

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

БАКАЛАВРСКАЯ РАБОТА
на тему
Программные средства взаимодействия с
СУБД для системы автозаполнения баз
данных

Студент	_____	<u>Шушарина Инна Сергеевна</u>
	(подпись, дата)	(ФИО полностью)
Руководитель	_____	<u>Шибанов С.В.</u>
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	_____	<u>Такташкин Д.В.</u>
	(подпись, дата)	(фамилия, инициалы)

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

Заведующий кафедрой	_____	<u>Макарычев П.П.</u>
	(подпись)	(фамилия, инициалы)

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

Секретарь ГЭК	_____	<u>Попова Н.А.</u>
	(подпись)	(фамилия, инициалы)

Пенза, 2020

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ П.П. Макарычев
«__» _____ 2020 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

1. Студент Шушарина Инна Сергеевна гр. 16ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программные средства взаимодействия с СУБД для системы автозаполнения баз данных

Тема утверждена приказом ПГУ № _____ от "_____" _____ 2020 г.

2. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Шибанов С.В.

3. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение: программное обеспечение предназначено для обеспечения взаимодействия приложения пользователя с СУБД.

Среда выполнения: операционная система Windows 10.

Языковые средства разработки: C#, SQL.

Инструментальные средства разработки: Microsoft Visual Studio 2019.

4. Перечень подлежащих разработке вопросов

Анализ предметной области и постановка задачи на разработку

Разработка требований к ПС взаимодействия с СУБД

Разработка ПС взаимодействия с СУБД

Тестирование ПС взаимодействия с СУБД

Планирование процесса разработки и расчет бюджета

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и постановка задачи на разработку	10	13.04.2020 – 19.04.2020	
2	Разработка требований к ПС взаимодействия с СУБД	15	20.04.2020 – 28.04.2020	
3	Разработка ПС взаимодействия с СУБД	40	29.04.2020 – 22.05.2020	
4	Тестирование ПС взаимодействия с СУБД	15	23.05.2020 – 01.06.2020	
5	Планирование процесса разработки и расчет бюджета	10	02.06.2020 – 07.06.2020	
6	Оформление пояснительной записки	10	08.06.2020 – 13.06.2020	

Дата выдачи задания « 30 » декабря 2019г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета _____
(подпись, дата)

Реферат

Пояснительная записка содержит 86 листов, 28 таблиц, 16 рисунков, 27 использованных источников и 3 приложения.

БД, АВТОЗАПОЛНЕНИЕ, ГЕНЕРАЦИЯ, НАПОЛНЕНИЕ, СЕРВЕР, ПЛАГИН, СПРАВОЧНИКИ, КЛАССИФИКАТОРЫ СУБД, БД, CSV, PLUG-IN, SQL, UML.

Объект разработки: программные средства взаимодействия с СУБД для системы автозаполнения баз данных

Цель работы: разработать программные средства взаимодействия с СУБД для системы автозаполнения баз данных.

Технология разработки – Microsoft Visual Studio 2019, язык программирования C#, язык запросов SQL, технология plug-in.

Результаты работы: в процессе разработки был проведен анализ предметной области с целью определения актуальности разрабатываемой системы, выполнено сравнение существующих систем и разработаны программные средства взаимодействия с СУБД для прототипа системы автозаполнения баз данных.

					ПГУ 09.03.02–8BP161.30 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Шушарина И.С.			«Программные средства взаимодействия с СУБД для системы автозаполнения баз данных» Пояснительная записка	Лит.	Лист	Листов
Провер.		Шибанов С.В.					4	86
Н. Контр.		Такташкин Д.В.				Группа 16ВИ1		
Утверд.		Макарычев П.П.						

Содержание

Введение	8
1 Анализ предметной области и постановка задачи на разработку	10
1.1 Автоматизированное заполнение баз данных информационных систем	10
1.1.1 Автозаполнение баз данных в жизненном цикле информационных систем	10
1.1.2 Современное состояние исследований проблемы автозаполнения баз данных	13
1.1.3 Актуальность разработки системы автозаполнения баз данных	15
1.2 Сравнительный анализ современных средств автоматизированного заполнения баз данных	15
1.2.1 Цели и задачи сравнительного анализа	15
1.2.2 Выбор средств автоматизированной генерации данных	16
1.2.3 Обзор средств для автоматизированной генерации данных	17
1.2.4 Результаты сравнительного анализа	25
1.3 Постановка задачи на разработку	26
1.3.1 Постановка задачи на разработку системы автозаполнения баз данных	26
1.3.1.1 Цели разработки и назначение системы автозаполнения баз данных	26
1.3.1.2 Пользователи системы автозаполнения баз данных	27
1.3.1.3 Автоматизируемые функции системы автозаполнения баз данных ..	28
1.3.1.4 Аппаратное и программное окружение системы автозаполнения баз данных	28

1.3.2	Постановка задачи на разработку программных средств взаимодействия с СУБД для системы автозаполнения баз данных	29
1.3.2.1	Цели разработки и назначение программных средств взаимодействия с СУБД для системы автозаполнения баз данных	29
1.3.2.2	Основные функции программных средств взаимодействия с СУБД для системы автозаполнения баз данных	30
1.3.2.3	Требования к программному интерфейсу.....	30
2	Разработка требований к программным средствам взаимодействия с СУБД для системы автозаполнения баз данных	32
2.1	Функциональные требования	32
2.2	Требования к программной архитектуре	36
2.2.1	Требования к обобщенной архитектуре системы.....	36
2.2.2	Требования к архитектуре программных средств взаимодействия с СУБД для системы автозаполнения баз данных.....	37
2.3	Требования к аппаратно-программному окружению программных средств взаимодействия с СУБД для системы автозаполнения баз данных ..	40
2.4	Обоснование выбора поддерживаемых СУБД	42
2.5	Обоснование выбора средств разработки программных средств взаимодействия с СУБД для системы автозаполнения баз данных	43
2.5.1	Обоснование выбора языковых средств разработки	43
2.5.2	Обоснование выбора инструментальных средств разработки.....	46
3	Разработка программных средств взаимодействия с СУБД для системы автозаполнения баз данных	48
3.1	Разработка системы классов ПС взаимодействия с СУБД	48
3.2	Разработка методов взаимодействия с серверами БД	52
3.3	Методика создания плагин-библиотеки для взаимодействия с новой СУБД.....	55

4	Тестирование программных средств взаимодействия с СУБД для системы автозаполнения баз данных	57
4.1	Обоснование выбора метода тестирования	57
4.2	Модульное тестирование	58
4.3	Расчет метрик программного обеспечения	60
5	Планирование процесса разработки и расчет бюджета	63
	Заключение	69
	Список использованных источников	70
	Приложение А. Листинг моделей представления данных	73
	Приложение Б. Листинг интерфейса плагинов	77
	Приложение А. Листинг плагина взаимодействия с СУБД MS SQL Server	79

Введение

В современном мире появляется все больше информационных систем, и в процессе их разработки возникает необходимость начального заполнения баз данных. Это требуется в следующих случаях:

- при переносе данных из предыдущей системы баз данных (БД);
- при тестировании нагрузки системы, когда требуется большое количество записей;
- при загрузке данных из справочников и классификаторов.

Актуальность выбранной темы связана с тем, что вручную внесение больших объемов тестовых данных в базы данных затруднительно. Решить задачу повышения качества и сокращения расходов на нагрузочное тестирование можно за счет привлечения эффективных средств автоматизации разработки тестовых данных.

Главной целью разработки является создание системы, которая будет автоматизировать процесс наполнения БД тестовыми данными. При этом должна учитываться поддержка различных систем управления базами данных (СУБД) реляционного типа.

Для достижения поставленной цели необходимо решить следующие задачи:

- выполнить анализ предметной области и постановку задачи на разработку;
- разработать требования к программным средствам взаимодействия с СУБД;
- разработать программные средства (ПС) взаимодействия с СУБД;
- провести тестирование разработанных программных средств взаимодействия с СУБД.

В ходе выполнения выпускной квалификационной работы был проведен сравнительный анализ систем-аналогов [1], а также выделены основные требования к реализуемой системе и ее функционал [2].

В результате выполнения работы были реализованы программные средства взаимодействия с СУБД, функционирующие в прототипе системы автоматизированного заполнения баз данных, которая позволяет:

- автоматизировать заполнение базы данных;
- автоматически генерировать случайные данные;
- загружать данные из справочников и классификаторов;
- перемещать данные между различными базами данных, в том числе под управлением разных СУБД;
- выполнять резервное копирование и восстановление данных.

В настоящий момент реализован прототип будущей системы, который осуществляет доступ к одной СУБД. В процессе развития прототипа планируется организовать возможность расширения числа подключаемых СУБД.

1 Анализ предметной области и постановка задачи на разработку

1.1 Автоматизированное заполнение баз данных информационных систем

1.1.1 Автозаполнение баз данных в жизненном цикле информационных систем

База данных – это важнейший компонент информационной системы (ИС), поэтому жизненные циклы (ЖЦ) приложения базы данных и информационной системы взаимосвязаны. ЖЦ ИС – это комплекс этапов развития ИС от принятия решения ее реализации до прекращения эксплуатации.

ЖЦ приложений БД включает в себя следующие этапы [3]:

- 1) планирование базы данных;
- 2) определение системы;
- 3) сбор и анализ требований;
- 4) проектирование базы данных:
 - концептуальное проектирование;
 - логическое проектирование;
 - физическое проектирование;
- 5) выбор СУБД;
- 6) проектирование приложений;
- 7) разработка прототипа;
- 8) реализация;
- 9) преобразование и загрузка данных;
- 10) тестирование;
- 11) эксплуатационное сопровождение.

Схема жизненного цикла приложений баз данных представлена на рисунке 1.

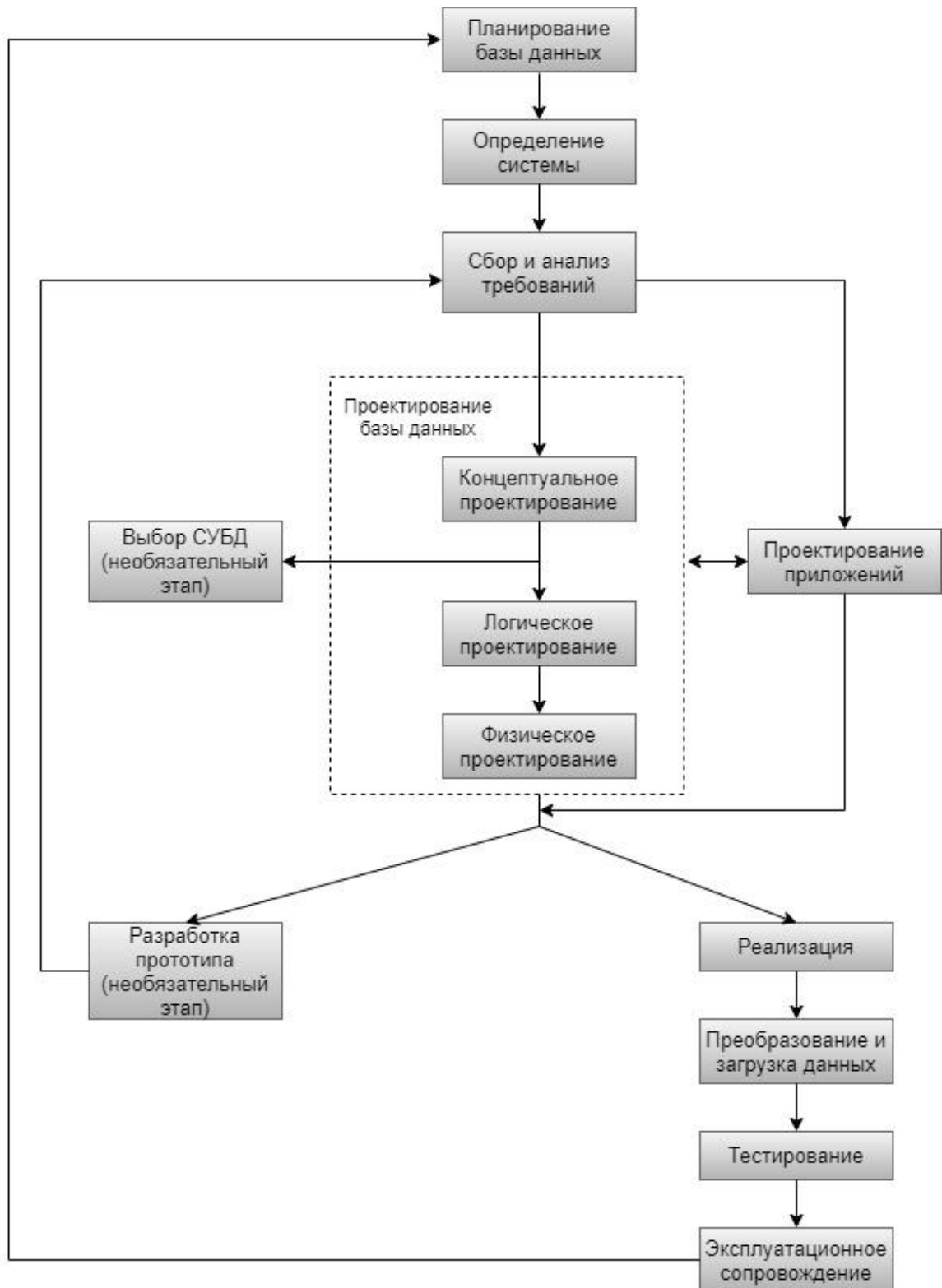


Рисунок 1 – Этапы жизненного цикла приложений баз данных

Планирование базы данных включает подготовительные действия для максимально эффективной реализации других этапов ЖЦ приложения БД.

Определение системы подразумевает определение диапазона действия приложения БД, его задач, состава пользователей и областей применения.

Сбор и анализ требований – это процесс получения и анализа информации о части организации, для которой будет создано приложение базы данных.

На этапе проектирования происходит создание проекта базы данных. Существует три этапа проектирования БД: концептуальное, логическое и физическое. Концептуальное проектирование базы данных – это процесс создания модели информации, которая не зависит от любых физических аспектов представления. В процессе логического проектирования БД создается модель информации на основе выбранной модели данных, но без учета типа целевой СУБД и других физических аспектов реализации. Физическое проектирование БД – это процесс описания реализации базы данных на вторичных запоминающих устройствах с учетом основных отношений, организации файлов и индексов, а также ограничений целостности и средств защиты.

Выбор СУБД подразумевает выбор целевой СУБД подходящего типа для поддержки реализуемого приложения базы данных.

Проектирование приложений включает проектирование интерфейса пользователя и прикладных программ, необходимых для работы с базой данных.

В процессе разработки прототипа создается рабочая модель приложения БД.

На этапе реализации происходит физическое создание базы данных и разработанных приложений.

Преобразование и загрузка данных – это процесс перемещения существующих данных в новую базу данных и модификация существующих приложений для организации совместной работы с новой БД.

Тестирование – это процесс выявления ошибок в прикладных программах.

Эксплуатационное сопровождение – это поддержка нормального функционирования системы по завершении ее развертывания.

Средства автоматизированного заполнения баз данных необходимы:

- 1) при начальной загрузке международных, общероссийских, отраслевых справочников и классификаторов на этапе реализации, что необходимо для реального функционирования ИС;
- 2) на этапе преобразования и загрузки данных из старой системы, когда из предыдущих версий ИС или резервных копий выбираются данные, которые также необходимы для реального функционирования ИС;
- 3) на этапе тестирования, когда осуществляется загрузка тестовых данных для оценки корректности и эффективности запросов к БД, функционального и комплексного тестирования приложений ИС, но по завершении необходимых проверок эти тестовые данные не сохраняются в БД.

Следует отметить, что применение средств автоматизированного заполнения баз данных играет важную роль в жизненном цикле приложений БД и приводит к повышению качества процессов, выполняемых на указанных этапах ЖЦ.

1.1.2 Современное состояние исследований проблемы автозаполнения баз данных

Анализ современного состояния исследований и разработок моделей, методов, алгоритмов и программных средств автозаполнения баз данных показывает актуальность данной темы и высокий интерес исследователей и разработчиков к ней. Это обусловлено сложностью задачи и востребованностью соответствующих инструментальных средств.

В [4] исследуется проблема оптимизации процесса генерации тестовых данных. Автор предлагает технологию тонкой настройки генерации данных, которая приводит к уменьшению объемов генерируемых данных и повышению их качества. Это достигается применением техники абстрактных моделей. Ограничения определяются с помощью итераторов – объектов, имеющих несколько последовательных состояний. В целом, генерация

данных сложной структуры сводится к генерации атрибутированных деревьев с наличием горизонтальных связей, что позволяет учитывать контекст элементов дерева. Ключевым моментом в схеме генерации является то, что для каждого типа вершин дерева имеется свой генератор, который появляется по требованию и зависит от контекста

В [5] сообщается о недостатках существующих инструментов. В случае генерации на основе реализации, отсутствует гарантия покрытия всех требуемых функциональных ветвей. Во многих современных инструментах такая гарантия возможна только при полном переборе комбинаций значений полей. Это приводит к временным затратам и при генерации, и при анализе полученных тестов. В качестве решения этих проблем, автор предлагает подход на основе формализации и анализа требований к тестируемой системе, который поддерживается генератором данных сложной структуры Pinery.

Подробнее о генераторе Pinery говорится в [6]. Данный инструмент предназначен для практического использования подхода к автоматической генерации тестовых данных сложной структуры, основанного на формальном описании тестовых данных в виде грамматик. В статье предложен метод разработки подобных генераторов, который подразумевает генерацию небольших объемов данных для тестирования заданных функций программного обеспечения (ПО), а также позволяет повторно использовать отдельные модули за счет обеспечения модульности настройки генератора.

