода алгоритмов и логики отображения в пользовательском интерфейсе.

При этом существует класс задач, для решения которых необходимо со временем изменять схему (например, добавлять новые таблицы, удалять неактуальные). Использование неактуального слоя сущностей для работы с обновленной БД неизбежно повлечет ошибки в работе (большинство ORM генерируют исключения при обнаружении несоответствий в связке «схема данных-свойства класса»). Для решения данной проблемы необходимо использовать такое решение, которому неважна изначальная схема БД. Такое решение позволит взаимодействовать со схемой, созданной не на момент компиляции приложения, а во время его выполнения.

Создание даже простой ORM является достаточно трудозатратным процессом, поэтому оптимальным вариантом является использование

функционала уже разработанных ORM для реализации поставленной задачи. При этом существует такой вариант, что полностью готового решения найти не удастся.

На данный момент существует более 10 ORM для платформы .NET. Однако часть из них (очень широко функциональная DataObjects.NET, ECO) являются коммерческими продуктами [23]. Недостатком большинства других продуктов является их слабый функционал, низкая распространенность, отсутствие какой-либо документации и технической поддержки. Поэтому в рамках данной работы рассматриваются 4 ведущие [24] свободно распространяемые ORM для платформы .NET (Entity Framework, NHibernate, Telerik Data Access и LINQ to SQL), и анализируется, насколько их функционал подходит для решения поставленной задачи.

Если рассматривать уже существующие решения, то нельзя не отметить, что изначально эти ORM проектировались под статическую схему базы данных. Применение динамических схем в большинстве ORM либо невозможно, либо достаточно тяжело в использовании. При этом нельзя забывать, что это накладывает дополнительные временные расходы, на и без того не самую быструю работу ORM. При этом нельзя с уверенностью сказать, что ORM поддерживает LINQ полностью, а исходный код не содержит ошибок, что может стать дополнительной неприятной неожиданностью.

2.1 NHibernate

NHibernate – это бесплатное ORM-решение с открытым исходным кодом для платформы Microsoft .NET, портированное с Java [25].

NHibernate поддерживает различные варианты маппингов: с использованием XML-файлов, с использование пользовательских атрибутов, маппинг в коде и маппинг «на лету» [26].

2.1.1 Маппинг с использованием XML-файлов

Маппинг с использованием XML-файлов является самым первым из разработанных маппингов. Для настроек подключения требуется создать файл Nhibernate.cfg.xml. Код для подключения приведен на рисунке Рисунок 2.1.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
        <session-factory>
                <property name="connection.provider">
                        NHibernate.Connection.DriverConnectionProvider
                </property>
                <property name="connection.driver_class">
                        NHibernate.Driver.SqlClientDriver
                </property>
                <property name="connection.connection_string">
                        Server=...\SQLENTERPRISE; database=NhibernateTutor; Integrated Security=SSPI;
                </property>
                <property name="dialect">
                        NHibernate.Dialect.MsSql2008Dialect
                </property>
        </session-factory>
</hibernate-configuration>
```

Рисунок 2.1 – Код для подключения к БД

Для .NET класса с virtual свойствами (для реализации отложенной загрузки и отслеживания изменений) требуется создать xml-файл с расширением *.nbm.xml. Структура маппинга приведена на рисунке Рисунок 2.2.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
        <class name="Book" dynamic-update="true" >
                <cache usage="read-write"/>
                <id name="Id" type="int">
                        <generator class="native" />
                </id>
                <property name="Name" />
                <property name="Description" />
        </class>
</hibernate-mapping>
```

Рисунок 2.2 – Маппинг с помощью XML-файла

После этого в специальном вспомогательном классе требуется явно указать связь между .NET классом и файлом конфигурации.

Плюс такого подхода в том, что в сети Интернет есть множество примеров. Однако при выборе данного варианта маппинга полностью отсутствует поддержка IntelliSense и валидация во время компиляции.

2.1.2 **Маппинг с использованием атрибутов**

Данный подход является дополнением к Nhibernate. Для каждого свойства .NET класса требуется указать атрибут, при этом для свойств, состоящих более чем из одного атрибута, требуется указать индексы. При этом классы маппинга (в формате *.hbm.xml) не требуются, так как они создаются «на лету», именно поэтому требуется задавать индексы атрибутов.

Плюсы данного подхода:

- не требуется создавать отдельные файлы (*.hbm.xml), сущность и маппинг находятся рядом (атрибуты внутри класса сущности);
- частичная поддержка Intellisense, есть подсказки для написания атрибутов класса (таких как Name), но нет для его свойств, которые представлены в виде строки;
- на данный маппинг легко перейти с xml-файлов.

Минусы использования маппинга атрибутов:

- .NET код становится менее читаемым;
- отсутствует валидация во время компиляции.
- у свойств, состоящих из более 1 атрибута следует прописывать индексы.

2.1.3 **Маппинг в коде**

При данном подходе информация об подключении хранится во вспомогательном классе среды .NET, поэтому файл Nhibernate.cfg.xml больше не требуется и его можно удалить.

