

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

МАТЕМАТИКИ ФИЗИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

(институт)

Прикладная математика и информатика

(кафедра)

09.03.03 Прикладная информатика

(код и наименование направления подготовки, специальности)

Бизнес-информатика

(наименование профиля, специализации)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

На тему: Разработка компаратора баз данных по критерию для
информационной системы управления предприятием (на примере ООО
«РОСТ»)

Студент(ка)	<u>Мазина Юлия Сергеевна</u>	_____
	(И.О. Фамилия)	(личная подпись)
Руководитель	<u>Ерофеева Елена Александровна</u>	_____
	(И.О. Фамилия)	(личная подпись)

Допустить к защите

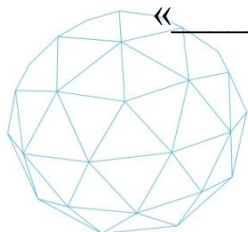
Заведующий кафедрой _____

(ученая степень, звание, И.О. Фамилия)

(личная подпись)

« _____ » _____ 20 _____ Г.

Тольятти 2019



ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	ОШИБКА! ЗАКЛАДКА НЕ ОПРЕДЕЛЕНА.
ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ.....	8
1.1. Техничко-экономическая характеристика предметной области.....	8
1.1.1. Характеристика предприятия.....	8
1.1.2. Краткая характеристика подразделения и его видов деятельности ...	11
1.1.3. Сущность задачи автоматизации	14
1.2. Концептуальное моделирование предметной области	15
1.3. Постановка задачи.....	20
1.3.1. Цель и назначение автоматизированного варианта решения задачи.	20
1.3.2. Общая характеристика организации решения задачи на ЭВМ	20
1.4. Анализ существующих разработок и обоснование выбора технологии проектирования	22
1.4.1. Определение критериев анализа	22
1.4.2. Сравнительная характеристика существующих разработок.....	23
1.5. Выводы	28
ГЛАВА 2. РАЗРАБОТКА И РЕАЛИЗАЦИЯ ПРОЕКТНЫХ РЕШЕНИЙ	30
2.1. Логическое моделирование предметной области.....	30
2.1.1. Логическая модель и ее описание.....	30
2.1.2. Используемые классификаторы и системы кодирования	30
2.1.3. Характеристика нормативно-справочной и входной оперативной информации.....	31
2.1.4. Характеристика базы данных	32
2.1.5. Характеристика результатной информации	34
2.2. Физическое моделирование АИС.....	35
2.2.1. Выбор архитектуры АИС.....	35
2.2.2. Функциональная схема проекта	50
2.2.3. Структурная схема проекта	50
2.2.4. Описание программных модулей.....	51

2.2.5. Схема взаимосвязи программных модулей и информационных файлов	53
2.3. Технологическое обеспечение задачи.....	53
2.3.1. Организация технологии сбора, передачи, обработки и выдачи информации.....	53
2.3.2. Схема технологического процесса сбора, передачи, обработки и выдачи информации	57
2.4. Контрольный пример реализации проекта.....	57
ГЛАВА 3. ОЦЕНКА И ОБОСНОВАНИЕ ЭКОНОМИЧЕСКОЙ ЭФФЕКТИВНОСТИ ПРОЕКТА.....	59
3.1. Выбор и обоснование методики расчета экономической эффективности.....	59
3.2. Расчет показателей экономической эффективности.....	63
ЗАКЛЮЧЕНИЕ.....	68
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ.....	70
ПРИЛОЖЕНИЕ.....	73

ВВЕДЕНИЕ

Базы данных представляют собой определенные наборы данных, которые зачастую связаны объединяющими признаком или свойством (или и тем, и другим) и представляют некоторые аспекты реального мира. Такие данные находятся в определенном порядке, к примеру, в алфавитном. Изобилие разнообразных данных, которые возможно поместить в одну базу, приводит к множеству вариантов того, что можно записать: личных данных пользователей, записей, дат, заказов и т. д. Например, в случае с интернет-магазином, база данных такого сайта может быть заполнена прайс-листами, каталогами товаров или услуг, отчетами, статистикой и информацией о клиентах. Система базы данных предназначена для построения и наполнения данными для определенной задачи.

СУБД (система управления базами данных) — это пользовательская оболочка, направленная на то, чтобы незамедлительно управлять запросами пользователей. Однако для отечественных пользователей это является менее значимым вследствие трудностей с пониманием англоязычного интерфейса. Система управления базами данных представляет собой программное обеспечение для хранения и извлечения данных пользователей с учетом соответствующих мер безопасности. Это позволяет пользователям создавать свои собственные базы данных в соответствии с запрашиваемыми ими требованиями.