Изучение современного состояния исследований по выбранной теме показало, что в настоящее время предпочтение отдается генерации небольшого количества тестовых данных. Такой подход должен обеспечивать экономию временных затрат на процесс тестирования и повышение его качества. В большинстве существующих инструментов отсутствует подобная оптимизация. Важной проблемой также является получение данных сложной структуры, которые обеспечивали бы полное покрытие функционала тестируемой системы.

1.1.3 Актуальность разработки системы автозаполнения баз данных

Актуальность разработки системы автоматизированного заполнения баз данных обусловлена тем, что аналогичных инструментов существует не так много, и в большинстве из них отсутствует возможность управления генерацией данных, в том числе данных сложной структуры. В рамках разрабатываемой системы под данными сложной структуры понимаются логически непротиворечивые данные.

Система автоматизированного заполнения баз данных актуальна при поддержке этапов реализации, начальной загрузки данных и тестирования в ЖЦ приложений БД.

На этапе реализации происходит загрузка международных, общероссийских, отраслевых справочников и классификаторов.

На этапе начальной загрузки данных в новую систему переносятся данные из предыдущих систем или резервных копий.

На этапе тестирования загружаются реальные тестовые данные, которые не сохраняются в БД после проведения проверок.

Внедрение разработанной системы позволит более тонко настраивать процесс генерации, облегчить миграцию данных из предыдущих версий БД, что обеспечит экономию времени и повысит качество тестирования. В дальнейшем к системе будет возможно подключать необходимые СУБД и тем самым расширять ее применение.

1.2 Сравнительный анализ современных средств автоматизированного заполнения баз данных

1.2.1 Цели и задачи сравнительного анализа

Сравнительный анализ современных средств автоматизированной генерации данных позволит определить наиболее используемые системы, а

также получить представление о том, насколько они распространены и актуальны.

Можно выделить следующие цели проведения анализа систем:

- выявление достоинств;
- выявление недостатков;
- выявление характерных базовых функций.

По итогам анализа главной задачей будет определение обязательных функций, а также уникальных возможностей, которые расширят функционал реализуемой системы и приведут к ее высокой конкурентоспособности.

1.2.2 Выбор средств автоматизированной генерации данных

На сегодняшний день существует множество инструментальных средств, которые позволяют выполнять заполнение БД тестовыми данными. Наиболее известными системами данного класса являются: Databene Benerator, DATATECT, DTM DATA GENERATOR, EMS DATA GENERATOR, RUBY-PLSQL-SPEC, SELITE, DBeaver, Red Gate SQL Data Generator, HTDGen, DBFORGE STUDIO, DBMonster, Turbo Data, AGENDA.

Как правило, эти средства ориентированы на генерацию тестовых данных. Для сравнительного анализа были выбраны наиболее востребованные системы:

- Databene Benerator от компании Volker Bergmann;
- DTM DATA GENERATOR от производителя DTM soft;
- EMS DATA GENERATOR от компании EMS;
- DBeaver от группы разработчиков Open-source community;
- Red Gate SQL Data Generator от компании Red Gate Software Ltd;
- DBFORGE STUDIO от компании Devart.

Среди представленных средств встречаются как бесплатные, так и платные, у которых возможно наличие on-line версий. Эта информация подробно представлена в таблице 1.

Таблица 1 – Общая информация об анализируемых системах

Система	Доступность	Наличие онлайн-версии	Фирма- производитель
Databene Benerator	Бесплатная	+	Volker Bergmann
DTM DATA GENERATOR	Платная, есть бесплатные версии	+	DTM soft
EMS DATA GENERATOR	Платная	+	EMS
DBeaver	Бесплатная	-	Open- source community
Red Gate SQL Data Generator	Платная. Демо-версия на две недели	+	Red Gate Software Ltd
DBFORGE STUDIO	Платная. Демо-версия на 30 дней, присутствует версия с неполным функционалом без временного ограничения	+	Devart

Начальный этап анализа позволил получить общие сведения о системах автоматизированной генерации данных.

1.2.3 Обзор средств для автоматизированной генерации данных

Databene Benerator – это платформа для создания реалистичных тестовых данных большого объема, используемых для тестирования нагрузки и производительности, а также для настройки витрины. Данные генерируются из легко конфигурируемой модели метаданных и экспортируются в базы данных, XML, CSV или плоские файлы, не имеющие внутреннюю иерархию [7]. На рисунке 2 показан интерфейс этого приложения.

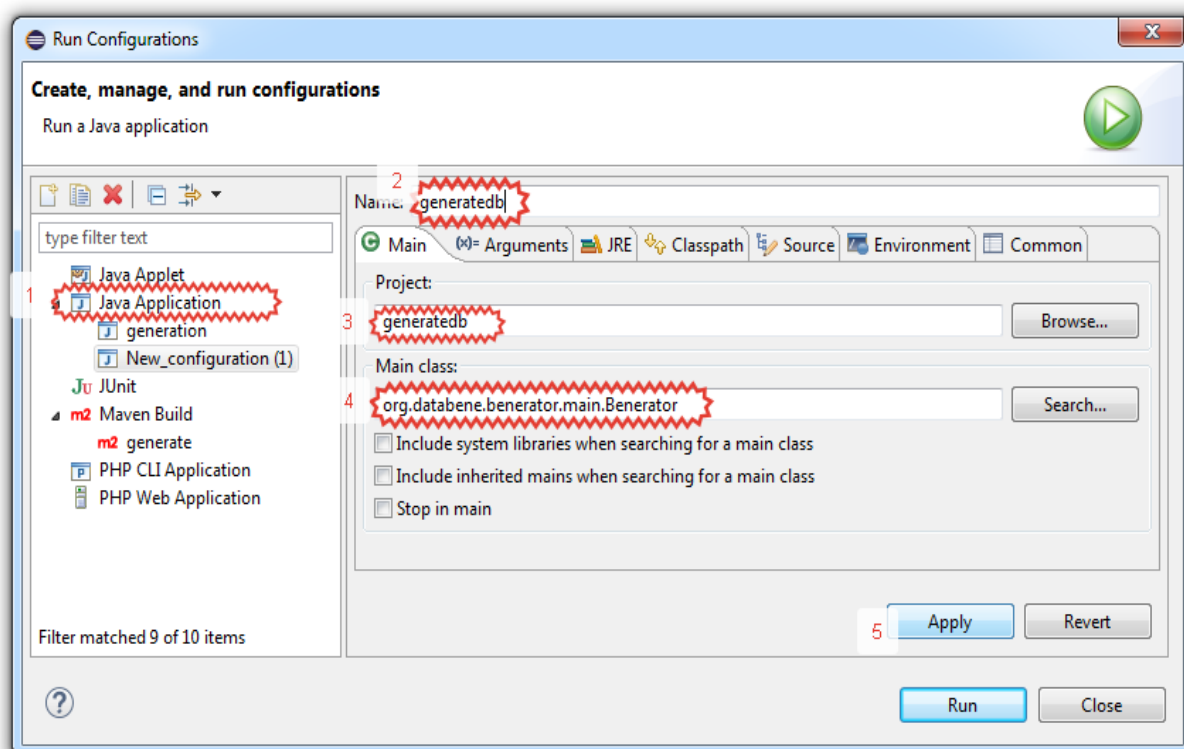


Рисунок 2 – Интерфейс Databene Venerator

EMS Data Generator – это программа для генерации тестовых данных для таблиц БД с возможностью сохранения и редактирования сценариев. С помощью этого инструмента возможно моделировать среду рабочей базы данных, заполнять одновременно несколько таблиц тестовыми данными, задавать таблицы для генерации данных, устанавливать диапазоны значений, создавать текстовые поля по маске, задавать списки значений вручную или выбирать их из запросов SQL, настраивать различные параметры генерации тестовых данных для каждого типа поля и многое другое. Эта программа предоставляет все необходимые инструменты для быстрой генерации тестовых данных и дополнительно включает консольную утилиту для генерации тестовых данных с помощью шаблонов генерации [8]. EMS Data Generator работает с несколькими СУБД, на рисунке 3 представлена первая страница интерфейса приложения для SQL Server, организованного в виде мастера.

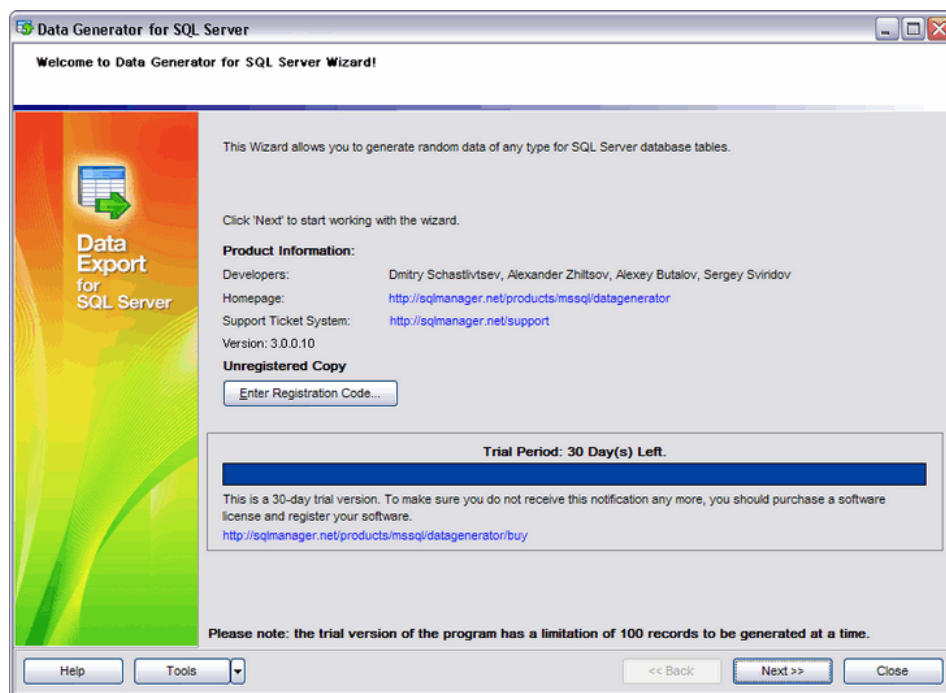


Рисунок 3 – Интерфейс EMS Data Generator for SQL Server

DTM Data Generator – это программный продукт, который создает строки данных и объекты схемы для целей тестирования: тестирование базы данных, анализ производительности, тестирование QA или выполнение тестов загрузки. Генератор был разработан для предоставления разработчикам и инженерам по обеспечению качества высококачественных и реалистичных тестовых массивов. Он автоматически создает значения данных и необязательные объекты схемы (таблицы, представления, процедуры, триггеры) [9]. На рисунке 4 представлен интерфейс системы.

Rule	Columns	Rows	SQL Prolog Script	SQL Epilog Script	Output	Preview
Sample Generated Data, top 50 rows						
CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Reg
N'MJNSW	N'Stryker Corp.'	N'Toney Mezzanotte'	NULL	N'64 Iron Mountain Road'	N'Biysk'	N'W
N'PLGNJ	N'Walgreen Co.'	N'Ellsworth Dejoode'	NULL	N'2313 Dundyan Gate'	N'Dumfries'	NUL
N'WVMQT	N'Stanley Works'	N'Tanisha Agudelo'	NULL	N'447 Easdale Path'	N'William'	N'Ei
N'WXXYC	N'American Standard Cos. Inc.'	N'Annabelle Hagins'	NULL	N'1582 Atkinson Drive'	N'Madrid'	N'N
N'LBGKZ	N'Hershey Foods Corp.'	N'Shon Demosthenes'	NULL	N'860 Parkway Drive'	N'Garland'	N'W
N'KSYKB	N'Anthem Inc.'	N'Melisa Pettibone'	NULL	NULL	NULL	NUL
N'ZZTSF	N'XL Capital'	N'Katy Yopp'	NULL	N'71 Arlene Drive'	NULL	NUL
N'QVMTF	N'Kroger Co.'	N'Erlene Krockick'	NULL	N'165 Kirk Street'	NULL	N'W
N'MFCRG	N'Federated Investors Inc.'	N'Isreal Millstead'	NULL	N'3 Horatius Street'	N'Streatham'	NUL
N'CUADV	N'Amerada Hess Corp.'	N'Liza Hewey'	N'M.D.'	N'4 Shady Birch Place'	N'Margate'	N'W
N'WACPX	N'Altera Corp.'	N'Dell Draa'	NULL	N'87 Cairnhill Avenue'	N'Pernik'	N'S
N'PDKRK	N'General Dynamics Corp.'	N'Michal Deliberato'	NULL	N'1995 Wintergreen Street'	N'Stamford'	N'W
N'SCHTO	N'American Financial Group Inc.'	N'Georgene Bajek'	NULL	N'454 Culmuir View'	N'Cherepovets'	NUL
N'BFUUF	N'Unocal Corp.'	N'Toney Janow'	NULL	NULL	N'Ben Guerir'	N'S
N'MCFTR	N'Ameren Corporation'	N'Stanton Iacobucci'	NULL	N'189 West 15th Terrace'	N'Nataf'	N'S
N'PVRYX	N'Block H&R'	N'Noreen Colvard'	NULL	N'60 Ashland Drive'	N'Lowestoft'	N'Ei

Рисунок 4 – Интерфейс DTM Data Generator

DBeaver – бесплатный мультиплатформенный инструмент работы с БД для разработчиков, администраторов баз данных, аналитиков и всех людей, которым необходимо работать с базами данных. Поддерживает все популярные базы данных: MySQL, PostgreSQL, SQLite, Oracle, DB2, SQL Server, Sybase, MS Access, Teradata и другие. DBeaver написан на фреймворке с открытым исходным кодом, что позволяет дописывать различные расширения [10]. Основной интерфейс системы приложения представлен на рисунке 5.

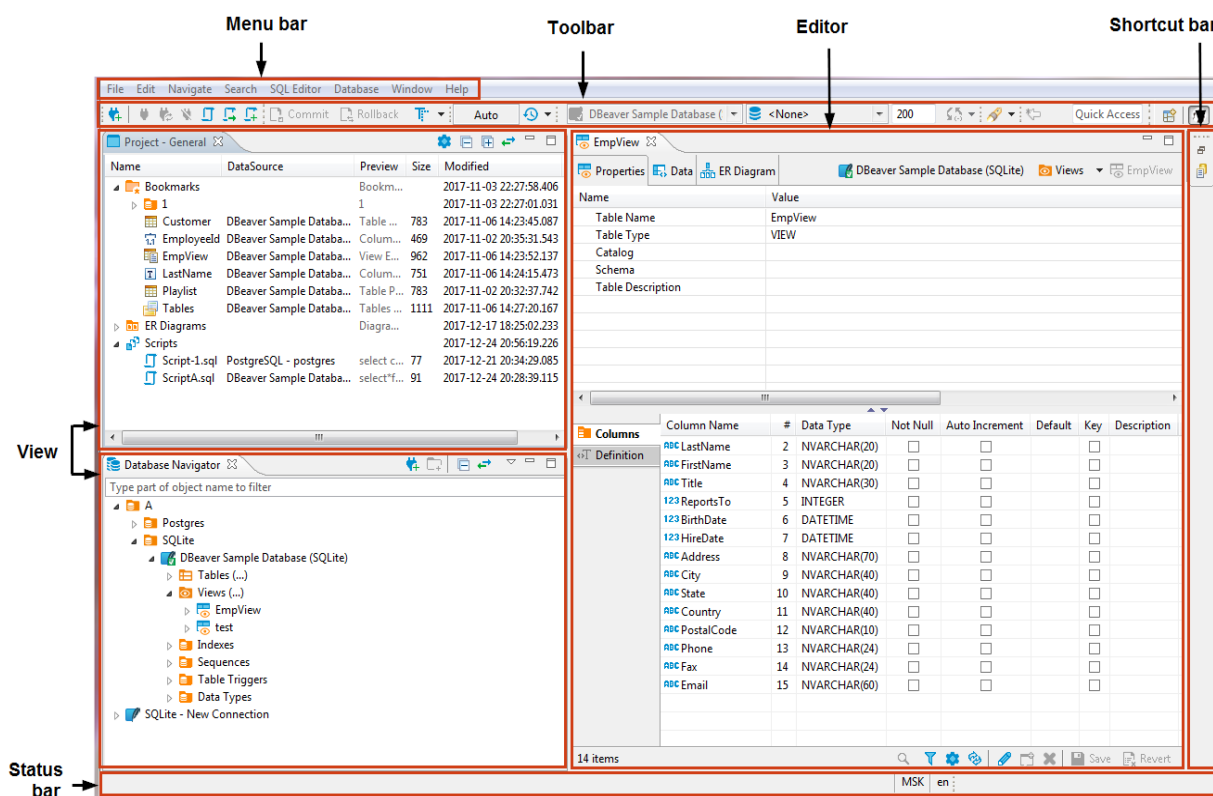


Рисунок 5 – Интерфейс DBeaver

Red Gate SQL Data Generator – программа, позволяющая генерировать тестовые данные для таблиц баз данных SQL Server. Использование приложения дает возможность моделировать среду рабочей базы данных, заполнять одновременно несколько таблиц SQL Server тестовыми данными, задавать формат таблиц и настраивать параметры генерации данных SQL Server для каждого типа поля. Red Gate SQL Data Generator предоставляет необходимые инструменты для быстрой и качественной генерации тестовых данных, настройки диапазонов и списков значений. Программа

предусматривает автоматическое заполнение таблиц и возможность редактирования накопленных баз данных, позволяет не перегружать базу данных дополнительной информацией [11]. Интерфейс данного средства представлен на рисунке 6.