Плюсы данного подхода:

- не требуются дополнительных библиотек (как в случае с атрибутами);
- полная поддержка IntelliSense;

- не требуются *.hbm.xml-файлы и Nhibernate.cfg.xml.

Минусом является то, что структура (в частности отношения между классами) не совсем точно соответствует элементам *.hbm.xml-файлов.

2.1.4 **Маппинг «на лету»**

Вместо того, чтобы писать XML-файлы, требуется прописать маппинги в строго типизированном C# коде (через лямбда-выражения). Благодаря этому есть поддержка удобного рефакторинга, улучшенная читаемость и легкость написания кода.

Плюсами данного подхода является:

- полная поддержка IntelliSense;
- подробная документация Fluent-Nhibernate;
- валидация на этапе компиляции;
- не требуется создавать файл Nhibernate.cfg.xml, все настройки, включая строку подключения, можно прописать во вспомогательном классе.

Минусом является несколько иной синтаксис для указания маппинга.

Необходимо учесть, что данная ORM изначально проектировалась под статическую схему БД. Любые изменения базы требуют корректировки данных в клиенте. В данной работе рассмотрено 4 варианта маппинга, однако все они являются статическими и создают жесткую связь между БД и клиентским приложением. Данная ORM не способна легко решить поставленную задачу.

2.2 **Entity Framework**

При использовании Entity Framework существует 3 подхода три подхода к проектированию базы данных: Database-First, Model-First, Code-First [27].

2.2.1 **Code-First**

Code-First подходит для программистов – при данном подходе модель EDMX вообще не используется, и классы C# объектной модели настраиваются вручную (данный подход поддерживает как генерацию сущностных классов из существующей базы данных, так и создание базы данных из созданной вручную

модели объектов C#). Очевидно, что это подходит для программистов, хорошо знакомых с синтаксисом C#.

При использовании такого подхода сначала требуется описать модель в коде, а после этого, на основе готовой модели создать (или обновить) базу данных. Изначально классы модели не имеют ничего общего с Entity Framework и являются описанием структуры бизнес-модели, которая используется в приложении. Для связи классов сущностей с Entity Framework требуется создать класс контекста. Entity Framework имеет 2 базовых класса контекста: ObjectContext (более общий класс, используется с момента создания ORM) и DbContext (появился в версии 4.1, поддерживает Code-First, но более сложен в использовании). С помощью класса контекста осуществляются запросы на чтение, добавление, изменение и удаление записей в таблицах. В конструктор класса DbContext передается либо имя БД, либо строка подключения. Генерация БД происходит при первом вызове метода контекста SaveChanges.

2.2.2 **Model-First**

Model-First подходит для архитекторов – сначала создаётся графическая модель EDMX в Visual Studio (в фоновом режиме создаются классы C# модели), а затем, на основе этой модели, генерируется БД. При данном подходе не нужно знать ни деталей T-SQL, ни синтаксиса C#.

Model-First полностью автоматически создает .NET код модели данных, код класса контекста и код генерации базы данных. При этом код классов сущностей и класса контекста будет отличаться от Code-First только названиями по умолчанию (в Code-First при создании класса явно указывается его имя).

Это и является главной концепцией работы с Entity Framework – разрабатывать модель и создавать базу данных возможно разными способами, но, когда дело доходит до использования классов контекста для работы с данными, код является одинаковым и не зависит от подхода.

2.2.3 **Database-First**

Database-First подходит для проектировщиков БД – сначала с помощью различных инструментов создаётся БД, а затем генерируется EDMX-модель БД, которая предоставляет удобный графический интерфейс для взаимодействия с базой данных в виде диаграмм и объектная модель в виде классов C#. Данный подход требует хорошего знания синтаксиса T-SQL, но при этом не нужно разбираться в C#.

Первым шагом при работе с подходом Database-First является проектирование базы данных. Фактически, подход Database-First является противоположным подходу Model-First: при подходе Model-First сначала создается графическую модель, а затем на ее основе генерируется или изменяется база данных и наоборот, при подходе Database-First сперва создается и проектируется БД, а затем на ее основе создается графическая модель. Visual Studio извлекает информацию из БД, а через некоторое время, мастер Entity Data Model Wizard попросит выбрать элементы базы данных, которые нужно использовать в приложении.

Рассмотрев все 3 подхода к работе с Entity Framework можно заметить, что независимо от очередности создания (сущности, база или EDMX модель) всегда требуется однозначное сопоставление между сущностями и таблицами. Ни один из подходов также не предоставляет возможности привязывать таблицы БД к динамическим сущностям объектной модели. К тому же Entity Framework имеет несколько крайне неприятных особенностей поведения: простая вставка 500 записей в таблицу БД может занимать порядка 30 секунд.

2.3 **Telerik Data Access**

Достаточно мощная ORM от компании Telerik. Каких-либо русскоязычных ресурсов по данной ORM на сегодняшний день не существует. Одна официальный сайт содержит достаточно подробную информацию, с описанием все основных преимуществ использования данного продукта и

руководство программиста. Данное руководство будет использоваться для того, чтобы оценить пригодность данной ORM для решения поставленной задачи.