Присутствие в СУБД языков программирования обеспечивает создание сложных систем обработки данных, решающих конкретные задачи. Основной особенностью СУБД является присутствие процедур, поддерживающих ввод и хранение данных, а также описаний структуры этих данных. Файлы, содержащие описание данных, которые в них хранятся, и управляемые СУБД, ранее назвались банками данных, а затем «Базами данных».

СУБД состоит из группы программ, которые управляют базой данных и обеспечивают интерфейс базы данных. Он включает в себя пользователя базы данных и другие прикладные программы.

СУБД принимает запрос данных из приложения и инструктирует операционную систему предоставить конкретные данные. В больших системах СУБД помогает пользователям и другому стороннему программному обеспечению хранить и извлекать данные.

Характеристики системы управления базами данных:

- обеспечивает безопасность и устраняет избыточность;
- само описывающая природа системы баз данных;
- изоляция между программами и абстракция данных;
- поддержка нескольких просмотров данных;
- обмен данными и обработка многопользовательских транзакций;
- СУБД позволяет сущностям и отношениям между ними формировать таблицы;
- она следует концепции ACID;
- СУБД поддерживает многопользовательскую среду, которая позволяет пользователям получать доступ к данным и манипулировать ими параллельно.

Среди преимуществ СУБД можно отметить то, что она предлагает различные методы для хранения и извлечения данных, служит эффективным обработчиком для балансирования потребностей нескольких приложений, использующих одни и те же данные, имеет единые административные процедуры для данных, разработчики приложений никогда не сталкиваются с деталями представления и хранения данных, использует различные мощные функции для эффективного хранения и извлечения данных, обеспечивает целостность данных и безопасность, подразумевает ограничения целостности для обеспечения высокого уровня защиты от запрещенного доступа к данным, планирует одновременный доступ к данным таким образом, что

только один пользователь может получить доступ к одним и тем же данным одновременно, имеет сокращенное время разработки приложений.

Но стоит отметить, что стоимость аппаратного и программного обеспечения СУБД довольно высока, что увеличивает бюджет организации и большинство систем управления базами данных часто являются сложными системами, поэтому требуется обучение пользователей использованию СУБД, а также в некоторых организациях все данные интегрированы в единую базу данных, которая может быть повреждена из-за сбоя электропитания или повреждена база данных на носителе. Использование одной и той же программы одновременно многими пользователями иногда приводит к потере некоторых данных. СУБД не может выполнять сложные вычисления

Компаратор – представляет собой устройство для сравнения различных сигналов и данных. Впервые понятие компаратора появилось в электротехнике, где этим термином называется специальное устройство сравнения и преобразования сигналов.

Компаратор, применительно к базам данных – специализированное программное обеспечение, которое позволяет производить сравнение данных из двух разнородных СУБД. Иногда компараторы предоставляют дополнительные возможности, такие как перенос недостающих данных из одной базы данных в другую, либо возможности резервного копирования.

Объектом данной работы являются предприятие ООО «Рост».

Предметом работы является компаратор баз данных.

Целью работы является разработка компаратора баз данных по критерию для рассматриваемого предприятия.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Дать общую характеристику предприятия;
2. Дать постановку задачи и пояснить сущность задачи автоматизации;

3. Провести концептуальное моделирование предметной области;
4. Проанализировать существующие разработки по теме;
5. Выбрать средства разработки системы;
6. Разработать и описать программное средство.

По своей структуре работа состоит из введения, трех глав, заключения, списка используемой литературы и приложений.

ГЛАВА 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Техничко-экономическая характеристика предметной области

1.1.1. Характеристика предприятия

Общие сведения об организации:

Полное наименование: Общество с ограниченной ответственностью «Рост»;

Местоположение: юридический адрес: Россия, г Москва, ул. Сретенка, д. 12.

Форма собственности: Частная;

Организационно-правовая форма: Общество с ограниченной ответственностью;

Отрасль: Регистратор. Профессиональный участник рынка ценных бумаг, обладающий лицензией, дающей право осуществления деятельности по ведению реестра. Ведением реестра называется деятельность регистратора, являющаяся исключительной, которая осуществляется при наличии лицензии, выданной Центральным банком РФ (до изменений 1 сентября 2013 г. — Службой Банка России по финансовым рынкам). Предприятие до октября 2014 г. могло заниматься осуществлением ведения реестра владельцев именных ценных бумаг мог самостоятельно эмитент, при условии, что количество акционеров было не больше 50. С 1 октября 2014 г. был установлен запрет на осуществление самостоятельного ведения эмитентами реестра акционеров.