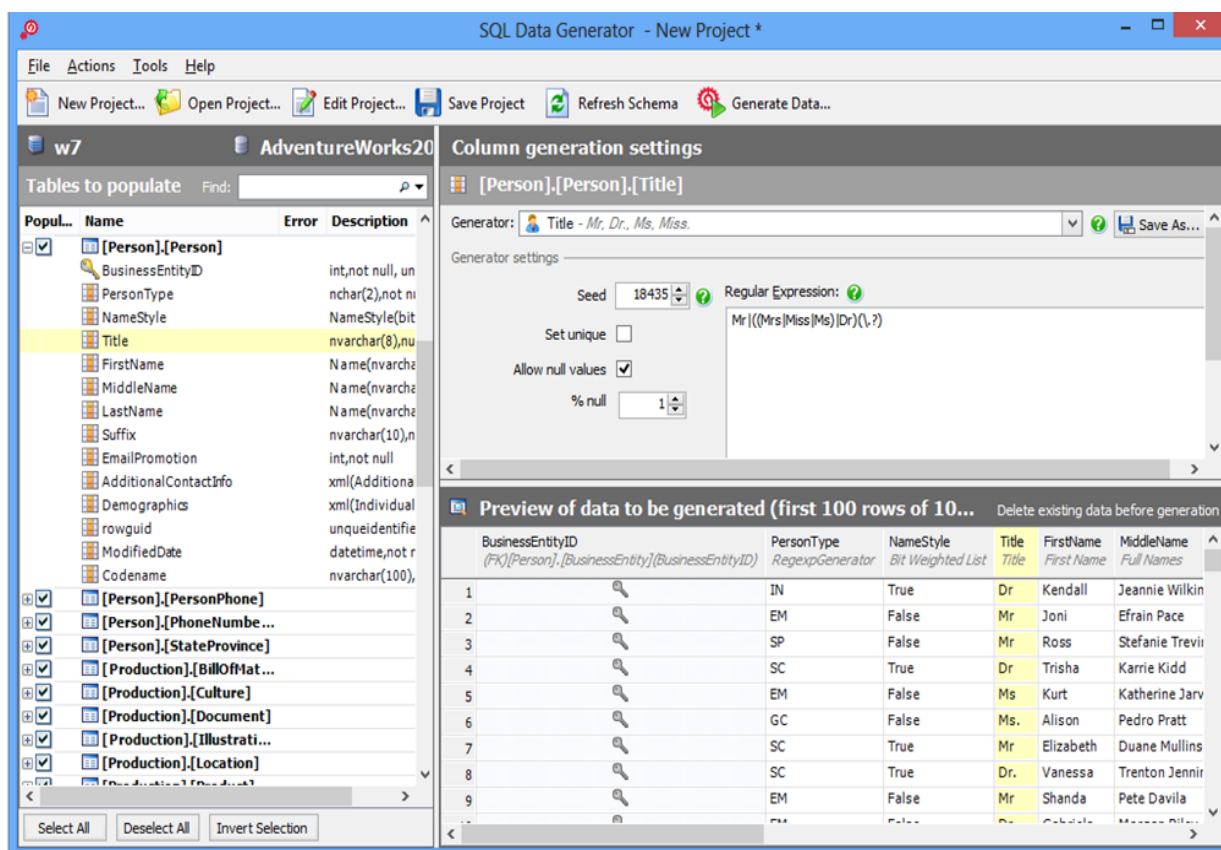


Рисунок 6 – Интерфейс Red Gate SQL Data Generator

DBFORGE STUDIO – это полно функциональная среда разработки для управления, администрирования, представления и анализа данных. Этот инструмент будет полезен для разработчиков и администраторов баз данных при выполнении сложных задач, таких как проектирование баз данных, написание SQL кода, сравнение баз данных, синхронизация схем и данных, генерации реальных тестовых данных и т.п. У производителя DBFORGE STUDIO есть несколько отдельных приложений для разных систем управления базами данных [12]. На рисунке 7 представлен интерфейс генерации данных в DBFORGE STUDIO для SQL Server.

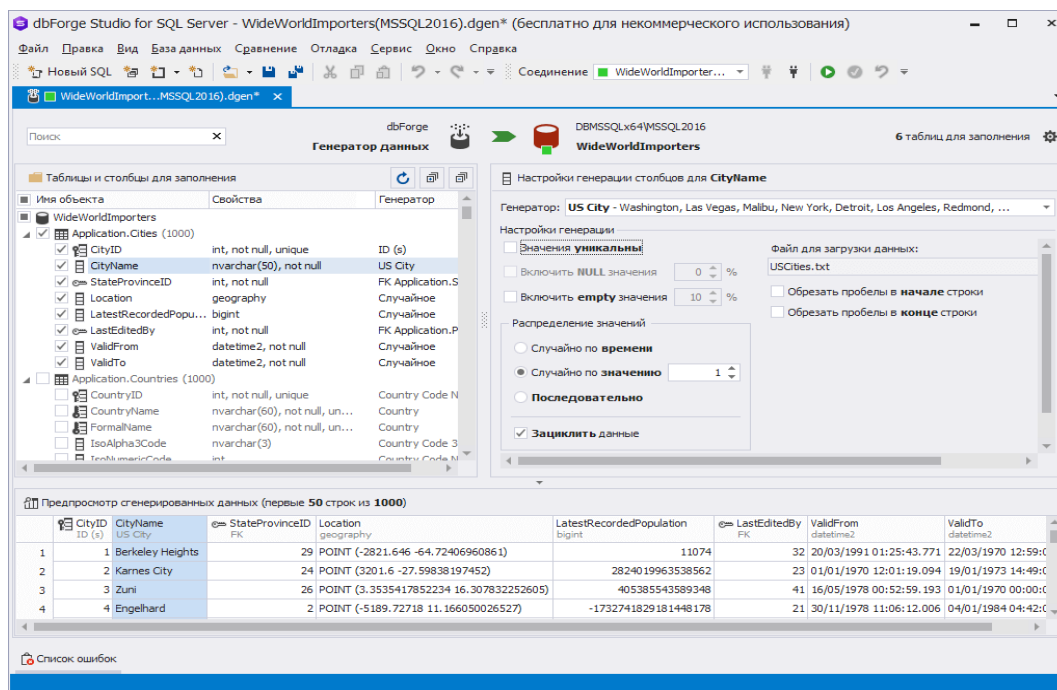


Рисунок 7 – Интерфейс DBFORGE STUDIO for SQL Server

Для оценки, анализа и сравнения инструментальных средств были выделены следующие критерии:

- загрузка из справочников – наличие функции загрузки тестовых данных из встроенных библиотек или подключение дополнительных справочников;
- импорт – возможность загрузки данных из сторонних БД;
- экспорт – возможность выгрузки в сторонние БД;
- поддерживаемые БД – перечень БД, взаимодействие с которыми поддерживает система;
- платформа – поддерживаемые операционные системы;
- форматы файлов – допустимые форматы файлов;
- анализ структуры БД – поддержка связи между таблицами и ссылочной целостности;
- предварительный просмотр – наличие функции вывода промежуточных результатов.

Результаты анализа выделенных инструментальных средств автоматической генерации данных представлены в таблице 2.

Таблица 2 – Результирующая таблица анализа систем по критериям

Функции	Системы					
	Databene Benerator	DTM DATA GENERATOR	EMS DATA GENERATOR	DBeaver	Red Gate SQL Data Generator	DBFORGE STUDIO
1	2	3	4	5	6	7
Загрузка из справочников	+	+	-	+	+	+
Импорт	+	-	-	+	+	+
Экспорт	+	-	+	+	-	+
Поддерживаемые СУБД	Oracle, DB2, MS SQL Server, MySQL, PostgreSQL, HSQL, H2, Derby, Firebird	MS SQL Server, Oracle, DB2, MySQL, PostgreSQL, Informix, Sybase, SQLite and Interbase/Firebird	MySQL, Oracle, Db2, Microsoft SQL Server, PostgreSQL, Interbase/ Firebird	MySQL, Oracle, PostgreSQL, IBM DB2, Microsoft SQL Server, Sybase, ODBC, Java DB (Derby), Firebird (Interbase), HSQLDB, SQLite, Mimer, IBM Informix, SAP MAX DB, Cache, Linter, Teradata, Vertica, MongoDB, Cassandra, Любой JDBC совместимый источник	Microsoft SQL Server	MySQL, Oracle, PostgreSQL, Microsoft SQL Server

Продолжение таблицы 2

1	2	3	4	5	6	7
Платформа	Linux, Windows и Mac OS	Имеет 32- и 64-разрядные версии и поддерживает все современные версии Windows : XP, Vista, Windows 7, Windows 8/10 (для настольных ПК), Windows Server 2003 и новее. Unix / Linux, Mac OS	Многоплатформенность	Linux, Windows и MacOS, Solaris, AIX, HP-UX	Windows	Linux, Windows и MacOS
Форматы файлов	Сценарий SQL, текст / CSV, XML, JSON .	CSV / текстовые файлы, электронные таблицы Excel, документы XML, файлы JSON, файлы Access, веб-ресурсы, пользовательские сценарии и т. д.	Текстовые файлы, электронные таблицы Excel, документы XML, веб-ресурсы, пользовательские сценарии, CSV и т. д.	Текстовые файлы, электронные таблицы Excel, документы XML, файлы JSON, веб-ресурсы, пользовательские сценарии и т. д.	CSV, электронные таблицы Excel	Текстовые файлы, электронные таблицы Excel, документы XML, файлы JSON, веб-ресурсы, пользовательские сценарии и т. д.
Анализ структуры БД	+	+	+	+	+	+
Предварительный просмотр	-	+	+	-	-	-

По представленной результирующей таблице можно судить о сильных и слабых сторонах каждой рассматриваемой системы.

1.2.4 Результаты сравнительного анализа

В результате анализа было выявлено, что функция загрузки из справочников присутствует во всех системах, кроме EMS DATA GENERATOR. Импорт из сторонних БД обеспечивают Databene Benerator, DBeaver, Red Gate SQL Data Generator и DBFORGE STUDIO. Системы DTM DATA GENERATOR и Red Gate SQL Data Generator не предоставляют возможности экспорта в сторонние БД. Все рассмотренные средства поддерживают достаточное количество различных БД, и только Red Gate SQL Data Generator ориентирована исключительно на Microsoft SQL Server. Каждая из выделенных систем совместима со всеми основными ОС, за исключением Red Gate SQL Data Generator, которая работает только на Windows. Доступна работа с наиболее популярными форматами файлов. Функция анализа структуры БД осуществляется во всех представленных средствах. Системы DTM DATA GENERATOR и EMS DATA GENERATOR имеют дополнительную опцию предварительного просмотра промежуточных результатов для удобства работы.

Таким образом, среди платных систем наиболее полным функционалом обладает DBFORGE STUDIO от Devart, но использование свободно распространяемых систем обеспечит существенную экономию.

Среди бесплатных систем Databene Benerator от Volker Bergmann и DBeaver от Open-source community сложно отдать приоритет какой-либо. Обе имеют одинаковый рассматриваемый функционал, за исключением списка поддерживаемых баз данных, который больше у DBeaver. Дальнейший выбор инструментальных средств генерации зависит от удобства интерфейса и предпочтений конкретного пользователя [1].

В большинстве рассмотренных средств автоматизированной генерации данных присутствуют стандартные функции, которые необходимо поддержать и в реализуемой системе. К ним относятся:

- загрузка из справочников;
- поддержка различных СУБД;
- работа с множеством форматов файлов;
- анализ структуры БД.

Кроме обязательных функций важна поддержка ряда дополнительных возможностей, обеспечивающих уникальность и конкурентоспособность системы. В качестве таких особенностей были определены следующие функции:

- расширение числа поддерживаемых СУБД;
- задание и модификация схемы БД в процессе наполнения;
- подключение новых справочников и классификаторов в процессе функционирования системы;
- резервное копирование и восстановление данных.

Указанные дополнительные функции планируется включить в разрабатываемую систему для поддержки этапов реализации, начальной загрузки данных и тестирования в ЖЦ приложений БД.

1.3 Постановка задачи на разработку

1.3.1 Постановка задачи на разработку системы автозаполнения баз данных

1.3.1.1 Цели разработки и назначение системы автозаполнения баз данных

Целью разработки является создание системы для автоматизированного заполнения баз данных с помощью генерации больших объемов случайных данных.

Система предназначена для быстрого подключения к нескольким СУБД с возможностью импорта и экспорта данных, генерации случайных данных, загрузки данных из других версий БД, а также для заполнения баз данных информацией из международных, общероссийских, отраслевых справочников и классификаторов.

1.3.1.2 Пользователи системы автозаполнения баз данных

Пользователей системы автоматизированного заполнения БД можно разделить на две категории: внутренние и конечные. К внутренним относятся пользователи, которые разрабатывают систему и поддерживают ее функционирование. Конечные пользователи используют систему непосредственно по назначению.

В рамках реализуемой системы внутренними пользователями являются:

- разработчик программных средств для взаимодействия с СУБД – специалист, который разрабатывает динамически подключаемые библиотеки для взаимодействия с СУБД по мере расширения их состава;
- разработчик приложения пользователя – специалист, разрабатывающий и развивающий приложение в процессе развития функционала системы;
- тестировщик – специалист по выявлению несоответствий фактического и ожидаемого результатов работы системы.

К конечным пользователям относятся:

- администратор БД – это специалист, который руководит всеми работами по созданию БД, принимает потребности пользователей, отвечает за эксплуатацию БД и обеспечивает ее работоспособность;
- прикладной программист – это разработчик программ для решения прикладных задач, который реализует запросы к базе данных;

- системный программист – специалист, который осуществляет контроль за функционированием СУБД в процессе работы, а также разрабатывает программы, расширяющие возможности СУБД.
- тестировщик БД – специалист, задачей которого является выявление сбоев и ошибок в работе БД;
- тестировщик ИС – специалист по выявлению несоответствий фактического и ожидаемого результатов работы системы в целом.

1.3.1.3 Автоматизируемые функции системы автозаполнения баз данных

Система автозаполнения баз данных должна автоматизировать следующие функции:

- генерация больших объемов случайных данных;
- генерация данных по заранее заданным правилам
- наполнение БД сгенерированными данными;
- перемещение данных между разными БД;
- перенос в БД информации из справочников и классификаторов;
- создание и изменение схемы базы данных.

С помощью интерфейса система задаёт правила генерации данных, соответствие этих правил атрибутам таблицы БД и контролирует процесс заполнения БД в процессе генерации.

1.3.1.4 Аппаратное и программное окружение системы автозаполнения баз данных

Для работы системы автозаполнения БД на ПК пользователя необходимы следующие минимальные требования к программно-аппаратным средствам:

- 1) операционная система: Microsoft Windows 7, 8.1, 10;

2) наличие необходимой СУБД: Microsoft SQL Server (2012 или выше), PostgreSQL (9.0.1 или выше); MySQL(5.7.11 или выше); Oracle(10g XE или выше);

3) параметры компьютера:

- процессор 32-разрядный или 64-разрядный с тактовой частотой 2,5 ГГц или выше;
- оперативная память не менее 4 ГБ;
- свободное место не менее 20 ГБ на жестком диске;
- графическое устройство DirectX 9 с драйвером WDDM 1.0 или более поздней версии.

Необходимым параметром является возможность подключения к серверу с помощью внешней сети.

1.3.2 Постановка задачи на разработку программных средств взаимодействия с СУБД для системы автозаполнения баз данных

1.3.2.1 Цели разработки и назначение программных средств взаимодействия с СУБД для системы автозаполнения баз данных

Целью разработки является создание программ-посредников, которые будут обеспечивать приложение пользователя всем необходимым функционалом для взаимодействия с конкретной СУБД.

ПС взаимодействия с СУБД предназначены для:

- доступа к различным СУБД;
- взаимодействия с приложением пользователя;
- доступа к таблицам и атрибутам БД;
- доступа к метаданным БД;
- наполнения БД сгенерированными данными или данными из файлов;
- генерации случайных данных.

Программные средства будут реализованы с помощью архитектуры plug-in, которая будет основана на динамически подключаемых библиотеках DLL [13]. Это и является ключевой особенностью разрабатываемой системы, которая гарантирует ее расширяемость и гибкость.

1.3.2.2 Основные функции программных средств взаимодействия с СУБД для системы автозаполнения баз данных

Основными группами функций ПС взаимодействия с СУБД для системы автоматического заполнения баз данных являются:

- взаимодействие с серверами БД;
- работа с БД;
- работа с таблицами;
- работа со справочниками и классификаторами;
- генерация данных случайным образом.

При взаимодействии с сервером должны выполняться операции получения списка доступных серверов и управления соединением с выбранным сервером.

Работа с БД подразумевает получение списка баз данных выбранного сервера, установку и управление соединением с выбранной базой данных.

В работу с таблицами входит взаимодействие со структурой и содержимым таблиц БД.

Работа со справочниками и классификаторами включает загрузка данных файла справочника в базу данных, при этом возможно изменение структуры БД.

Генерация данных случайным образом должна управляться приложением пользователя и осуществлять заполнение таблиц БД.

1.3.2.3 Требования к программному интерфейсу

Программный интерфейс – это интерфейс, позволяющий двум независимым компонентам программного обеспечения обмениваться

информацией, а также выступающий в роли посредника между внутренними и внешними программными функциями [14].

Разрабатываемые программные средства взаимодействия с СУБД осуществляют связь между приложением пользователя и базами данных. Доступ к базам данных обеспечивают динамически подключаемые к СУБД библиотеки.