На первом этапе создаются классы сущностей, которые в последующем будут связаны с определенными таблицами БД. Здесь же требуется создать следующие классы:

- класс для контекста (аналог DbContext от Entity Framework). Данный класс реализует интерфейс IUnitOfWork, код которого приведен на рисунке Рисунок 2.3.

```
public interface IFluentModelUnitOfWork : IUnitOfWork
{
    IQueryable<RentalRate> RentalRates
    {
        get;
    }
    IQueryable<RentalOrder> RentalOrders
    {
        get;
    }
    IQueryable<Employee> Employees
    {
        get;
    }
    IQueryable<Customer> Customers
    {
        get;
    }
    IQueryable<Category> Categories
    {
        get;
    }
    IQueryable<Car> Cars
    {
        get;
    }
}
```

Рисунок 2.3 – Код класса контекста

- достаточно объемный класс маппинга, в котором прописываются связи между классами сущностей и таблицами [28].

Исходя из этого кода очевидно, что еще на этапе компиляции четко определяется структура БД и классов сущностей. Однако Telerik Data Access поддерживает добавление типов или полей для привязывания «на лету». На официальном сайте указывается, что «возможно добавить новые классы, добавить новые свойства существующих классов, сопоставить их со столбцами

базы данных и начать использовать их сразу без каких-либо головных болей» [29].

В примере на рисунке Рисунок 2.4 создаются два искусственных типа – FluentModel.Product и FluentModel.Category, используя Generic метод HasArtificialPrimitiveProperty<T>, задаются первичные ключи.

```
public class FluentModelMetadataSource : FluentMetadataSource
{
   protected override IList<MappingConfiguration> PrepareMapping()
   {
       List<MappingConfiguration> configurations = new List<MappingConfiguration>();

       MappingConfiguration productConfiguration = new MappingConfiguration( "Product", "FluentModel" );
       productConfiguration.HasArtificialPrimitiveProperty<int>( "Id" ).IsIdentity();

       MappingConfiguration categoryConfiguration = new MappingConfiguration( "Category", "FluentModel" );
       categoryConfiguration.HasArtificialPrimitiveProperty<int>( "Id" ).IsIdentity();

       MappingConfiguration<TestClass> testConfiguration = new MappingConfiguration<TestClass>();
       testConfiguration.MapType();
       testConfiguration.HasProperty( x => x.ID ).IsIdentity();

       configurations.Add( testConfiguration );
       configurations.Add( productConfiguration );
       configurations.Add( categoryConfiguration );

       return configurations;
   }
}
```

Рисунок 2.4 – Код маппинга Telerik Data Access

Из примера видно, что в основе работы TDA всё также лежит работа с классами, созданными на момент компиляции, и для добавления новой сущности потребуется переписать класс маппинга, а значит – пересобрать приложение. Такой вариант не сможет решить поставленную задачу.

2.4 Dynamic LINQ to SQL

Еще одним вариантом решения задачи является использование ORM LINQ to SQL, которая является предшественником Entity Framework. При этом LINQ to SQL отказано развитии, но для этой ORM создано расширение, а именно Dynamic LINQ to SQL [30].

Основная идея данной библиотеки – возможность вставки в лямбда выражения языка LINQ строковых значений. Это новшество позволяет передавать внутрь оператора строковую конструкцию вместо делегата метода-предиката и выполнять ее в произвольный момент времени. В стандартном

LINQ, равно как и в любой из трех рассмотренных ORM такое выполнить было нельзя, так как для типизированной коллекции IEnumerable<T> должен существовать определенный тип T, а свойства, используемые в лямбда выражении должны быть свойствами класса T. В общем случае компилятор в момент сборки выдаст ошибку, связанную с отсутствующим классом (свойством, если используется наследование).

В целом использование Dynamic LINQ to SQL выглядит достаточно подходящим инструментом, с помощью которого возможно реализовать поставленную задачу.

3 ФИНАНСОВЫЙ МЕНЕДЖМЕНТ, РЕСУРСОЭФФЕКТИВНОСТЬ И РЕСУРСОСБЕРЕЖЕНИЕ

Цель раздела – комплексное описание и анализ финансово-экономических аспектов выполненной работы. Необходимо оценить полные денежные затраты на исследование (проект), а также дать хотя бы приближенную экономическую оценку результатов ее внедрения. Это в свою очередь позволит с помощью традиционных показателей эффективности инвестиций оценить экономическую целесообразность осуществления работы. Раздел должен быть завершен комплексной оценкой научно-технического уровня ВКР на основе экспертных данных.

3.1 Организация и планирование работ

При организации процесса реализации конкретного проекта необходимо рационально планировать занятость каждого из его участников и сроки проведения отдельных работ.

Для этого требуется определить полный перечень работ. Для каждого из выделенных этапов определим его исполнителей и долю участия каждого из исполнителей в его выполнении. При этом следует отметить, что исполнителей всего двое – научный руководитель (НР) и инженер (И), являющийся исполнителем ВКР. Полученные результаты сведем в таблицу 4.1Таблица 3.1.