Программные средства должны быть ориентированы на выполнение методов доступа к данным и метаданным БД. Чтобы в дальнейшем можно было подключать к системе новые серверы БД, а также для обеспечения понятности и простоты поддержки системы, ПС должны быть построены на основе универсального шаблона.

Шаблон проектирования в разработке программного обеспечения – это повторяемая конструкция в архитектуре, которая представляет собой решение проблемы проектирования в рамках некоторого часто возникающего контекста [15]. Существует множество шаблонов проектирования, но общий результат их применения – структурированный программный продукт.

Выделяют три вида шаблонов: порождающие, структурные и поведенческие. Порождающие шаблоны направлены на создание экземпляра объекта. Структурные шаблоны связаны с композицией объектов и показывают. Поведенческие шаблоны определяют общие закономерности связей между объектами, облегчая их коммуникацию.

Преимущества использования шаблонов:

- снижение количества ошибок в связи с унификацией модулей и элементов проекта;
- возможность создания более модифицируемого и гибкого программного обеспечения;
- повышение читаемости кода и понимание архитектуры проекта;
- повышение эффективности труда отдельных исполнителей и всей группы в целом.

2 Разработка требований к программным средствам взаимодействия с СУБД для системы автозаполнения баз данных

2.1 Функциональные требования

Программные средства взаимодействия с СУБД должны осуществлять функции, объединенные в следующие группы:

- взаимодействие с сервером;
- работа с БД;
- работа с таблицами;
- работа со справочниками и классификаторами;
- генерация данных случайным образом.

Далее рассмотрим составляющие каждой группы. Взаимодействие с сервером включает операции:

- получения списка доступных серверов заданной СУБД;
- установки соединения с выбранным сервером;
- закрытия соединения с сервером.

Работа с БД подразумевает выполнение следующих действий:

- получение списка баз данных выбранного сервера;
- установка соединения с выбранной базой данных;
- закрытие соединения с базой данных.

В работу с таблицами входят следующие функции:

- получение списка таблиц выбранной базы данных;
- получение информации о столбцах заданной таблицы;
- создание новой таблицы;
- добавление данных в таблицу;
- удаление содержимого таблицы.

Операции работы со справочниками и классификаторами включают:

- загрузку данных файла в таблицу БД;
- создание справочной таблицы;

- удаление справочной таблицы.

Генерация данных случайным образом будет осуществляться под управлением приложения пользователя. Эта функция включает в себя саму генерацию данных и помещение полученного результата в заданную таблицу БД.

Общий функционал и поведение ПС взаимодействия с СУБД представляется с помощью диаграммы вариантов использования UML [16]. Функциональные возможности ПС взаимодействия с СУБД показаны на рисунке 8.

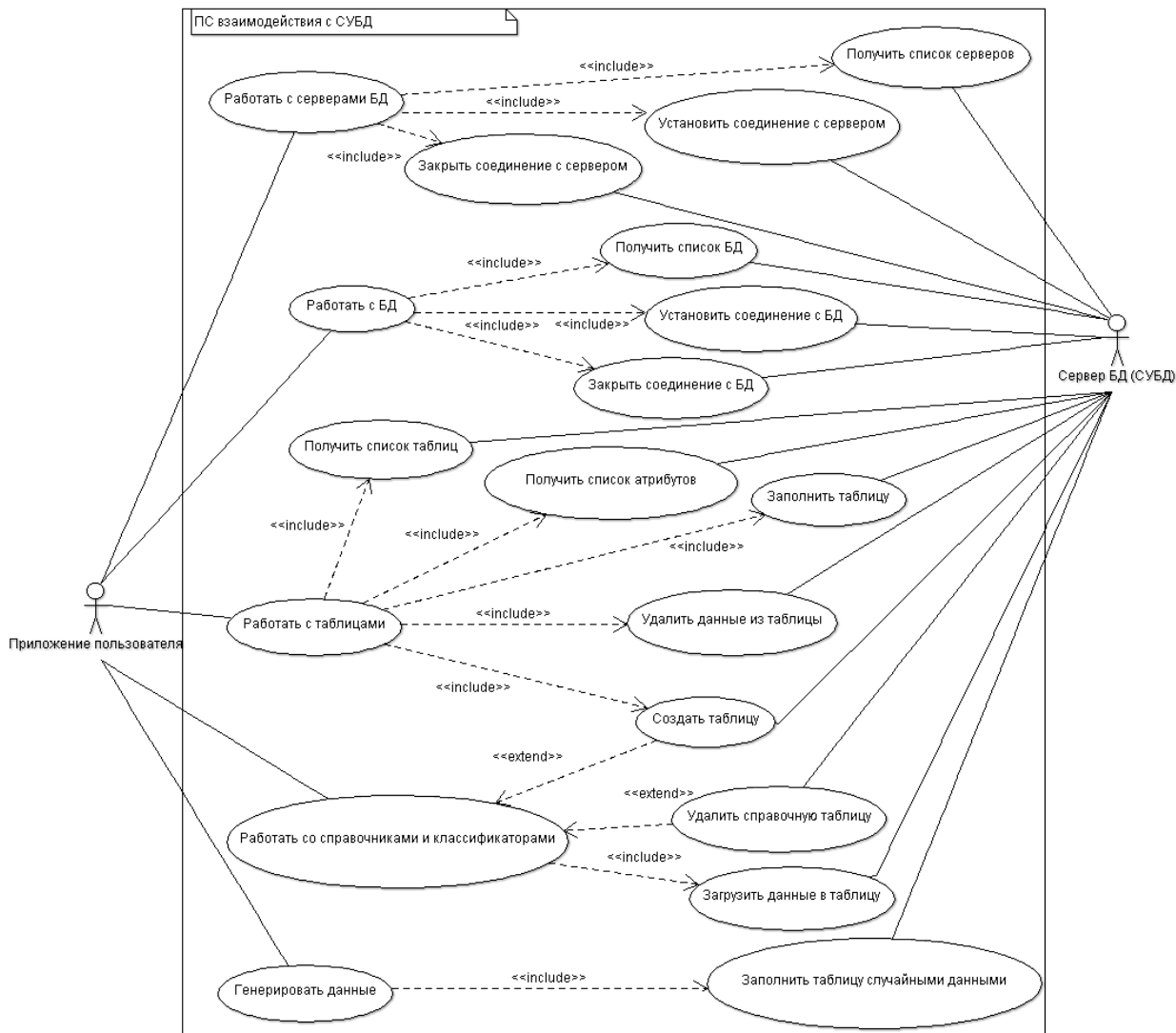


Рисунок 8 – Диаграмма вариантов использования ПС взаимодействия с СУБД

В таблице 3 приведено краткое описание вариантов использования.

Таблица 3 – Краткое описание вариантов использования

Цель	Краткое описание
Работать с серверами БД	Приложение пользователя может получить список серверов, подключиться или отключиться от выбранного сервера
Работать с БД	Приложение пользователя может получить список БД заданного сервера, подключиться или отключиться от выбранной БД
Работать с таблицами	Приложение пользователя может получить список таблиц, их атрибутов, наполнить таблицу данными или удалить ее содержимое, а также создать новую таблицу в БД
Работать со справочниками и классификаторами	Приложение пользователя может наполнить таблицу данными из справочника и, при необходимости, выполнить создание или удаление справочной таблицы
Генерировать данные	ПС взаимодействия с СУБД выполняют генерацию случайных данных по правилам, заданным приложением пользователя, и заполняют этими данными БД

Полное описание процесса генерации случайных данных приведено в таблице 4.

Таблица 4 – Полное описание прецедента «Генерировать данные»

Название варианта использования	Генерировать данные
Участники и интересы	Пользователь хочет заполнить таблицу БД случайными данными
Основные действующие лица	Приложение пользователя
Второстепенные действующие лица	Сервер БД (СУБД)
Предусловие	Пользователь заходит в приложение системы автозаполнения БД
Постусловие	Пользователь успешно заполнил таблицу БД случайными данными
Основной сценарий	<ol style="list-style-type: none"> 1. Приложение запросило список доступных серверов 2. Плагин получил список серверов и вернул его приложению 3. Приложение запросило список БД выбранного сервера 4. Плагин получил список БД сервера и вернул его приложению 5. Приложение запросило список таблиц выбранной БД сервера 6. Плагин получил список таблиц БД сервера и вернул его приложению 7. Приложение запросило список атрибутов выбранной таблицы 8. Плагин получил список атрибутов и вернул его приложению
	<ol style="list-style-type: none"> 9. Приложение вызвало генерацию данных с указанными пользователем параметрами 10. Плагин сгенерировал данные согласно полученным правилам 11. Плагин заполнил таблицу БД сгенерированными данными
Альтернативные потоки	<ol style="list-style-type: none"> 1) таблица БД уже заполнена; <ol style="list-style-type: none"> а) сгенерированные данные заносятся в конец таблицы; б) сгенерированные данные заносятся в таблицу с удалением имеющихся данных.

Были определены функциональные требования к ПС взаимодействия с СУБД, построена диаграмма вариантов использования и описаны основные прецеденты.

2.2 Требования к программной архитектуре

2.2.1 Требования к обобщенной архитектуре системы

Для реализации средства автоматизированного заполнения баз данных была выбрана клиент-серверная архитектура [17].

Клиент-сервер – это архитектура, в которой участвуют две стороны: клиент – заказчик услуг и сервер – поставщик услуг. Клиент отправляет запрос на сервер с целью выполнения задач или получения информации, сервер в свою очередь обрабатывает полученный запрос и возвращает клиенту результат.

Среди преимуществ данной архитектуры можно выделить:

- разделение программного кода клиентского и серверного приложений;
- низкие требования к клиентским ЭВМ;
- основная часть вычислительных операций выполняется на сервере;
- клиентское приложение не требует администрирования;
- разделение задач разработки и, следовательно, более быстрое ее выполнение;
- гибкость и масштабируемость;
- высокая степень защиты локальной сети.

В системе автоматизированного заполнения БД роль клиента играет приложение пользователя. Задачами этого уровня являются генерация случайных данных, сбор и отправка на второй уровень введенных пользователем данных, формирование пользовательского запроса и вывод на экран пользователя информации в удобном виде. На стороне клиента содержатся все инструменты для взаимодействия с БД: создания таблиц, добавления и удаления записей.

В качестве сервера выступает СУБД, которая предоставляет доступ к таблицам и метаданным БД. СУБД принимает запрос от приложения пользователя и выполняет необходимые действия с данными, после чего возвращает результат.

Детально архитектура разрабатываемой системы автоматизированного заполнения БД показана в виде диаграммы развертывания на рисунке 9.

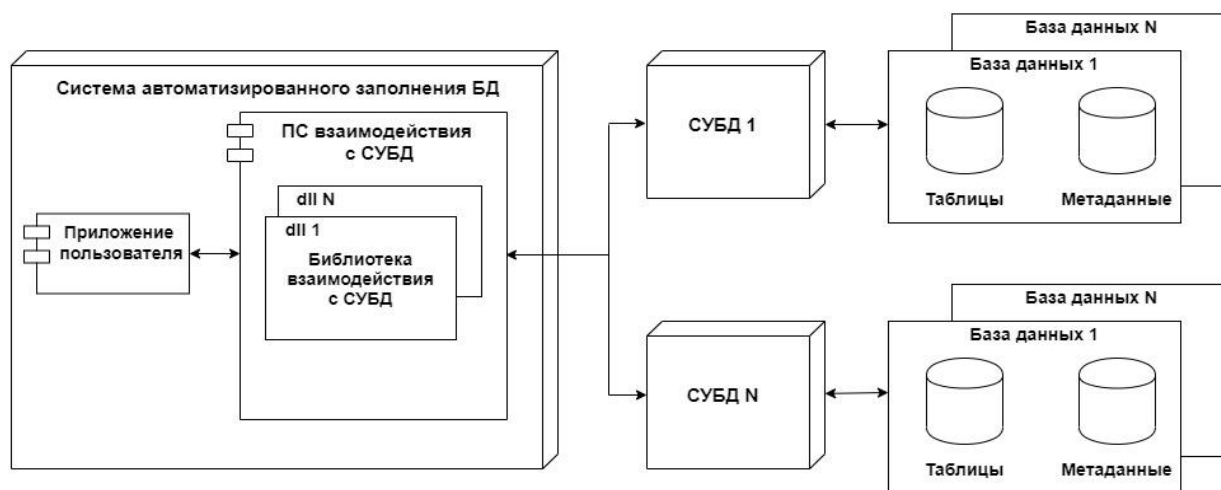


Рисунок 9 – Диаграмма развертывания системы автоматизированного заполнения БД

На диаграмме представлена система автозаполнения БД, которая состоит из приложения пользователя и программных средств взаимодействия с СУБД, включающих набор динамически подключаемых библиотек для взаимодействия с конкретным классом СУБД. Каждая СУБД имеет доступ данным и метаданным нескольких БД. Число доступных СУБД будет увеличиваться по мере подключения к приложению пользователя новых dll-библиотек. Такая архитектура обеспечит быстрое расширение возможностей системы без необходимости постоянно изменять приложение пользователя.

2.2.2 Требования к архитектуре программных средств взаимодействия с СУБД для системы автозаполнения баз данных

Для реализации программных средств взаимодействия с СУБД в системе автоматизированного заполнения БД рассматривались следующие варианты архитектуры:

- монолитное приложение в составе пользовательского приложения;
- монолитное приложение в виде единой dll-библиотеки;
- совокупность dll-библиотек в виде plug-in модулей.

Монолитное приложение – это программа, объединяющая интерфейс пользователя и код доступа к данным. Такой вариант прост в разработке, но не приспособлен к дальнейшему развитию.

Монолитное приложение в виде динамически подключаемой библиотеки – это программа, взаимодействующая с приложением пользователя и подключающая его к тем СУБД, доступ к которым в ней описан. При использовании такого подхода подключение к новой СУБД будет невозможным без перекомпиляции основного приложения.

В качестве архитектуры ПС взаимодействия с СУБД была выбрана совокупность dll-библиотек в виде plug-in модулей. Этот вариант имеет ряд достоинств:

- экономное использование ресурсов;
- удобство изменения и обновления dll-библиотек;
- простота развертывания;
- повторное использование модулей.

Плагин – это программный модуль, который предназначен для расширения базовых возможностей основной программы. Данные модули представляют собой подпрограммы, которые предоставляют свои функции загрузившему их приложению, но сами не являются исполняемыми [18].

Для реализации поддержки выбранной архитектуры необходимо описать интерфейс, на основе которого будут созданы плагин-модули. Каждый плагин содержит код подключения к некоторой СУБД и находится в файле с расширением dll.

Взаимодействие плагина с приложением пользователя будет осуществляться динамически, что приведет к экономии ресурсов компьютера при выполнении программы. Вызов dll должен осуществляться явно, при выполнении определенного оператора программы.

Чтобы осуществлять взаимодействие между БД и приложением на C#, необходим посредник, в качестве которого была выбрана технология ADO.NET [19]. Это наиболее развитая технология доступа к данным, основанная на платформе .NET Framework от компании Microsoft, которая развивает принципы, заложенные в технологиях ADO и DAO, и является более мощной, универсальной и простой в применении.

ADO.NET позволяет работать с отсоединенными источниками данных. Они представляют собой структуры, организующие данные в оперативной памяти компьютера. Это обеспечивает автономную работу, что является необходимым при разработке масштабируемого приложения.

ADO.NET содержит набор классов, через которые можно отправлять запросы к БД, устанавливать подключения, получать ответ от БД и выполнять ряд других операций. ADO.NET осуществляет взаимодействие с БД определенного типа с помощью встроенных провайдеров. К ним относятся:

- провайдер для MS SQL Server;
- провайдер для Oracle;
- провайдер для ODBC;
- провайдер для OLE DB;
- провайдер для сервера SQL Server Compact 4.0
- провайдер EntityClient.

Наличие данных провайдеров обеспечивает гибкость и расширяемость технологии ADO.NET и делает ее наиболее подходящей для построения программных средств взаимодействия с СУБД для системы автозаполнения БД. Каждый провайдер реализует одинаковые базовые классы: Connection, Command, DataAdapter, DataReader, Parameter, Transaction. Их имена зависят от провайдера.

Функционал технологии ADO.NET предоставляет унифицированный интерфейс для работы с различными СУБД. Все провайдеры используют одинаковые интерфейсы и функции, в связи с этим код доступа к данным

выглядит практически идентично независимо от провайдера. Разработчик может создать обобщенный код доступа к данным с помощью интерфейсов. В дальнейшем процесс преобразования кода для доступа к данным конкретной БД будет быстрым и удобным.

В качестве общих требований к программной архитектуре можно рассматривать эвристики, предложенные Рилем [20]:

- два класса должны быть либо независимыми, либо один из классов должен зависеть только от открытого интерфейса другого класса;
- атрибуты и связанные методы должны находиться в одном классе;
- класс должен охватывать только одну абстракцию, а информация, не относящаяся к данной абстракции, когда подмножество методов работает на соответствующем подмножестве атрибутов, должна быть перенесена в другой класс;
- системная логика должна быть распределена по возможности равномерно.

Часто нарушается эвристика о нахождении атрибутов и связанных методов в одном классе. Это происходит из-за классов, обладающих большим количеством методов открытия доступа (get, set), определенных в их открытом интерфейсе.

2.3 Требования к аппаратно-программному окружению программных средств взаимодействия с СУБД для системы автозаполнения баз данных

Для разработки ПС взаимодействия с СУБД для системы автозаполнения БД необходимо определить минимальные и рекомендуемые требования к программно-аппаратным средствам.

В таблице 5 представлены минимальная и рекомендуемая конфигурации ПО для разработки ПС взаимодействия с СУБД.

Таблица 5 – Минимальная и рекомендуемая конфигурации ПО для разработки ПС взаимодействия с СУБД

Программное обеспечение	Минимальная конфигурация	Рекомендуемая конфигурация
Операционная система	Microsoft Windows 7, 8.1	Microsoft Windows 10
Среда разработки	Microsoft Visual Studio v. 14.0	Microsoft Visual Studio v. 16.0
Фреймворк	Microsoft Framework .NET v.4.6	Microsoft Framework .NET v.4.8
СУБД	Microsoft SQL Server 2012 / PostgreSQL 9.0.1 / MySQL 5.7.11 / Oracle 10g XE	Microsoft SQL Server 2019 / PostgreSQL 10 / MySQL 5.7.24 / Oracle 18c

Минимальная и рекомендуемая конфигурации аппаратного обеспечения для разработки ПС взаимодействия с СУБД представлены в таблице 6.

Таблица 6 – Минимальная и рекомендуемая конфигурации аппаратного обеспечения для разработки ПС взаимодействия с СУБД

Параметр аппаратного обеспечения	Минимальная конфигурация	Рекомендуемая конфигурация
Процессор	32-разрядный или 64-разрядный, 2.5 ГГц	64-разрядный, 5 ГГц
Оперативная память	4 Гб	8 Гб и больше
Свободное пространство жесткого диска	20 Гб	100 Гб и больше
Графическое устройство	DirectX 9 с драйвером WDDM 1.0	DirectX 11.3 с драйвером WDDM 2.0

Сервер должен допускать подключение к нему с помощью внешней сети.

2.4 Обоснование выбора поддерживаемых СУБД

Разрабатываемая система автоматизированного заполнения БД направлена на работу с СУБД, которые отвечают следующим общим требованиям: поддержка реляционной модели, надежность, эффективность, популярность.

В 2019 году компания Gartner выпустила отчет по очередному исследованию в области систем управления базами данных [21]. Результаты исследования показали, какие СУБД являются наиболее эффективными и востребованными для реализации систем оперативного учета. На рисунке 10 представлен так называемый магический квадрант, на котором распределены ИТ-вендоры, поставляющие СУБД. Лидирующую позицию занимает компания Microsoft. В области лидеров также присутствуют Oracle, Amazon Web Services, Google и SAP. В области лидеров также присутствуют Oracle, Amazon Web Services, Google и SAP. В области лидеров также присутствуют Oracle, Amazon Web Services, Google и SAP.

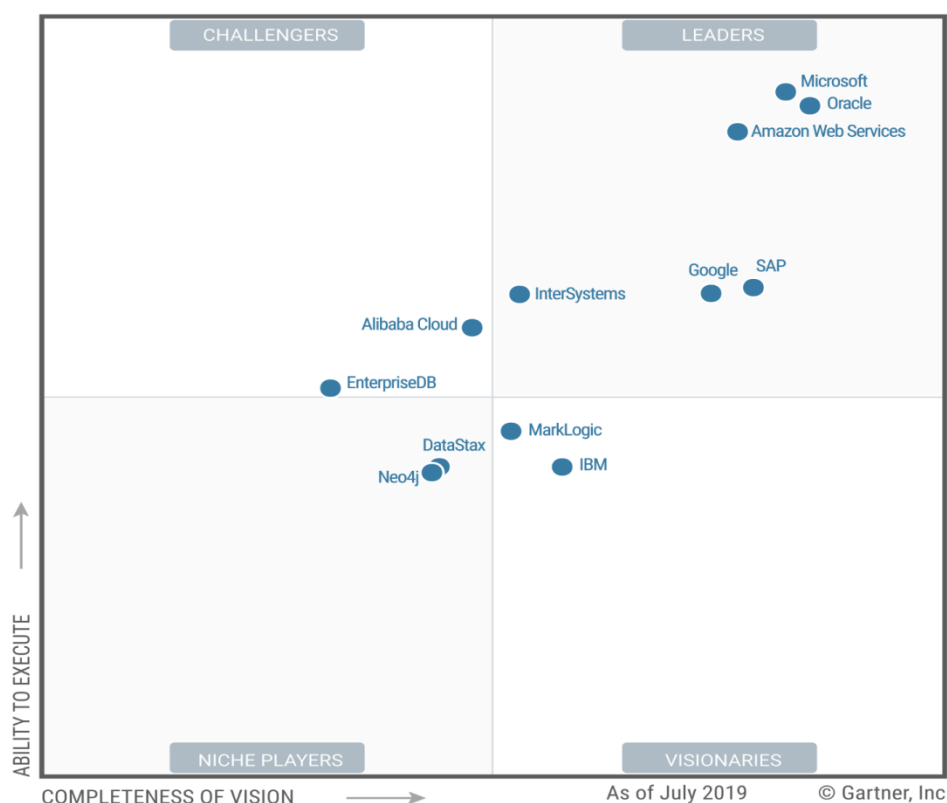


Рисунок 10 – Магический квадрант для операционных СУБД по версии Gartner

В системе предполагается использование следующих популярных СУБД:

- MS SQL Server;
- Oracle;
- PostgreSQL;
- MySQL.

В таблице 7 приведены некоторые характеристики выделенных СУБД.

Таблица 7 – Характеристики современных СУБД

Характеристика	MS SQL Server	Oracle	PostgreSQL	MySQL
Интерфейс	GUI, SQL, другое	GUI, SQL	GUI, SQL	SQL
Поддерживаемые языки	Python, Ruby, Java, .Net, VB, PHP	C, C++, C#, Ruby, Java, Objective-C и другие	Java, C, C++, Delphi, .Net, Python, PHP, Perl, Tcl	C, C++, C#, Java, Ruby, Objective-C и другие
Методы доступа	ADO.NET, ODBC, JDBC, OLE DB, табличный поток данных (TDS)	Oracle (OCI), ODBC, JDBC, ODP.NET	ADO.NET, ODBC, JDBC, потоковое API для больших объектов, нативная C библиотека	ADO.NET, API, ODBC, JDBC
Ограничения размера БД	524 272 ТБ	8 ЭБ	Нет	64 ТБ на таблицу

На текущий момент система поддерживает только MS SQL Server. В процессе развития будет возможен доступ к любым другим СУБД.

2.5 Обоснование выбора средств разработки программных средств взаимодействия с СУБД для системы автозаполнения баз данных

2.5.1 Обоснование выбора языковых средств разработки

При разработке программных средств взаимодействия с СУБД необходимо использовать язык программирования, с помощью которого будут созданы библиотеки взаимодействия с СУБД, и язык запросов к БД.

Язык программирования – формальная знаковая система, предназначенная для записи компьютерных программ. Язык программирования определяет набор лексических, синтаксических и семантических правил, задающих внешний вид программы и действия, которые выполнит исполнитель (компьютер) под её управлением [22].

На сегодняшний день рынок языковых средств разработки очень велик. Для разработки системы автозаполнения баз данных наиболее удобными являются объектно-ориентированные языки. Самые популярные на данный момент языки C++, C# и Java.

Языки C# и Java были разработаны в качестве потомков языка C++. Для выбора языковых средств разработки необходимо провести сравнительный анализ языков C# и Java.

C# – это язык программирования общего назначения, который впервые появился в 2000 году в рамках инициативы Microsoft .NET. Он был разработан для общей языковой инфраструктуры (CLI) – открытой спецификации, разработанной Microsoft и стандартизированной ISO и ECMA. Java, первоначально выпущенный Sun Microsystems в 1995 году, является языком программирования общего назначения, который был разработан с конкретной целью, позволяющей разработчикам “write once, run anywhere” – написать код единожды и запускать в любом месте. Java-приложения скомпилированы в байт-код, который может запускаться при реализации виртуальной машины Java (JVM). Подобно CLI, JVM помогает преодолеть разрыв между исходным и машинным кодом [23].

Далее перечислены некоторые сходные преимущества языков:

- оба языка компилируются в машинно-независимый и независимый от языка программирования программный код, запускаемый в рамках управляемой среды выполнения (для Java – Java Virtual Machine, для C# – Common Language Runtime);
- автоматический сбор мусора;

- весь код помещается в пакеты, благодаря чему отсутствуют заголовочные файлы;
- все элементы должны принадлежать какому-то классу, нельзя объявлять глобальные функции и константы.

К различиям можно отнести то, что Java имеет более сложную структуру, в связи с этим для изучения языка и сопровождения уже готовых систем требуются более длительные сроки, чем при работе с C#. Но главным различием между языками является полная интеграция C# со стандартом Component Object Model (COM). Разработчики могут наследовать компоненты, созданные с помощью любого языка программирования, который поддерживается .NET. Эти компоненты в дальнейшем могут сами выступить в роли базовых для программ на других языках. Код на промежуточном языке MSIL не интерпретируется, а компилируется в машинный код с помощью JIT-компилятора [24]. Вследствие этого, приложения, созданные с помощью C#, выполняются быстрее, чем те, что написаны на Java, именно поэтому для разработки системы автоматизированного заполнения БД был выбран язык программирования C#.

Для взаимодействия с базами данных необходим язык запросов. В настоящее время существует несколько таких языков, например: SQL, DDL, XPath, XQuery, Language Integrated Query.

В системе автозаполнения БД будет использоваться язык SQL [25], который является мощным инструментом для работы с данными.

Выбор языка SQL обоснован следующими преимуществами:

- обеспечение программного доступа к БД;
- реляционная основа;
- возможность переноса с одной вычислительной системы на другую;
- наличие стандартов;
- поддержка интерактивных запросов;

- отсутствие зависимости от конкретных СУБД;
- поддержка клиент-серверной архитектуры;
- динамическое изменение и расширение структуры БД;
- различное представление данных.

SQL – это язык структурированных запросов к БД, который основан на международном стандарте ANSI и поддерживается всеми реляционными СУБД. Однако каждая СУБД имеет свой диалект, расширяющий возможности данного языка. В таблице 8 представлены диалекты языка SQL, используемые в тех СУБД, поддержка которых запланирована в разрабатываемой системе.

Таблица 8 – Диалекты SQL в современных СУБД

СУБД	Краткое название диалекта	Полное название диалекта
MS SQL Server	T-SQL	Transact-SQL
PostgreSQL	PL / pgSQL	Procedural Language / PostGres Structured Query Language
Oracle	PL / SQL	Procedural Language / SQL
MySQL	SQL / PSM	SQL / Persistent Stored Module

Внутренний язык конкретной СУБД обеспечивает полноценное программирование и получение системной информации, а также упрощает запросы к БД. Разработчики СУБД постоянно модифицируют и расширяют диалекты, в результате чего они все больше отличаются друг от друга.

2.5.2 Обоснование выбора инструментальных средств разработки

Наиболее распространенной и удобной для написания кода на C# является интегрированная среда разработки Microsoft Visual Studio 2019 (версия 4.16). Данная среда обеспечивает полную поддержку в написании кода на протяжении всего цикла от разработки до внедрения. На

сегодняшний день Visual Studio одно из самых мощных средств разработанных компанией Microsoft. Для удобной работы на языке C# данная среда поддерживает архитектуру .NET Framework. Благодаря этой платформе, разработка на C# – это удобный и быстрый процесс. Главная особенность каркаса .NET – общезыковая среда исполнения Common Language Runtime (CLR), с помощью которой удобно вести разработку на разных языках [26].

Некоторые возможности среды:

- поддержка нескольких языков: C#, VB.NET, C++, F#, Delphi.NET;
- кроссплатформенность: разработка для платформ Windows, MacOS, Linux, Android, iOS, Tizen;
- широкая библиотека классов;
- возможность быстрого написания кода за счет легкого перехода к любому символу, типу, элементу или файлу;
- возможность просмотра истории, авторов, даты изменения в строке;
- анализ кода;
- подключение облачного хранения;
- пакет Entity Framework для удобного подключения к различным СУБД;
- системы Windows Presentation Foundation (WPF) и WinForms для взаимодействия с пользователем;
- удобный интерфейс для работы с тестами [27];
- множество возможностей для будущей Web-разработки.

Данный список включает основные преимущества разработки на Visual Studio .NET Framework. Сегодня платформа развивается и содержит множество надстроек для повышения эффективности разработки. Именно поэтому для создания системы автоматизированного заполнения БД была выбрана среда Microsoft Visual Studio.

3 Разработка программных средств взаимодействия с СУБД для системы автозаполнения баз данных

3.1 Разработка системы классов ПС взаимодействия с СУБД

Разрабатываемые программные средства взаимодействия с СУБД состоят из интерфейса, который является контрактом между плагинами и приложением пользователя, самих плагинов, обеспечивающих доступ к конкретной СУБД, а также набора классов, выступающих в роли определяемых разработчиком типов. На данный момент реализован прототип системы автозаполнения БД, имеющий в своем составе один плагин для взаимодействия с СУБД MS SQL Server. На рисунке 11 показана диаграмма классов ПС взаимодействия с СУБД.

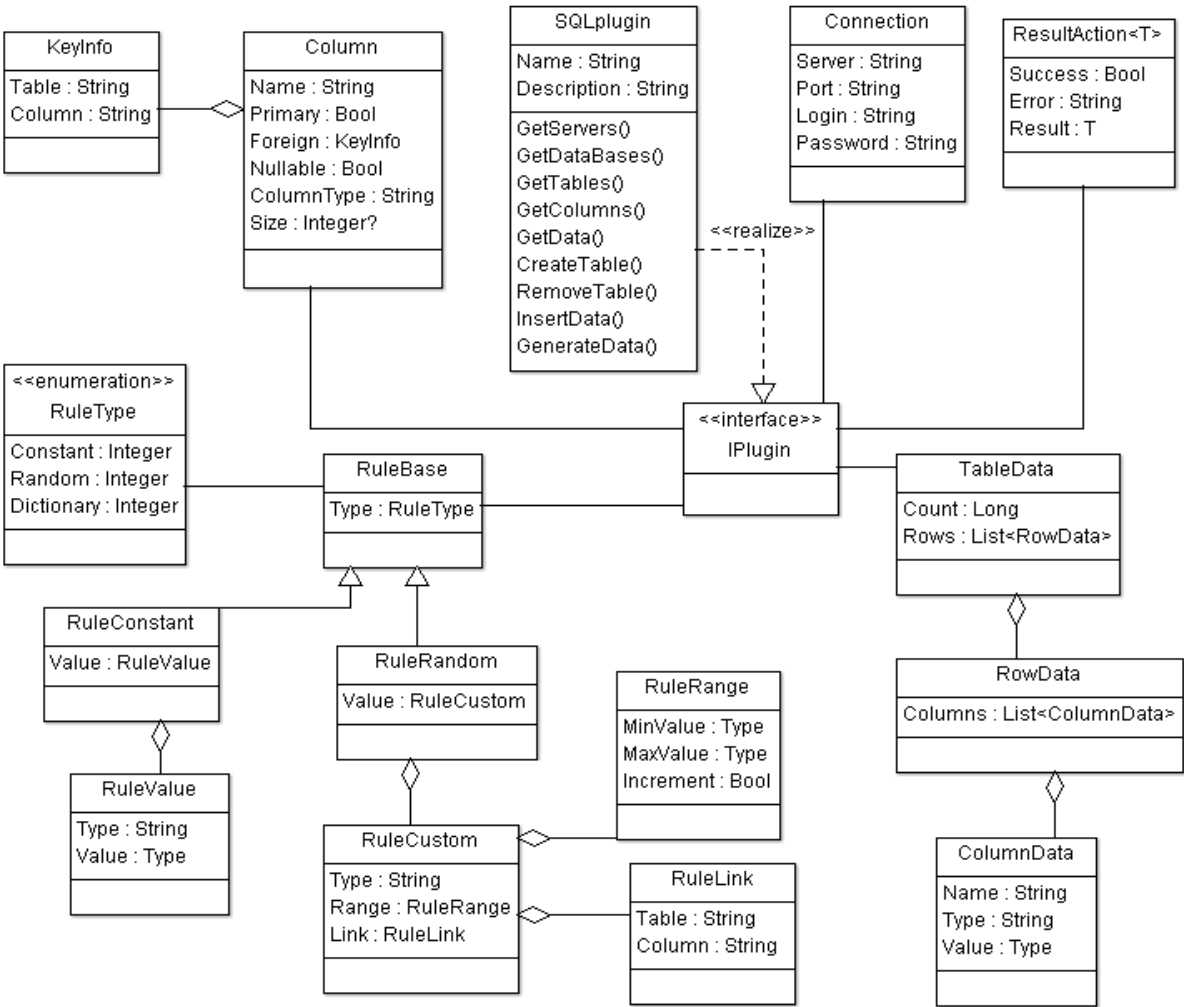


Рисунок 11 – Диаграмма классов ПС взаимодействия с СУБД

Классы, описывающие определяемые типы, помещены в единый проект и для удобства объединены в следующие модели представления данных:

- Column;
- Connection;
- ResultAction;
- Rules;
- TableData.

Модель Column представляет данные об атрибутах таблицы БД, она содержит класс Column, который хранит данные о столбцах таблиц и класс KeyInfo, который хранит данные внешних ключей. Спецификации соответствующих классов представлены в таблицах 9 и 10.

Таблица 9 – Спецификация класса Column

Свойство	Тип	Назначение
Name	String	Имя столбца
Primary	Bool	Первичный ключ
Foreign	KeyInfo	Данные о внешнем ключе
Nullable	Bool	Отсутствие значения
ColumnType	String	Тип значений столбца
Size	Integer?	Размерность

Таблица 10 – Спецификация класса KeyInfo

Свойство	Тип	Назначение
Table	String	Имя таблицы
Column	String	Имя столбца

Модель Connection содержит одноименный класс, который предназначен для хранения данных подключений к серверам БД. Его спецификация приведена в таблице 11.