Таблица 3.1 – Перечень работ и продолжительность их выполнения

Этапы работы	Исполнители	Загрузка исполнителей
Постановка целей и задач, получение исходных данных	НР	НР – 100%
Составление и утверждение ТЗ	НР, И	НР – 100% И – 25%
Составление и утверждение календарного плана работ.	НР, И	НР – 100% И – 30%
Подбор и изучение материалов по тематике, обзор существующих решений, их возможностей	НР, И	НР – 40% И – 100%
Обсуждение литературы	НР, И	НР – 60% И – 100%
Разработка архитектуры ПО	НР, И	НР – 100% И – 70%

Продолжение таблицы 4.1Таблица 3.1

Выбор технологий, используемых в ПО	НР, И	НР – 80% И – 100%
Разработка ПО	И	И – 100%
Обсуждение проблем и вариантов решения	НР, И	НР – 80% И – 100%
Анализ узких мест и способов повышения производительности	НР, И	НР – 80% И – 100%
Доработка ПО	И	И – 100%
Оформление расчетно-пояснительной записки	И	И – 100%
Оформление графического материала	И	И – 100%
Подведение итогов	НР, И	НР – 60% И – 100%

3.1.1 **Продолжительность этапов работ**

Ввиду отсутствия нормативной базы трудоемкости планируемых процессов воспользуемся опытно-статистическим методом расчета продолжительности работ. По причине отсутствия достоверной информации о процессах выполнения схожих проектов расчет выполним экспертным способом.

Для этого, исходя из собственного опыта, для каждого этапа определим минимальное и максимальное время выполнения. Затем, воспользовавшись следующей формулой, рассчитаем ожидаемое время выполнения работ:

$$t_{ож} = \frac{3 \cdot t_{\min} + 2 \cdot t_{\max}}{5},$$

где t_{min} – минимальная продолжительность работы, дн.;

t_{max} – максимальная продолжительность работы, дн.;

$t_{ож}$ – ожидаемое время выполнения этапа работ в чел.-дн.

Для построения линейного графика необходимо рассчитать длительность этапов в рабочих днях, а затем перевести ее в календарные дни. Расчет продолжительности выполнения каждого этапа в рабочих днях ($T_{РД}$) ведется по формуле:

$$T_{РД} = \frac{t_{ож}}{K_{ВН}} \cdot K_{Д},$$

где $t_{ож}$ – продолжительность работы, дн.;

$K_{ВН}$ – коэффициент выполнения работ, учитывающий влияние внешних факторов на соблюдение предварительно определенных длительностей, примем $K_{ВН} = 1$;

$K_{Д}$ – коэффициент, учитывающий дополнительное время на компенсацию непредвиденных задержек и согласование работ, примем $K_{Д} = 1{,}2$.

Для перевода рабочих дней в календарные воспользуемся формулой:

$$T_{КД} = T_{РД} \cdot T_{К},$$

где $T_{КД}$ – продолжительность выполнения этапа в календарных днях;

$T_{К}$ – коэффициент календарности (для шестидневной рабочей недели), позволяющий перейти от длительности работ в рабочих днях к их аналогам в календарных днях, и рассчитываемый по формуле:

$$T_{К} = \frac{T_{КАЛ}}{T_{РАБ}} = \frac{T_{КАЛ}}{T_{КАЛ} - T_{ВД} - T_{ПД}},$$

где $T_{КАЛ}$ – календарные дни ($T_{КАЛ} = 366$);

$T_{ВД}$, $T_{ПД}$ – выходные и праздничные дни (66, согласно производственному календарю для шестидневной рабочей недели на 2016 год).

Коэффициент календарности для данной работы:

$$T_{К} = \frac{366}{366 - 66} = 1{,}22.$$

Исходя из данных таблицы 4.1Таблица 3.1, воспользуемся приведенными выше формулами и рассчитаем продолжительность выполнения работ исполнителями проекта в календарных днях. Рассчитанные данные внесены в таблицу 4.2Таблица 3.2, а на основе этих данных строится линейный график работ, приведенный в приложении Б.

Таблица 3.2 – Трудозатраты на выполнение проекта

Этап	Исполнители	Продолжительность работ, дни			Трудоемкость работ по исполнителям чел.- дн.			
					$T_{РД}$		$T_{КД}$	
		t_{min}	t_{max}	$t_{ож}$	НР	И	НР	И
1	2	3	4	5	6	7	8	9
Постановка целей и задач, получение исходных данных	НР	1,00	2,00	1,40	1,68	-	2,05	-
Составление и утверждение ТЗ	НР, И	3,00	5,00	3,80	4,56	1,14	5,56	1,39
Составление и утверждение календарного плана работ.	НР, И	2,00	5,00	3,20	3,84	1,15	4,68	1,41
Подбор и изучение материалов по тематике, обзор существующих решений, их возможностей	НР, И	3,00	6,00	4,20	2,02	5,04	2,46	6,15
Обсуждение литературы	НР, И	1,00	2,00	1,40	1,01	1,68	1,23	2,05
Разработка архитектуры ПО	НР, И	2,00	4,00	2,80	3,36	2,35	4,10	2,87
Выбор технологий, используемых в ПО	НР, И	2,00	6,00	3,60	3,46	4,32	4,22	5,27
Разработка ПО	И	15,00	25,00	19,00	-	22,80	-	27,82
Обсуждение проблем и вариантов решения	НР, И	1,00	3,00	1,80	1,73	2,16	2,11	2,64
Анализ узких мест и способов повышения производительности	НР, И	3,00	5,00	3,80	3,65	4,56	4,45	5,56
Доработка ПО	И	5,00	10,00	7,00	-	8,40	-	10,25
Оформление расчетно-пояснительной записки	И	5,00	10,00	7,00	-	8,40	-	10,25
Оформление графического материала	И	2,00	4,00	2,80	-	3,36	-	4,10
Подведение итогов	НР, И	1,00	2,00	1,40	1,01	1,68	1,23	2,05
Итого				63,20	26,30	67,04	32,09	81,79