Таблица 11 – Спецификация класса Connection

Свойство	Тип	Назначение
Server	String	Адрес сервера
Port	String	Номер порта
Login	String	Логин
Password	String	Пароль

Модель ResultAction представляет данные о результате выполнения какого-либо действия. Она включает класс ResultAction<T>, спецификация которого приведена в таблице 12.

Таблица 12 – Спецификация класса ResultAction<T>

Свойство	Тип	Назначение
Success	Bool	Успех выполнения
Error	String	Сообщение ошибки
Result	T	Результат выполнения

Модель TableData представляет данные таблицы и включает 3 класса: TableData, RowData и ColumnData. Класс TableData необходим для хранения данных о строках таблицы, класс RowData – для хранения списка столбцов таблицы, а класс ColumnData – для хранения данных о содержимом столбцов. Спецификации данных классов представлены в таблицах 13 – 15.

Таблица 13 – Спецификация класса TableData

Свойство	Тип	Назначение
Count	Long	Количество строк таблицы
Rows	List<RowData>	Список строк

Таблица 14 – Спецификация класса RowData

Свойство	Тип	Назначение
Columns	List<ColumnData>	Список значений столбцов

Таблица 15 – Спецификация класса ColumnData

Свойство	Тип	Назначение
Name	String	Имя столбца
Type	String	Тип значений столбца
Value	Type	Значение

Модель Rule представляет данные, характеризующие правила генерации и заполнения. Она включает 7 классов и 1 перечисление RuleType, содержащее режимы заполнения таблицы БД. В данной модели присутствует базовый класс RuleBase с наследниками RuleConstant и RuleRandom. В этих классах содержится информация о способе заполнения таблицы: константой или случайными данными соответственно. Класс RuleValue хранит данные о

константе для заполнения таблицы. Класс RuleCustom содержит информацию для генерации случайных данных. В классах RuleLink и RuleRange задаются связи и диапазон генерации соответственно. Спецификации классов модели Rule приведены в таблицах 16 – 22.

Таблица 16 – Спецификация класса RuleBase

Свойство	Тип	Назначение
Type	RuleType	Тип правила

Таблица 17 – Спецификация класса RuleConstant

Свойство	Тип	Назначение
Value	RuleValue	Данные константы

Таблица 18 – Спецификация класса RuleValue

Свойство	Тип	Назначение
Type	String	Тип значения
Value	Type	Значение

Таблица 19 – Спецификация класса RuleRandom

Свойство	Тип	Назначение
Value	RuleCusom	Данные генерации

Таблица 20 – Спецификация класса RuleCustom

Свойство	Тип	Назначение
Type	String	Тип значений
Range	RuleRange	Диапазон значений
Link	RuleLink	Связь

Таблица 21 – Спецификация класса RuleLink

Свойство	Тип	Назначение
Table	String	Имя таблицы
Column	String	Имя столбца

Таблица 22 – Спецификация класса RuleRange

Свойство	Тип	Назначение
MinValue	Type	Минимальное значение
MaxValue	Type	Максимальное значение
Increment	Bool	Инкрементное изменение данных

Код классов, описывающих определяемые типы, представлен в приложении А.

Класс SQLplugin реализует интерфейс IPlugin. Спецификация данного класса представлена в таблице 23.

Таблица 23 – Спецификация класса SQLplugin

Свойство	Тип	Назначение
Name	String	Название плагина
Description	String	Описание плагина

Коды интерфейса и плагина приведены в приложении Б и в приложении В соответственно.

3.2 Разработка методов взаимодействия с серверами БД

В разрабатываемой системе автозаполнения БД приложение пользователя не взаимодействует напрямую с серверами БД. Это осуществляет конкретный плагин в составе программных средств, который является посредником и, при подключении к приложению пользователя, предоставляет весь необходимый функционал для работы с базами данных.

Плагин доступа к СУБД MS SQL Server разработан в виде динамически подключаемой библиотеки и выполняет следующие команды, поступающие от приложения пользователя:

- получение списка локальных серверов БД;
- получение списка БД заданного сервера и ее таблиц;
- получение списка атрибутов и метаданных заданной таблицы;
- получение содержимого заданной таблицы;
- создание и удаление таблицы в БД;
- запись данных в таблицу;
- генерация случайных данных по заданным правилам.

С целью демонстрации взаимодействия плагина с серверами БД была построена изображенная на рисунке 12 диаграмма последовательности для задачи получения списка атрибутов и метаданных таблицы БД.

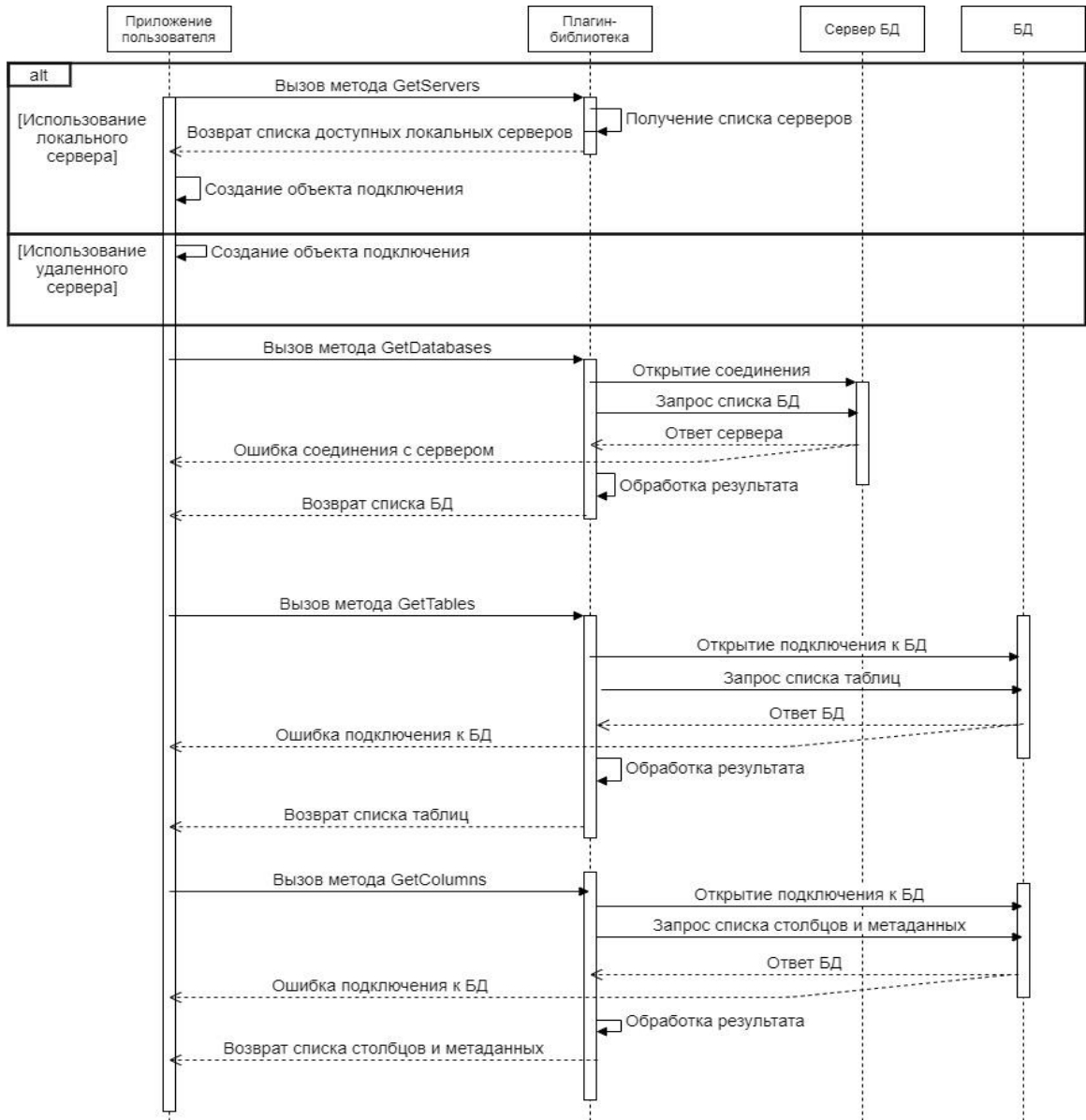


Рисунок 12 – Диаграмма последовательности для задачи получения списка атрибутов и метаданных таблицы

Приложение пользователя вызывает методы плагина, который в свою очередь соединяется с сервером и БД, получает от них необходимые данные и возвращает приложению результат в требуемом виде. Подключение к серверу или БД инициируется в каждом методе плагина и в случае ошибки приложение получит соответствующее сообщение.

Методы взаимодействия с серверами БД описаны в интерфейсе IPlugin и должны быть реализованы каждым плагином системы автозаполнения БД. Спецификация этих методов приведена в таблице 24.

Таблица 24 – Спецификация методов взаимодействия с серверами БД

Имя	Назначение	Параметры	Возвращаемое значение
1	2	3	4
GetServers	Получение списка локальных серверов	–	ResultAction<List<string>> – список имен серверов при успехе, иначе код ошибки и сообщение
GetDatabases	Получение списка БД заданного сервера	Connection connection – объект подключения к серверу	ResultAction<List<string>> – список имен БД при успехе, иначе код ошибки и сообщение
GetTables	Получение списка таблиц БД заданного сервера	Connection connection – объект подключения к серверу string database – имя базы данных	ResultAction<List<string>> – список имен таблиц при успехе, иначе код ошибки и сообщение
GetColumns	Получение информации о столбцах таблицы БД заданного сервера	Connection connection – объект подключения к серверу string database – имя базы данных string table – имя таблицы	ResultAction<List<Column>> – список объектов класса Column с метаданными при успехе, иначе код ошибки и сообщение
GetData	Получение содержимого таблицы БД	Connection connection – объект подключения к серверу string database – имя базы данных string table – имя таблицы	ResultAction<TableData> – объект класса TableData со строками таблицы при успехе, иначе код ошибки и сообщение
CreateTable	Создание новой таблицы	Connection connection – объект подключения string database – имя базы данных string table – таблица List<Column> columns – список объектов Column	bool
RemoveTable	Удаление таблицы	Connection connection – объект подключения к серверу; string database – имя базы данных; string table – имя таблицы	bool
InsertData	Запись данных в таблицу	Connection connection – объект подключения к серверу; string database – имя базы данных; string table – имя таблицы; TableData data – объект класса TableData со строками таблицы	bool

Продолжение таблицы 24

1	2	3	4
GenerateData	Генерация случайных данных по правилам	Connection connection – объект подключения к серверу string database – имя базы данных string table – имя таблицы int rows – количество строк Dictionary<string, RuleBase> rules – словарь объектов типа ключ-значение, где ключ – строка, а значение – правила генерации	bool

Все методы класса IPlugin имеют модификатор доступа public, что является необходимым при разработке приложения с поддержкой архитектуры плагинов.

3.3 Методика создания плагин-библиотеки для взаимодействия с новой СУБД

В прототипе разрабатываемой системы автозаполнения БД реализован один плагин взаимодействия с СУБД MS SQL Server. В дальнейшем возможно расширение числа поддерживаемых СУБД путем разработки новых плагинов.

Чтобы создать плагин для взаимодействия с другой СУБД необходимо выполнить следующие действия:

- 1) в среде MS Visual Studio создать проект библиотеки классов;
- 2) в новом проекте добавить ссылку на библиотеку с интерфейсом IPlugin;
- 3) в новом проекте создать класс-наследник интерфейса IPlugin;
- 4) в целях экономии времени можно продублировать в классе нового плагина свойства и методы класса уже имеющегося плагина;

- 5) внести корректировки в свойства названия и описания плагина, в реализацию методов и в тексты запросов, адаптируя программу под конкретную СУБД;
- 6) выполнить сборку проекта;
- 7) полученный файл с расширением .dll поместить в папку с плагинами приложения пользователя;
- 8) запустить приложение и проверить корректность работы нового плагина;
- 9) при необходимости внести поправки в код плагина, пересобрать проект и обновленный dll-файл поместить в папку с плагинами приложения пользователя.

4 Тестирование программных средств взаимодействия с СУБД для системы автозаполнения баз данных

4.1 Обоснование выбора метода тестирования

Тестирование – это процесс проверки соответствия между фактическим поведением программного продукта и его ожидаемым поведением с помощью набора определенных тестов. Тестирование ПО позволяет оценить аспекты его работы на основе наблюдений за его выполнением в специальных условиях, выявить ситуации, не соответствующие спецификациям, технически исследовать программный продукт и получить информацию о его качестве. Существует множество видов тестирования, которые классифицируются по ряду признаков.

Виды тестирования по целям включают функциональное, нефункциональное и виды тестирования, связанные с изменениями. Функциональное тестирование предназначено для проверки того, какие функции программного продукта реализованы и насколько корректно. Нефункциональное тестирование направлено на проверку требований, не относящихся к функциональности, и определение работоспособности системы при разных условиях.

По степени автоматизации тестирование бывает ручное, когда дополнительные программные средства не используются, и автоматизированное.

По сценарию тестирование может быть позитивным, когда ПО проверяют на соответствие ожидаемым результатам работы, и негативным, которое оценивает работоспособность ПО при условии поведения пользователя, отличного от ожидаемого.

По доступу к коду выделяют тестирование «белого ящика», тестирование «черного ящика» и тестирование «серого ящика».

В зависимости от уровня тестирование подразделяют на модульное, интеграционное и системное.

По признаку формальности существует тестирование по тестам, проводимое по заранее разработанным тест-кейсам, исследовательское, которое выполняется при одновременном написании и исполнении тестов, и свободное, основанное на опыте тестировщика.

По важности тестируемых функций выделяют дымовое тестирование, тестирование критического пути и расширенное. При дымовом тестировании проверяется самая важная функциональность ПО. Тестирование критического пути включает проверку функций, используемых обычными пользователями, расширенное тестирование направлено на проверку всей функциональности.

Для тестирования ПС взаимодействия с СУБД был выбран метод модульного тестирования. Данный метод используется для проверки на корректность отдельных блоков программного обеспечения. Цель модульного тестирования заключается в выявлении ошибок, локализованных в тестируемом компоненте ПО.

Преимущества модульного тестирования:

- возможность оценить функционал отдельного модуля и получить общее представление о его программном интерфейсе;
- выявление ошибок в начале цикла разработки;
- мотивация усовершенствования кода и его упрощения без изменения функциональности;
- возможность независимого тестирования частей проекта.

4.2 Модульное тестирование

Для проведения модульного тестирования в среде MS Visual Studio необходимо выполнить следующие шаги:

- 1) добавить в проект с плагином тестовый проект типа «Проект модульного теста»;
- 2) в тестовом проекте добавить ссылку на основной проект;
- 3) в тестовом решении создать методы тестирования;

4) выполнить тесты.

Рассмотрим подход модульного тестирования на примере метода получения списка баз данных сервера. В тестовом решении был создан представленный на листинге 1 метод для проверки корректности работы метода класса плагина `GetDatabases(Connection connection)`. Метод подключается к серверу, данные о котором передаются в виде объекта подключения, и возвращает список БД этого сервера.