3.1.2 Расчет накопления готовности проекта

Для оценки текущего состояния готовности проекта необходимо рассчитать величину накопления готовности работы, которая показывает, на сколько процентов по окончании текущего (*k*-го) этапа выполнен общий объем работ по проекту в целом. Данную величину будем считать по формуле:

$$СГ_i = \frac{TP_i^H}{TP_{общ}} = \frac{\sum_{k=1}^{i} TP_k}{TP_{общ}} = \frac{\sum_{k=1}^{i}\sum_{j=1}^{m} TP_{k,j}}{TP_{общ}},$$

где $TP_{k,j}$ – трудоемкость *k*-го этапа работ для *j*-го исполнителя;

$TP_{общ}$ – общая трудоемкость проекта для всех исполнителей;

i, m – количество этапов работы и исполнителей соответственно.

Данные для расчета указаны в таблице 4.2Таблица 3.2 (столбцы 6 и 7). Результат расчета представлен в таблице 4.3Таблица 3.3.

Таблица 3.3 – Нарастание технической готовности проекта

Этап	TPi, %	СГi, %
Постановка целей и задач, получение исходных данных	1,68	1,80
Составление и утверждение ТЗ	5,70	7,91
Составление и утверждение календарного плана работ.	4,99	13,25
Подбор и изучение материалов по тематике, обзор существующих решений, их возможностей	7,06	20,81
Обсуждение литературы	2,69	23,69
Разработка архитектуры ПО	5,71	29,81
Выбор технологий, используемых в ПО	7,78	38,14
Разработка ПО	22,80	62,57
Обсуждение проблем и вариантов решения	3,89	66,73
Анализ узких мест и способов повышения производительности	8,21	75,52
Доработка ПО	8,40	84,52
Оформление расчетно-пояснительной записки	8,40	93,52
Оформление графического материала	3,36	97,12
Подведение итогов	2,69	100,00

3.2 Расчет сметы затрат на выполнение проекта

В состав затрат на создание проекта включается величина всех расходов, необходимых для реализации комплекса работ, составляющих содержание

данной разработки. Расчет сметной стоимости ее выполнения производится по следующим статьям затрат:

– материалы и покупные изделия;
– заработная плата;
– социальный налог;
– расходы на электроэнергию (без освещения);
– амортизационные отчисления;
– командировочные расходы;
– оплата услуг связи;
– арендная плата за пользование имуществом;
– прочие услуги (сторонних организаций);
– прочие (накладные расходы) расходы.

Следует отметить, что при выполнении работ исполнители не используют услуги сторонних организаций, не арендуют имущество, а также не несут командировочных расходов и расходов на оплату связи.

3.2.1 Расчет затрат на материалы и покупные изделия

Ввиду того что планируемые работы включают в себя проведение теоретических исследований, разработку программного обеспечения и не предполагают изготовление устройств, в состав оборудования входят персональный компьютер и принтер, которые были приобретены заранее, поэтому их стоимость не учитывается в материальных затратах. Программное обеспечение является бесплатным для всех студентов НИ ТПУ. Поэтому материальные затраты включают в себя только затраты на расходные материалы для принтера (картридж и бумага) и канцелярские принадлежности.

Дополнительно следует учесть транспортно-заготовительные расходы, (ТЗР), обусловленные затратами на совершение сделок купли-продажи материалов, на их доставку к месту использования. Обычно транспортно-заготовительные расходы оцениваются в 5...20% от цены материалов. Норма ТЗР принята на уровне $k_{ТЗР} = 0{,}1$.

Структура и величина материальных затрат приведена в таблице 4.4Таблица 3.4.

Таблица 3.4 – Расчет затрат на материалы

Наименование материалов	Цена за ед., руб.	Кол-во	Сумма, руб.
Картридж лазерный HP	4799	1	4799
Бумага для принтера, упак. 500 листов	230	1	230
Папка-скоросшиватель	81	2	162
Мультифоры, упак. 100 шт.	100	2	200
Тетрадь общая формата А4, 96листов	112	1	112
Ручка шариковая, синяя	55	3	165
Итого			5668
Итого (с учетом ТЗР)			6234,8

3.2.2 **Расчет заработной платы**

Данная статья расходов включает заработную плату научного руководителя и инженера. Для расчета заработной платы инженера примем величину месячного оклада равную окладу инженера кафедры $МО_1$ = 7864,12 *руб*. для расчета заработной платы руководителя примем величину месячного оклада равную окладу доцента, к.н. $МО_2$ = 23264,86 *руб*.

Среднедневная тарифная заработная плата ($ЗП_{дн\text{-}т}$) рассчитывается по формуле:

$$ЗП_{дн-т} = \frac{МО}{25},$$

где 25 – среднее число рабочих дней в месяце (300 рабочих дней в году).

Зная среднедневную ставку и количество часов, необходимых на реализацию проекта, можно вычислить затраты на полную заработную плату, рассчитываемую по формуле:

$$ЗП_{полн} = ЗП_{дн-т} \cdot Т_{РД} \cdot К_{ПР} \cdot К_{доп} \cdot К_{Р},$$

где $Т_{РД}$ – трудоемкость проекта для сотрудника в рабочих днях; $К_{ПР} = 1{,}1$ – коэффициент премирования; $К_{доп} = 1{,}188$ – коэффициент дополнительной заработной платы для шестидневной рабочей недели; $К_{Р} = 1{,}3$ – районный

коэффициент. Выполним расчеты по приведенным формулам, результаты сведем в таблицу 4.5Таблица 3.5.

Таблица 3.5 – Затраты на заработную плату

Исполнитель	Оклад, руб./мес.	Среднедневная ставка, руб./раб.день	Затраты времени, раб.дни	Коэффициент	Фонд з/платы, руб.
НР	23264,86	930,59	26	1,70	41104,21
И	7864,12	314,56	67	1,70	35804,48
Итого					76908,69

3.2.3 **Расчет затрат на социальный налог**

Затраты на единый социальный налог (ЕСН), включающий в себя отчисления в пенсионный фонд, на социальное и медицинское страхование, составляют 30 % от полной заработной платы по проекту, $C_{соц.}$ = 76908,69 * 0,3 = 23072,61 *руб*.

3.2.4 **Расчет затрат на электроэнергию**

Данный вид расходов включает в себя затраты на электроэнергию, потраченную в ходе выполнения проекта на работу используемого оборудования, рассчитываемые по формуле:

$$C_{эл.об.} = P_{об} \cdot t_{об} \cdot Ц_э,$$

где $P_{об}$ – мощность оборудования;

$t_{об}$ – время работы оборудования;

$Ц_э = 5{,}257\ руб/кВт \cdot ч$ – цена 1кВт·ч электроэнергии.

Время работы оборудования определяется по формуле:

$$t_{об} = T_{РД} \cdot K_t,$$

где $T_{РД}$ – трудоемкость проекта в часах (рабочий день – 8 часов);

K_t – коэффициент загрузки оборудования (равен доле рабочего времени, в течении которой использовалось оборудование; определяется исполнителем).

Мощность, потребляемая оборудованием, определяется по формуле:

$$P_{об} = P_{НОМ} \cdot K_c,$$

где $P_{ном.}$ – номинальная мощность оборудования, кВт;

$K_C \leq 1$ – коэффициент загрузки, зависящий от средней степени использования номинальной мощности. Для технологического оборудования малой мощности $K_C = 1$.

Для выполнения работы инженер использовал персональный компьютер и лазерный принтер. Затраты на электроэнергию приведены в таблице 4.6Таблица 3.6.

Таблица 3.6 – Затраты на электроэнергию

Наименование оборудования	Потребляемая мощность, кВт	K_t	Время работы оборудования, час	Затраты на электроэнергию, руб.
Персональный компьютер	1,00	0,90	482,72	2537,64
Лазерный принтер	2,00	0,01	5,36	28,20
Итого				2565,84

3.2.5 **Расчет амортизационных расходов**

Амортизационные расходы рассчитываются по формуле:

$$C_{АМ} = \frac{H_A \cdot Ц_{ОБ} \cdot t_{РФ} \cdot n}{F_Д},$$

где H_A – годовая норма амортизации оборудования;

$Ц_{ОБ}$ – стоимость единицы оборудования;

$t_{РФ}$ – фактическое время работы единицы оборудования в ходе выполнения проекта;

n – количество однотипных единиц оборудования;

$F_Д$ – действительный годовой фонд времени работы оборудования.

При выполнении проекта использовалось по одной единице двух видов оборудования: персональный компьютер и лазерный принтер. Оба типа оборудования входят в одну группу – вычислительная техника и имеют одинаковый срок полезного использования – 2-3 года. Выберем срок полезного использования 2,5 года; тогда норма амортизации составит 0,4. Цены компьютера и принтера равны 35000 *руб*. и 5000 *руб*. соответственно. Данные о

времени использования возьмем из таблицы 5.6. Годовой фонд времени работы составляет 2400 часов. Воспользуемся приведенной формулой для расчета амортизационных расходов; результаты расчетов сведем в таблицу 4.7Таблица 3.7.