Листинг 1 – Тестовый метод для проверки получения списка баз данных сервера

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using SqlPlugin;
using Project.Models;

namespace UnitTestProject1
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]

        public void TestMethod1()
        {
            var con = new Project.Models.Connection
            {
                Server = @"DESKTOP-48V0K0S/SQLEXPRESS",
                Port = "",
                Login = "",
                Password = ""
            };

            SqlServerPlugin plug = new SqlServerPlugin();

            List<string> expected = new List<string> { "master", "model", "msdb", "tempdb",
                "H_O_T_E_L", "HOTEL", "Hotels", "storage", "TestDB" };

            List<string> actual = (List<string>) plug.GetDatabases(@"DESKTOP-
48V0K0C\SQLEXPRESS", "", "");

            Assert.AreEqual(expected, actual);
        }
    }
}
```

Результат выполнения теста находится в обозревателе тестов, который показан на рисунке 13.

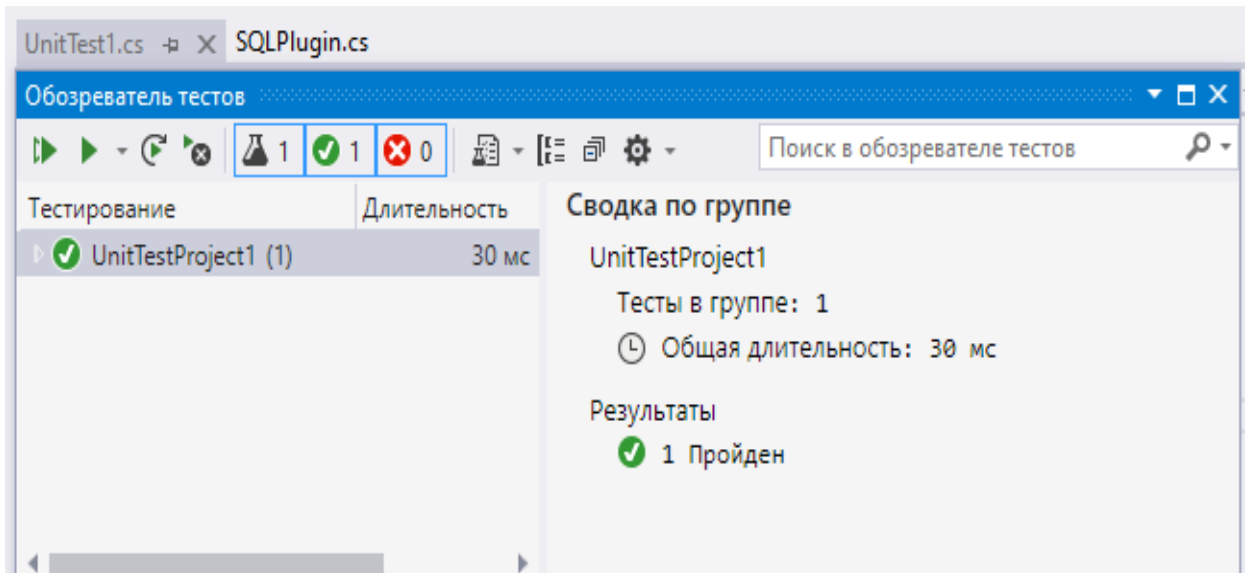


Рисунок 13 – Результат выполнения модульного теста

Созданный тест обработан успешно, следовательно, метод получения списка БД работает корректно.

4.3 Расчет метрик программного обеспечения

Метрика ПО – это мера, позволяющая получить количественный показатель свойства программного обеспечения, используемый для оценки качества исходного кода программы, определения сложности разработанного ПО и затрат на разработку. В данной работе рассмотрен расчет метрики Холстеда.

В основе данной метрики 4 параметра, значения которых необходимо вычислить для проведения расчетов:

- число уникальных операторов программы;
- число уникальных операндов программы;
- общее число операторов в программе;
- общее число операндов в программе.

Рассчитаем метрику Холстеда на примере варианта использования «Получить список серверов».

Операнды и операторы приведены в таблице 25.

Таблица 25 – Операторы и операнды

Операторы	Число вхождений	Операнды	Число вхождений
=	6	servers_list	4
!=	1	result	1
.	6	dt	2
()	10	row	3
<>	6	name	2
[]	2	instance	5
+	2	e	2
new	2		
in	1		
if	1		
as	2		
add	1		
try	1		
catch	1		
return	2		
foreach	1		
n1 = 16	N1 = 45	n2 = 7	N2 = 19

На основе полученных параметров необходимо определить словарь программы и длину программы по формулам (1) и (2):

$$n = n1 + n2 = 23, \quad (1)$$

$$N = N1 + N2 = 64. \quad (2)$$

На основе полученных результатов можно рассчитать показатели объема кода (3) и уровня качества программирования (4):

$$V = N \times \log_2 n \quad (3)$$

$$L' = \frac{(2 \times n2)}{(n1 \times N2)} \quad (4)$$

В результате подстановки получены следующие значения: $V = 289,536$; $L' = 0,05$.

Далее можно определить трудоемкость программы на основе полученных значений объема кода и уровня качества программирования (5):

$$E_c = \frac{V}{L'} \times 2. \quad (5)$$

Получено следующее значение: $E_c = 5790,72$.

Проведенное вычисление показывает, что разработанный код метода получения списка серверов обладает средней сложностью поддержки кода.

5 Планирование процесса разработки и расчет бюджета

Планирование процесса разработки включает формирование плана выполнения работ, который отражает порядок действий при реализации проекта с учетом необходимых ресурсов. Основной задачей планирования является определение целей и методов их достижения.

Для демонстрации плана разработки подходит диаграмма Ганта, отображающая длительность и сроки выполнения этапов разработки на шкале времени, а также требуемые для их реализации ресурсы. В качестве инструмента управления проектом была выбрана программа Microsoft Project. Дальнейший расчет будет производиться для ПС взаимодействия с СУБД с одним плагином в составе.

В таблице 26 приведены результаты определения задач проекта, их тип, длительность их выполнения и задачи, предшествующие им.

Таблица 26 – Задачи проекта

№	Задача	Тип	Длительность	Предшественники
1	2	3	4	5
1	Начало проекта	Веха	0	–
2	Анализ предметной области и постановка задачи на разработку	Фаза	9,5	–
3	Анализ актуальности разработки	Задача	4	1
4	Сравнительный анализ систем-аналогов	Задача	6	1
5	Постановка задачи на разработку системы	Задача	1,5	3;4
6	Постановка задачи на разработку ПС взаимодействия с СУБД	Задача	2	5
7	Анализ предметной области и постановка задачи на разработку завершены	Веха	0	6
8	Разработка требований к системе	Фаза	10	2

Продолжение таблицы 26

1	2	3	4	5
9	Разработка требований к архитектуре системы	Задача	2	7
10	Разработка требований к ПС взаимодействия с СУБД	Фаза	8	2
11	Функциональные требования	Задача	4	9
12	Требования к архитектуре ПС	Задача	3	11
13	Выбор поддерживаемых СУБД	Задача	1	12
14	Выбор средств разработки системы	Задача	1	12
15	Разработка требований к системе завершена	Веха	0	13;14
16	Разработка ПС взаимодействия с СУБД	Фаза	35	8;10
17	Разработка классов ПС	Задача	7	15
18	Реализация методов	Задача	28	17
19	Разработка системы завершена	Веха	0	18
20	Тестирование ПС взаимодействия с СУБД	Фаза	5	16
21	Выбор методов тестирования	Задача	1	19
22	Проведение тестирования	Задача	4	21
23	Тестирование ПС взаимодействия с СУБД завершено	Веха	0	22
24	Интеграция с приложением пользователя	Задача	0,5	20;23
25	Тестирование собранной системы	Задача	3	24
26	Завершение работы над проектом	Фаза	16	25
27	Оформление документации	Задача	14	25
28	Сдача проекта	Задача	2	27
29	Проект завершен	Веха	0	28

После составления плана выполнения работ необходимо оценить ресурсы проекта и распределить их по задачам. Информация об имеющихся в проекте ресурсах и их распределение представлены в таблицах 27 и 28.

Таблица 27 – Ресурсы проекта

Название ресурса	Тип ресурса	Затраты		Начисление
		Стандартная ставка	Ставка сверхурочных	
Программист ПС	Трудовой	180,00 Р/ч	300,00 Р/ч	Пропорциональное
Программист приложения	Трудовой	180,00 Р/ч	300,00 Р/ч	Пропорциональное
Тестировщик	Трудовой	120,00 Р/ч	200,00 Р/ч	Пропорциональное
Аналитик	Трудовой	80,00 Р/ч	150,00 Р/ч	Пропорциональное
Компьютер	Материальный	25 000,00 Р		Пропорциональное
Интернет	Затраты			Пропорциональное
Печать	Затраты			Пропорциональное

Таблица 28 – Распределение ресурсов по задачам

Название задачи	Ресурсы	Единицы (затраты)
1	2	3
Начало проекта	Компьютер	1
Анализ предметной области и постановка задачи на разработку		
Анализ актуальности разработки	Аналитик Интернет	50% 500р.
Сравнительный анализ систем-аналогов	Аналитик Интернет	50% 500р.
Постановка задачи на разработку системы	Программист ПС Программист приложения	100% 100%
Постановка задачи на разработку ПС взаимодействия с СУБД	Программист ПС	100%
Анализ предметной области и постановка задачи на разработку завершены		
Разработка требований к системе		
Разработка требований к архитектуре системы	Программист приложения Программист ПС	100% 100%
Разработка требований к ПС взаимодействия с СУБД		
Функциональные требования	Программист ПС	100%
Требования к архитектуре ПС	Программист ПС	100%

Продолжение таблицы 28

Реализация методов	Программист ПС	100%
Разработка системы завершена		
Тестирование ПС взаимодействия с СУБД		
Выбор методов тестирования	Тестирующий	100%
Проведение тестирования	Тестирующий	100%
Тестирование ПС взаимодействия с СУБД завершено		
Интеграция с приложением пользователя	Программист приложения Программист ПС	100% 100%
Тестирование собранной системы	Тестирующий	100%
Завершение работы над проектом		
Оформление документации	Аналитик; Программист приложения; Программист ПС; Тестирующий; Интернет Печать	100% 100% 100% 100% 500р. 300р.
Сдача проекта	Программист приложения Программист ПС	100% 100%
Проект завершен		
Конец проекта		
Реализация методов	Программист ПС	100%
Разработка системы завершена		
Тестирование ПС взаимодействия с СУБД		
Выбор методов тестирования	Тестирующий	100%
Проведение тестирования	Тестирующий	100%

На рисунке 14 показана диаграмма Ганта, которая наглядно отображает план выполнения задач проекта и распределенные для них ресурсы.

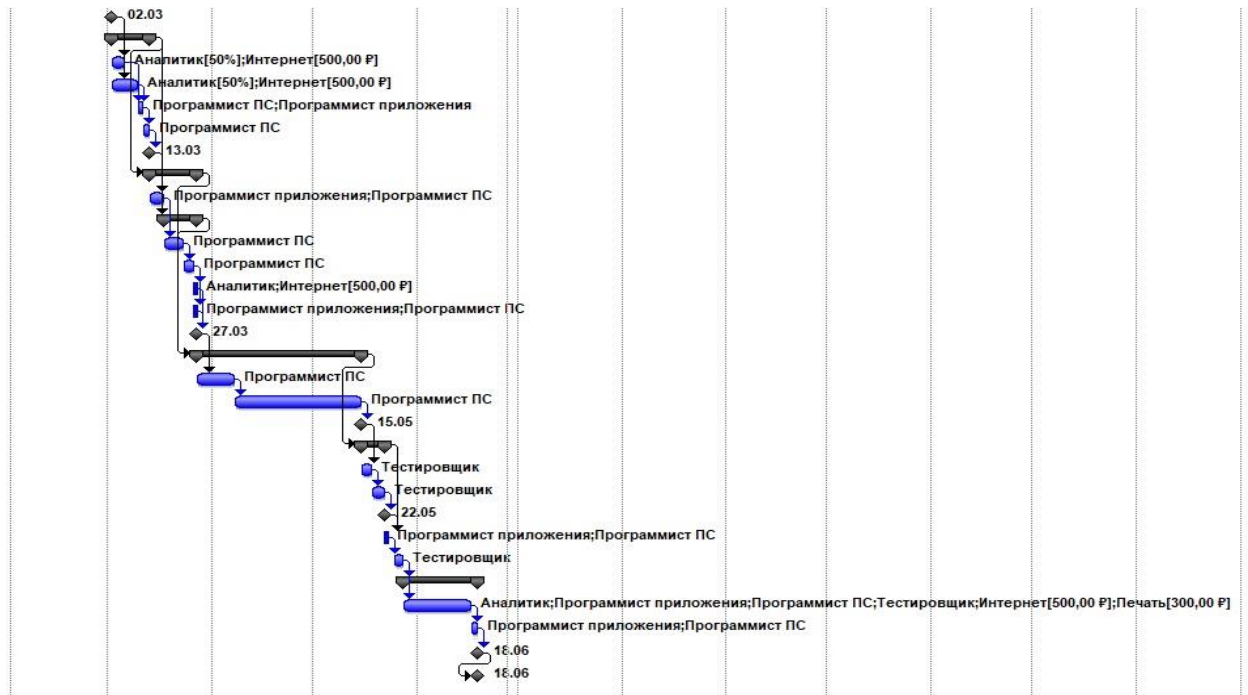


Рисунок 14 – Диаграмма Ганта

В Microsoft Project можно сформировать различные отчеты, среди которых наиболее распространенными являются отчет о бюджете и сводка по проекту. Отчет о бюджете отражает задачи проекта в порядке убывания по объему использованных ими денежных средств. Сводка по проекту показывает прогресс и общие сведения по проекту. На рисунках 15 и 16 представлены отчет о бюджете и сводка по проекту соответственно.

Ид.	Название задачи	Фиксированные затраты	Начисление фикс. затрат	Общие затраты	Базовые	Отклонение
27	Оформление документации	0,00 Р	Пропорциональное	63 520,00 Р	0,00 Р	63 520,00 Р
18	Реализация методов	0,00 Р	Пропорциональное	40 320,00 Р	0,00 Р	40 320,00 Р
1	Начало проекта	0,00 Р	Пропорциональное	25 000,00 Р	0,00 Р	25 000,00 Р
17	Разработка классов ПС	0,00 Р	Пропорциональное	10 080,00 Р	0,00 Р	10 080,00 Р
9	Разработка требований к архитектуре	0,00 Р	Пропорциональное	5 760,00 Р	0,00 Р	5 760,00 Р
11	Функциональные требования	0,00 Р	Пропорциональное	5 760,00 Р	0,00 Р	5 760,00 Р
28	Сдача проекта	0,00 Р	Пропорциональное	5 760,00 Р	0,00 Р	5 760,00 Р
5	Постановка задачи на разработку	0,00 Р	Пропорциональное	4 320,00 Р	0,00 Р	4 320,00 Р
12	Требования к архитектуре ПС	0,00 Р	Пропорциональное	4 320,00 Р	0,00 Р	4 320,00 Р
22	Проведение тестирования	0,00 Р	Пропорциональное	3 840,00 Р	0,00 Р	3 840,00 Р
6	Постановка задачи на разработку ПС	0,00 Р	Пропорциональное	2 880,00 Р	0,00 Р	2 880,00 Р
14	Выбор средств разработки системы	0,00 Р	Пропорциональное	2 880,00 Р	0,00 Р	2 880,00 Р
25	Тестирование собранной системы	0,00 Р	Пропорциональное	2 880,00 Р	0,00 Р	2 880,00 Р
4	Сравнительный анализ систем-аналогов	0,00 Р	Пропорциональное	2 420,00 Р	0,00 Р	2 420,00 Р
3	Анализ актуальности разработки	0,00 Р	Пропорциональное	1 780,00 Р	0,00 Р	1 780,00 Р
24	Интеграция с приложением пользователя	0,00 Р	Пропорциональное	1 440,00 Р	0,00 Р	1 440,00 Р
13	Выбор поддерживаемых СУБД	0,00 Р	Пропорциональное	1 140,00 Р	0,00 Р	1 140,00 Р
21	Выбор методов тестирования	0,00 Р	Пропорциональное	960,00 Р	0,00 Р	960,00 Р
7	Анализ предметной области и постановка задачи	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
15	Разработка требований к системе	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
19	Разработка системы завершена	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
23	Тестирование ПС взаимодействия	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
29	Проект завершен	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
30	Конец проекта	0,00 Р	Пропорциональное	0,00 Р	0,00 Р	0,00 Р
		0,00 Р		185 060,00 Р	0,00 Р	185 060,00 Р

Рисунок 15 – Отчет о бюджете

<u>Даты</u>			
Начало:	Пн 02.03.20	Окончание:	Чт 18.06.20
Баз. начало:	НД	Баз. окончание:	НД
Факт. начало:	НД	Факт. окончание:	НД
Отклон. начала:	0 дней	Отклон. окончания:	0 дней
<u>Длительность</u>			
Текущий план:	79 дней	Оставшаяся:	79 дней
Базовый план:	0 дней?	Фактическая:	0 дней
Отклонение:	79 дней	Процент завершения	0%
<u>Трудозатраты</u>			
Текущий план:	1 024 ч	Оставшиеся:	1 024 ч
Базовый план:	0 ч	Фактические:	0 ч
Отклонение:	1 024 ч	Процент завершения	0%
<u>Затраты</u>			
Текущий план:	185 060,00 Р	Оставшиеся:	185 060,00 Р
Базовый план:	0,00 Р	Фактические:	0,00 Р
Отклонение:	185 060,00 Р		
<u>Состояние задачи</u>		<u>Состояние ресурса</u>	
Еще не начатые задачи:	30	Трудовые ресурсы:	6
Выполняющиеся задачи:	0	Трудовые ресурсы с прев. дост	0
Завершенные задачи:	0	Материальные ресурсы:	1
Итого задач:	30	Итого ресурсов:	7

Рисунок 16 – Сводка по проекту

Построенный отчет о бюджете показал, что общая стоимость разработки ПС взаимодействия с СУБД составила 185060 рублей. Исходя из сводки по проекту, следует, что в ходе реализации проекта было выполнено 30 задач. Проект был выполнен на 100% за 79 дней.

Заключение

В ходе выполнения выпускной квалификационной работы был проведен анализ предметной области автоматизированного заполнения БД, в процессе которого было рассмотрено современное состояние исследований по данной проблеме, а также проведен сравнительный анализ существующих систем генерации случайных данных и автозаполнения БД, который позволил определить основной функционал разрабатываемой системы и пути ее усовершенствования. Кроме этого, определены функциональные и архитектурные требования к системе автозаполнения БД и ПС взаимодействия с СУБД, в частности.

В результате были разработаны ПС взаимодействия с СУБД для прототипа системы автозаполнения БД с плагином доступа к MS SQL Server в составе.

Тестирование ПС взаимодействия с СУБД заключалось в выборе метода тестирования и непосредственном проведении тестов. Результаты модульного тестирования показали, что ПС взаимодействия с СУБД для системы автозаполнения БД работают корректно.

В разработанном прототипе системы автозаполнения БД использована плагин-технология, что в дальнейшем позволит совершенствовать систему и адаптировать ее к новым СУБД без значительных затрат времени и труда.

Список использованных источников

1. Бинус, П.И. Сравнительный анализ средств автоматического заполнения баз данных / П.И. Бинус, И.С. Шушарина // НИТиС: материалы XVI Международной научно-технической конференции. – Пенза: Изд-во ПГУ, 2019. – С. 190 – 193.
2. Бинус, П.И. Функциональные возможности системы автоматического заполнения баз данных / П.И. Бинус, И.С. Шушарина // Информационные технологии в науке и образовании. Проблемы и перспективы: материалы VII ежегодной всероссийской межвузовской научно-практической конференции. – Пенза: Изд-во ПГУ, 2020. – С. 123 – 125.
3. Коннолли, Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание / Томас Коннолли, Каролин Бегг, Анна Страчан. – М.: Издательский дом «Вильямс», 2003 – 1440 с.
4. Демаков, А.В. Генерация тестовых данных сложной структуры с учетом контекстных ограничений / А.В. Демаков, С.В. Зеленов, С.А. Зеленова // Труды института системного программирования РАН. – 2006. – Т. 9. С. 83 – 96.
5. Костычев, Е.А. Нацеленная генерация данных для тестирования приложений над базами данных / Е.А. Костычев, В.А. Омельченко, С.В. Зеленов // Труды института системного программирования РАН. – 2011. – Т. 20. С. 253-268.
6. Демаков, А.В. Генератор сложных данных Pinery: реализация новых возможностей UniTESK / А.В. Демаков, С.В. Зеленов, С.А. Зеленова // Труды института системного программирования РАН. – 2008. – Т. 14. № 1. С. 119 – 136.
7. ООО Тестматик – Утилиты для генерации тестовых данных [Электронный ресурс]. – URL: <https://testmattick.com/ru/utility-dlya-generatsii-testovyh-dannyh/> (дата обращения 18.03.2020).

8. ООО «EMC Софтваре Девелопмент» – EMS DATA GENERATOR [Электронный ресурс]. – URL: <https://www.sqlmanager.net/ru/products/datagenerator> (дата обращения 18.03.2020).
9. DTM soft – DTM Data Generator [Электронный ресурс]. – URL: <http://www.sqledit.com/dg/> (дата обращения 18.03.2020).
10. DBeaver Community – Universal Database Tool [Электронный ресурс]. – URL: <https://dbeaver.io/> (дата обращения 18.03.2020).
11. We Manage IT – Red Gate SQL Data Generator [Электронный ресурс]. – URL: <https://www.sigirgroup.ru/programmirovanie/red-gate-sql-data-generator.html> (дата обращения 18.03.2020).
12. Devart – DBFORGE STUDIO [Электронный ресурс]. – URL: <https://www.devart.com/ru/dbforge/sql/studio/features.html> (дата обращения 18.03.2020).
13. Microsoft User Group Community – Создание Plugin архитектуры с помощью C# [Электронный ресурс]. – URL: <http://msugvnua000.web710.discountasp.net/Posts/Details/3477> (дата обращения 21.04.2020).
14. Wrike, Inc. – Что такое программный интерфейс приложений (API) [Электронный ресурс]. – URL: <https://www.wrike.com/ru/blog/chto-takoe-programmnyj-interfejs-prilozhenij-api/> (дата обращения 30.04.2020).
15. Кошелев, О.В. Шаблоны проектирования в программировании / О.В. Кошелев // РТУ МИРЭА. – 2018. – № 4 (27). С. 43-49.
16. Мацяшек, А.Т. Анализ требований и проектирование систем. Разработка информационных систем с использованием UML / Т.А Мацяшек, А. Лешек, – М.; Издательский дом «Вильямс» 2013. – 100 с.
17. Коржов, В. Многоуровневые системы клиент-сервер / В. Коржов // Сети. –1997 – №06. – С. 29– 31.
18. СССР блог – Что такое плагины: плюсы и минусы для разработчиков и пользователей [Электронный ресурс]. – URL: <http://ccsr-blog.com/sozdanie-saytov/chto-takoe-plaginy> (дата обращения 10.05.2020).

19. Шумаков, П. В. ADO.NET и создание приложений баз данных в среде Microsoft Visual Studio .NET. Руководство разработчика с примерами на C# / П. В. Шумаков. – М.: ДИАЛОГ-МИФИ, 2003 – 528 с.
20. Riel, A.J. Object-Oriented Design Heuristics / Riel, A.J. – М.: Addison-Wesley, 1996 – 380 с.
21. Gartner, Inc. – Magic Quadrant for Operational Database Management Systems [Электронный ресурс]. – URL: <https://www.gartner.com/doc/reprints?id=1-1XT9MEFJ&ct=191125&st=sb> (дата обращения 02.05.2020).
22. Забудский, Е.И. Учебно-методический комплекс дисциплины «Объектно-ориентированный анализ и программирование». М.: Кафедра АПС ГУ-ВШЭ, 2008.
23. ITVDN – C# против Java: какой язык программирования общего назначения выбрать? [Электронный ресурс]. – URL: https://itvdn.com/ru/blog/article/csharp_vs_java (дата обращения 10.05.2020).
24. MSNET.NAROD.RU. – Сравнение C#, Java и C++? [Электронный ресурс]. – URL: http://msnet.narod.ru/art/art_001/art_001.htm (дата обращения 11.05.2020).
25. Кригель, А. SQL. Библия пользователя, 2-е издание / А. Кригель, Б. Трухнов. – М.: ООО “И.Д. Вильямс”, 2010 – 752 с.
26. Microsoft Corporation – Visual Studio IDE [Электронный ресурс]. – URL: <https://visualstudio.microsoft.com/ru/> (дата обращения 12.05.2020).
27. METANIT – Язык C# и платформа .NET Core [Электронный ресурс]. – URL: <https://metanit.com/sharp/tutorial/1.1.php> (дата обращения 12.05.2020).

Приложение А
Листинг моделей представления данных
(рекомендуемое)

Исходный код классов модели представления данных Column:

```
namespace Project.Models
{
    public class Column
    {
        public string Name { get; set; }
        public bool Primary { get; set; }
        public KeyInfo Foreign { get; set; }
        public bool Nullable { get; set; }
        public string ColumnType { get; set; }
        public int? Size { get; set; }
    }

    public class KeyInfo
    {
        public string Table { get; set; }
        public string Column { get; set; }
    }
}
```

Исходный код класса модели представления данных Connection:

```
namespace Project.Models
{
    public class Connection
    {
        public string Server { get; set; }
        public string Port { get; set; }
        public string Login { get; set; }
        public string Password { get; set; }
    }
}
```

Исходный код класса модели представления данных ResultAction:

```
namespace Project.Models
{
    public class ResultAction<T>
    {
        public bool Success { get; set; }
        public string Error { get; set; }
        public T Result { get; set; }
    }
}
```

Исходный код классов модели представления данных Rule:

```
using System;

namespace Project.Models
{
    public class RuleBase
    {
        public RuleType Type { get; set; }
    }

    public class RuleConstant : RuleBase
    {

```

```

        public RuleValue Value { get; set; }
    }

    public class RuleValue
    {
        public string Type { get; set; }
        public Type Value { get; set; }
    }

    public class RuleRandom : RuleBase
    {
        public RuleCustom Value { get; set; }
    }

    public enum RuleType
    {
        Constant,
        Random,
        Dictionary
    }

    public class RuleCustom
    {
        public string Type { get; set; }
        public RuleRange Range { get; set; }
        public RuleLink Link { get; set; }
    }

    public class RuleLink
    {
        public string Table { get; set; }
        public string Column { get; set; }
    }

    public class RuleRange
    {
        public Type MinValue { get; set; }
        public Type MaxValue { get; set; }
        public bool Increment { get; set; }
    }
}

```

Исходный код классов модели представления данных TableData:

```

using System;
using System.Collections.Generic;

namespace Project.Models
{
    public class TableData
    {
        public long Count { get { return Rows.Count; } }
        public List<RowData> Rows { get; set; } = new List<RowData>();
    }

    public class RowData
    {
        public List<ColumnData> Columns { get; set; } = new List<ColumnData>();
    }

    public class ColumnData

```

```
{  
    public string Name { get; set; }  
    public string Type { get; set; }  
    public Type Value { get; set; }  
}  
}
```

Приложение Б
Листинг интерфейса плагинов
(рекомендуемое)

```

using Project.Models;
using System.Collections.Generic;

namespace Project.Interfaces
{
    public interface IPlugin
    {
        string Name { get; set; }
        string Description { get; set; }

        ResultAction<List<string>> GetServers();
        ResultAction<List<string>> GetDatabases(Connection connection);
        ResultAction<List<string>> GetTables(Connection connection, string database);
        ResultAction<List<Column>> GetColumns(Connection connection, string database, string
table);

        ResultAction<TableData> GetData(Connection connection, string database, string
table);

        bool CreateTable(Connection connection, string database, string table, List<Column>
columns);
        bool RemoveTable(Connection connection, string database, string table);

        bool InsertData(Connection connection, string database, string table, TableData
data);
        bool GenerateData(Connection connection, string database, string table, int rows,
Dictionary<string, RuleBase> rules);
    }
}

```

Приложение В
Листинг плагина взаимодействия с СУБД MS SQL Server
(рекомендуемое)

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.Sql;
using System.Data.SqlClient;
using Interface;
using Project.Models;

namespace SqlPlugin
{
    public class SqlServerPlugin : IPlugin
    {
        private string PluginName = "MS SQL Server plugin";
        private string PluginDescription = "Плагин для работы с MS SQL Server";

        public string Name
        {
            get
            {
                return PluginName;
            }
            set
            {
                PluginName = value;
            }
        }

        public string Description
        {
            get
            {
                return PluginDescription;
            }
            set
            {
                PluginDescription = value;
            }
        }

        private SqlConnectionStringBuilder BuildStringConnection(Connection
connection, string database)
        {
            bool security = true;
            string server = "";
            if (connection.Login == "" || connection.Password == "")
            {
                security = true;
                server = connection.Server;
            }
            else if (connection.Port == "")
            {
                security = false;
                server = connection.Server;
            }
            else // удаленное подключение
            {
                security = false;
            }
        }
    }
}

```

```

        server = connection.Server + "," + connection.Port;
    }
    return new SqlConnectionStringBuilder
    {
        DataSource = server,
        InitialCatalog = database,
        ConnectTimeout = 30,
        IntegratedSecurity = security,
        UserID = connection.Login,
        Password = connection.Password,
    };
}

private ResultAction<T> BuildResultAction<T>(bool success, ref T result,
string error)
{
    return new ResultAction<T>
    {
        Success = success,
        Result = result,
        Error = error,
    };
}

public ResultAction<List<string>> GetServers()
{
    List<string> servers_list = new List<string>();
    ResultAction<Connection> result = new ResultAction<Connection>();
    try
    {
        DataTable dt = SqlDataSourceEnumerator.Instance.GetDataSources();
        foreach (System.Data.DataRow row in dt.Rows)
        {
            string name = row[0] as string;
            string instance = row[1] as string;
            if (instance != null)
            {
                instance = @"\" + instance;
            }
            servers_list.Add(name + instance);
        }
        return BuildResultAction(true, ref servers_list, "");
    }
    catch (Exception e)
    {
        return BuildResultAction(false, ref servers_list, e.Message);
    }
}

public ResultAction<List<string>> GetDatabases(Connection connection)
{
    SqlConnectionStringBuilder connect = BuildStringConnection(connection,
    "");
    List<string> db_list = new List<string>();
    using (SqlConnection con = new SqlConnection())
    {
        con.ConnectionString = connect.ConnectionString;
        try
        {
            con.Open();

```

```

        using (SqlCommand cmd = new SqlCommand("SELECT name FROM
sys.databases", con))
        {
            using (IDataReader dr = cmd.ExecuteReader())
            {
                while (dr.Read())
                {
                    db_list.Add(dr[0].ToString());
                }
            }
            return BuildResultAction(true, ref db_list, "");
        }
        catch (Exception e)
        {
            return BuildResultAction(false, ref db_list, e.Message);
        }
    }

    public ResultAction<List<string>> GetTables(Connection connection, string
database)
    {
        SqlConnectionStringBuilder connect = BuildStringConnection(connection,
database);
        List<string> table_list = new List<string>();
        using (SqlConnection con = new SqlConnection())
        {
            con.ConnectionString = connect.ConnectionString;
            try
            {
                con.Open();
                SqlCommand cmd = new SqlCommand("SELECT name FROM sys.Tables",
con);

                System.Data.SqlClient.SqlDataReader reader = cmd.ExecuteReader();
                while (reader.Read())
                {
                    table_list.Add(reader["name"].ToString());
                }
                return BuildResultAction(true, ref table_list, "");
            }
            catch (Exception e)
            {
                return BuildResultAction(false, ref table_list, e.Message);
            }
        }
    }

    public ResultAction<List<Column>> GetColumns(Connection connection, string
database, string table)
    {
        SqlConnectionStringBuilder connect = BuildStringConnection(connection,
database);
        List<Column> column_list = new List<Column>();
        string cmdString = String.Format("SELECT * FROM {0}", table);

        using (SqlConnection con = new SqlConnection())
        {
            con.ConnectionString = connect.ConnectionString;
            try
            {
                con.Open();

```

```

con))
        using (SqlDataAdapter adapter = new SqlDataAdapter(cmdString,
        {
            adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;
            var dtab = new DataTable();
            adapter.FillSchema(dtab, SchemaType.Source);
            foreach (DataColumn col in dtab.Columns)
            {
                Column item = new Column();
                item.Name = col.ColumnName;
                item.Nullable = col.AllowDBNull;
                item.ColumnType = col.DataType.ToString();
                item.Size = col.MaxLength;
                item.Primary = dtab.PrimaryKey.Contains(col);
                KeyInfo foreign = new KeyInfo();
                foreign.Table = "";
                foreign.Column = "";
                item.Foreign = foreign;
                column_list.Add(item);
            }
        }
        return BuildResultAction(true, ref column_list, "");
    }
    catch (Exception e)
    {
        return BuildResultAction(false, ref column_list, e.Message);
    }
}

string GetTypeColumn(string column_type, string column_size)
{
    string type = "";
    switch (column_type)
    {
        case "System.Int32":
            type = "INT";
            break;

        case "System.String":
            type = @"VARCHAR({column_size})";
            break;
    }
    return type;
}

string BuildCreateQuery(string table_name, List<Column> columns)
{
    StringBuilder query = new StringBuilder();
    query.Append("CREATE TABLE ");
    query.Append(table_name);
    query.Append(" ( ");

    //for (int i = 0; i < columns.Count; i++)
    foreach (var item in columns)
    {
        query.Append(item.Name);
        query.Append(" ");
        if (item.Primary)
        {
            query.Append("PRIMARY KEY");
            query.Append(" ");
        }
    }
}

```

```

        if (!item.Nullable)
        {
            query.Append("NOT NULL");
            query.Append(" ");
        }
        query.Append(GetTypeColumn(item.ColumnType, item.Size.ToString()));
        query.Append(", ");
    }

    if (columns.Count > 1) { query.Length -= 2; } // удаляем последнее ", "
    query.Append(")");

    return query.ToString();
}

public bool CreateTable(Connection connection, string database, string table,
List<Column> column)
{
    SqlConnectionStringBuilder connect = BuildStringConnection(connection,
database);
    List<string> db_list = new List<string>();
    using (SqlConnection con = new SqlConnection())
    {
        con.ConnectionString = connect.ConnectionString;
        try
        {
            {
                con.Open();
                using (SqlCommand cmd = new SqlCommand(BuildCreateQuery(table,
column), con))
                {
                    cmd.ExecuteNonQuery();
                }
                return true;
            }
        }
        catch // (Exception e)
        {
            return false;
        }
    }
}

string BuildDropQuery(string table_name)
{
    StringBuilder query = new StringBuilder();
    query.Append("DROP TABLE IF EXISTS ");
    query.Append(table_name);
    return query.ToString();
}

public bool RemoveTable(Connection connection, string database, string table)
{
    SqlConnectionStringBuilder connect = BuildStringConnection(connection,
database);
    List<string> db_list = new List<string>();
    using (SqlConnection con = new SqlConnection())
    {
        con.ConnectionString = connect.ConnectionString;
        try
        {
            {
                con.Open();
                using (SqlCommand cmd = new SqlCommand(BuildDropQuery(table),
con))

```

```

        {
            cmd.ExecuteNonQuery();
        }
        return true;
    }
    catch //(Exception e)
    {
        return false;
    }
}

}

public ResultAction<TableData> GetData(Connection connection, string
database, string table)
{
    SqlConnectionStringBuilder connect = BuildStringConnection(connection,
database);
    TableData table_data = new TableData();
    List<Column> column_list = new List<Column>();
    string cmdString = String.Format("SELECT * FROM {0}", table);

    using (SqlConnection con = new SqlConnection())
    {
        con.ConnectionString = connect.ConnectionString;
        try
        {
            con.Open();
            using (SqlDataAdapter adapter = new SqlDataAdapter(cmdString,
con))
            {
                DataTable data_table = new DataTable();
                adapter.Fill(data_table);

                foreach (var row in data_table.Rows)
                {
                    //foreach(var col in row.)
                    //!!! здесь заполняем table_data
                }
            }
            return BuildResultAction(true, ref table_data, "");
        }
        catch (Exception e)
        {
            return BuildResultAction(false, ref table_data, e.Message);
        }
    }
}

public bool InsertData(Connection connection, string database, string table,
TableData data)
{
    SqlConnectionStringBuilder connect = BuildStringConnection(connection,
database);
    string cmdString = String.Format("select * from {0} where 0 = 1",
table);

    using (SqlConnection con = new SqlConnection())
    {
        con.ConnectionString = connect.ConnectionString;
        try
        {

```

```

con.Open();
using (SqlDataAdapter adapter = new SqlDataAdapter(cmdString,
con))
{
    DataTable data_table = new DataTable();
    DataSet dataSet = new DataSet();
    adapter.Fill(dataSet);

    var newRow = dataSet.Tables[table].NewRow();
    //newRow["имя столбца"] = 55;
    //здесь заполнить строку
    dataSet.Tables[table].Rows.Add(newRow);

    new SqlCommandBuilder(adapter);
    adapter.Update(dataSet);
}
return true;
}
catch
{
    return false;
}
}

public bool GenerateData(Connection connection, string database, string
table, int rows, Dictionary<string, RuleBase> rules)
{
    TableData data_table = new TableData();
    InsertData(connection, database, table, data_table);
    return true;
}
}
}

```