Таблица 3.7 – Расчет амортизационных расходов

Наименование оборудования	H_A, $г^{-1}$	$Ц_{ОБ}$, руб.	$t_{РФ}$, час	n	$F_Д$, час	$C_{АМ}$, руб.
Компьютер персональный	0,4	35000	482,72	1	2400	2815,85
Принтер лазерный	0,4	5000	5,36	1	2400	4,47
Итого						2820,32

3.2.6 **Расчет прочих расходов**

В данной статье отражены расходы на выполнение проекта, которые не учтены в предыдущих статьях, их следует принять равными 10% от суммы всех предыдущих расходов, т.е.:

$$C_{проч} = (C_{мат} + C_{зп} + C_{вн.взн} + C_{эл.об.} + C_{АМ}) \cdot 0{,}1 =$$
$$= (6234{,}8 + 76908{,}69 + 23072{,}61 + 2565{,}84 + 2820{,}32) \cdot 0{,}1 =$$
$$= 111602{,}3 \cdot 0{,}1 = 11160{,}23.$$

3.2.7 **Расчет общей себестоимости разработки**

Проведя расчет по всем статьям сметы затрат на разработку, можно определить общую себестоимость проекта, которая представлена в таблице 4.8Таблица 3.8.

Таблица 3.8 – Смета затрат на разработку проекта

Статья затрат	Условное обозначение	Сумма, руб.
Материалы и покупные изделия	$C_{мат}$	6234,8
Основная заработная плата	$C_{зп}$	76908,69
Отчисления в социальные фонды	$C_{соц}$	23072,61
Расходы на электроэнергию	$C_{эл}$	2562,84
Амортизационные отчисления	$C_{АМ}$	2820,32

Продолжение таблицы Таблица 3.8

Непосредственно учитываемые расходы	$C_{нр}$	0
Прочие расходы	$C_{проч}$	11160,23

Итого	122762,48

3.2.8 Расчет прибыли

Ввиду отсутствия данных для применения «сложных» методов расчета прибыли заложим в проект прибыль в размере 15% от себестоимости:

$$P = 0{,}15 \cdot C_{полн} = 18414{,}37\,руб.$$

3.2.9 Расчет НДС

НДС составляет 18% от суммы затрат на разработку и прибыли:

$$НДС = (P + C_{полн}) \cdot 0{,}18 = 25411{,}83\,руб.$$

3.2.10 Цена разработки НИР

Цена равна сумме полной себестоимости, прибыли и НДС:

$$Ц_{НИР} = 122762{,}48 + 18414{,}37 + 25411{,}83 = 166588{,}68\,руб.$$

3.3 Оценка экономической эффективности проекта

Актуальным аспектом качества выполненного проекта является экономическая эффективность его реализации, т.е. соотношение обусловленного ей экономического результата (эффекта) и затрат на разработку проекта.

Так как существует множество направлений инвестиционных средств, каждое из которых различается по отдаче, а инвестиционный фонд является ограниченным, необходимо качественно и количественно проанализировать исходное множество проектов с целью отбора ограниченного множества наиболее эффективных.

С помощью качественного анализа можно радикально ограничить круг перспективных проектов, однако для формирования окончательного множества, подлежащего реализации его недостаточно. Поэтому он дополнятся количественным анализом, в котором используется ряд расчетных показателей, которые позволяют отсортировать оставшиеся проекты с точки зрения их экономической эффективности (соотношение величины инвестиций с экономических эффектом от реализации).

Для начала определим экономический эффект от реализации проекта. Инвестором проекта является ИТ-компания, которая рассчитывает сократить расходы на разработку продуктов посредством внедрения нового способа взаимодействия с базами данных.

3.3.1 Расчет экономического эффекта инвестиций

Экономический эффект от внедрения новых технологий может быть лишь косвенным, так как эти технологии не являются прямым источником дохода, а помогают минимизировать затраты. Величина экономического эффекта определяется по формуле:

$$Э = P - З,$$

где Р – стоимостная оценка результатов, полученных фирмой в результате внедрения мероприятий за расчетный период;

З – стоимостная оценка затрат фирмы на осуществление внедрения мероприятий за расчетный период.

Затраты являются единовременными и равны сумме объема инвестиций. Исходя из того, что стоимость одного человеко-часа для компании обходится в 900 рублей, и учитывая примерное количество сэкономленных человеко-часов по годам (200, 450 и 150), рассчитаем экономический эффект от внедрения. Расчет выполним двумя способами: без учета (таблица 4.9Таблица 3.9) и с учетом ставки дисконтирования (таблица 4.10Таблица 3.10).

Таблица 3.9 – Экономический эффект без учета ставки дисконтирования

Год	Инвестиции	Экономия	Накопленный денежный поток
0	-167	0	-167
1		180	13
2		405	418
3		135	553

Здесь уже к первому году эффект от внедрения будет положительным, время окупаемости составляет 167/180=0,928 года, а экономический эффект за три года – 553 тысячи рублей.

Недостатком рассмотренного показателя является его относительный характер, он не отражает масштаб проекта и объем полученного результата.

Для более точного расчета будем использовать не просто данные об объеме сэкономленных средств, а и дисконтированные значения, полученные в результате деления на $(1 + i)^j$, где i – ставка дисконтирования, примем i=0,2.

Таблица 3.10 – Экономический эффект с учетом ставки дисконтирования

Год	Инвестиции	Номинальная экономия	Коэффициент дисконтирования $1/(1+0,2)^j$	Дисконтированная экономия	Накопленный денежный поток
0	-167	0	1	0	-167
1	0	180	0,8333	150	-17
2	0	405	0,6944	281,25	264,25
3	0	135	0,5787	78,125	342,375

Таким образом экономический эффект окажется положительным уже на втором году после создания (1+17/281,25=1,06), а спустя три года экономический эффект будет равен 342 тысячам рублей, что говорит о высокой экономической эффективности разработки.

3.3.2 **Оценка научно-технического уровня НИР**

Оценим научно-технический уровень разработки с помощью вычисления интегрального индекса научно-технического уровня (НТУ) $I_{НТУ}$. Данный индекс вычисляется как взвешенная сумма количественных оценок НИР по трем признакам: уровню новизны, теоретическому уровню, возможности реализации.

Весовые коэффициенты признаков НТУ приведены в таблице 4.11Таблица 3.11. Баллы оценки уровня новизны приведены в таблице 4.12Таблица 3.12; баллы значимости теоретического уровня – в таблице 4.13Таблица 3.13; баллы возможности реализации результатов по времени – в таблице 4Таблица 3.14.14.

Таблица 3.11 – Весовые коэффициенты признаков НТУ

Признаки научно-технического эффекта НИР	Характеристика признака НИР	Коэффициент
Уровень новизны	Систематизируются и обобщаются сведения, определяются пути дальнейших исследований	0,4
Теоретический уровень	Разработка способа (алгоритм, программа мероприятий, устройство, вещество и т.п.)	0,1
Возможность реализации	Время реализации в течение первых лет	0,5

Таблица 3.12 – Баллы оценки уровня новизны

Уровень новизны	Характеристика уровня новизны	Баллы
Принципиально новая	Новое направление в науке и технике, новые факты и закономерности, новая теория, вещество, способ	8-10
Новая	По-новому объясняются те же факты, закономерности, новые понятия дополняют ранее полученные результаты	5-7
Относительно новая	Систематизируются, обобщаются имеющиеся сведения, новые связи между известными факторами	2-4
Не обладает новизной	Результат, который ранее был известен	0

Таблица 3.13 – Баллы значимости теоретических уровней

Теоретический уровень полученных результатов	Баллы
Установка закона, разработка новой теории	10
Глубокая разработка проблемы, многоспектральный анализ взаимодействия между факторами с наличием объяснений	8
Разработка способа (алгоритм, программа и т. д.)	6
Элементарный анализ связей между фактами (наличие гипотезы, объяснения версии, практических рекомендаций)	2
Описание отдельных элементарных факторов, изложение наблюдений, опыта, результатов измерений	0,5

Таблица 3.14 – Баллы возможности реализации результатов по времени

Время реализации	Баллы
В течение первых лет	10
От 5 до 10 лет	4
Свыше 10 лет	2

Разработанное в ходе выполнение исследовательской работы решение для работы с СУБД использует разработанные ранее технологии и не является революционно новым. Однако аналогов у данного решения нет, поэтому уровень новизны можно оценить в 4 балла. Теоретический уровень разработки новых способов и алгоритмов можно оценить на 6 баллов. Возможность реализации НИР можно оценить на 10 баллов, так как работа выполнялась для нужд сторонней организации, а результат уже может быть использован в реализации новых проектов. Обоснования выбранных баллов приведены в таблице 4.15Таблица 3.15.

Таблица 3.15 – Оценки научно-технического уровня НИР

Значимость	Фактор НТУ	Уровень фактора	Балл	Обоснование выбранного балла
0,4	Уровень новизны	Относительно новая	4	Облегчит привязку клиентского приложения к СУБД, исключит необходимость правок кода и пересборки проекта
0,1	Теоретический уровень	Разработка способа (алгоритм, программа и т. д.)	6	Простой механизм взаимодействия с СУБД, схема которой известна только во время выполнения
0,5	Возможность реализации	В течение первых лет	10	Прототип уже реализован, решены основные проблемы, доработка до полноценного решения не займет много времени

Так как все частные признаки научно-технического уровня оцениваются по 10-балльной шкале, а сумма весов равна единице, то величина интегрального показателя также принадлежит интервалу [0, 10]. Используя данные из таблицы 4.15Таблица 3.15, рассчитаем интегральный индекс НТУ НИР:

$$I_{НТУ} = \sum_{i=1}^{3} k_i \cdot n_i = 0{,}4 \cdot 4 + 0{,}1 \cdot 6 + 0{,}5 \cdot 10 = 7{,}2$$

Таким образом, индекс НТУ равен 7,2 балла, данный проект имеет средний уровень научно-технического эффекта.

СПИСОК ПУБЛИКАЦИЙ СТУДЕНТА

Промежуточные результаты исследования представлены в следующих статьях:

1. XIII Всероссийская научно-практическая конференция студентов, аспирантов и молодых ученых «Технологии Microsoft в теории и практике программирования», название статьи: «Преимущества и недостатки использования ORM в разработке»;
2. III Международная научная конференция "Информационные технологии в науке, управлении, социальной сфере и медицине", название статьи: «Решение проблемы взаимодействия приложения с базой данных, имеющей динамическую схему».