На основании договоров, заключаемых с эмитентами ценных бумаг, управляющими компаниями, управляющими ипотечными покрытиями, регистратором ведется реестр владельцев ценных бумаг (акционеры, владельцы инвестиционного пая, владельцы сертификата участия). Реестр является системой записей о лице, которому открыт лицевой счет (зарегистрированное лицо) и о ценных бумагах, которые учитываются на

таком счету. Регистратор отвечает за достоверность и полноту информации, предоставляемой из реестра. От лица регистратора и по договору с ним осуществление функций, связанных с приемом и передачей данных и документов, используемых при исполнении операций в реестре, от зарегистрированного лица регистратору и обратно может осуществляться уполномоченным регистратором юридическим лицом – трансфер-агентом.

Регистратор обязан входить в СРО¹ профучастников рынка ценных бумаг. В качестве таковых СРО можно привести Национальную фондовую ассоциацию (НФА) и Профессиональную ассоциацию регистраторов, трансфер-агентов и депозитариев (ПАРТАД). Регистраторы должны состоять по меньшей мере в одной из СРО. Всего в России около 35 регистраторов.

Общество с ограниченной ответственностью «Рост» основано в 1994 году и является одним из старейших регистраторов России. Общество представляет собой юридическое лицо, действующее на основании Устава и законодательства РФ.

Общество «Рост» является предприятием, занятым в сфере ценных бумаг, основная цель которого заключается в получении прибыли посредством объединения материальных, трудовых, интеллектуальных и финансовых ресурсов участников общества.

Видами услуг предоставляемых АО «Рост» в городе Москва являются:

Ведение и учет ценных бумаг на лицевых счетах зарегистрированных лиц;

Ведение и учет ценных бумаг на казначейском и эмиссионном счетах Эмитента;

Проведение в реестре всех типов операций, которые предусматривает законодательство Российской Федерации;

Ведение регистрационного журнала в хронологическом порядке в бумажном и электронном виде по всем ценным бумагам Эмитента;

¹ Саморегулируемая организация

Проведение ежедневных сверок государственного регистрационного номера, вида, категории (типа), количества размещенных ценных бумаг с количеством ценных бумаг, которые учитываются на счету зарегистрированного лица;

Проведение с номинальным держателем – центральным депозитарием ежедневных сверок записей относительно количества на лицевых счетах номинальных держателей, открытых в реестре, на предмет соответствия количеству ценных бумаг, которые учитываются на каждом счету депо и ином счете, в том числе неустановленных лиц, которые осуществляются номинальным держателем – центральным депозитарием;

Проведение сверки количества ценных бумаг, учитываемых номинальным держателем на открытых им счетах депо и иных счетах, в том числе счета неустановленного лица, с количеством ценных бумаг, которые учитываются держателем реестра на лицевом счете такого номинального держателя, после проведения операции в реестре по счету такого номинального держателя;

Предоставление информации из реестра по запросу Эмитента и зарегистрированных лиц;

Представление информации представителям государственных органов на основании письменного запроса, оформленного в соответствии с требованиями действующего законодательства;

Архивирование и учет информации, составляющей реестр владельцев ценных бумаг;

Обеспечение сохранности и конфиденциальности информации, содержащейся в реестре, а также любой другой информации, разглашение которой может нанести ущерб Эмитенту и/или зарегистрированным лицам;

Архивное хранение документов, являющихся основанием для внесения записей в реестр;

Организационная схема предприятия показана на рисунке 1.1.



Рис. 1.1. – Организационная схема предприятия

1.1.2. Краткая характеристика подразделения и его видов деятельности

В компьютерном парке имеются ПК, объединенные в сеть с применением высокопроизводительных серверов HPProLiantDL380 (G7, G9), LenovoThinkServerRD340 и управляемых L3 коммутаторов CISCO с поддержкой VLAN. Сервера находятся в двух серверных помещениях, разнесённых по разным этажам (основная и резервная) и соединены с внешними отказоустойчивыми сетевыми хранилищами (SAN) для данных виртуальных машин посредством 10 GbitEthernet и протокола ISCSI. На всех серверах используется технология виртуализации на основе VMwarevSphereESXi.

Основная серверная рассчитана на 2 и более часов автономной работы при отключении электричества в здании. Резервная серверная рассчитана на случай отказа основной, автономность до 1 часа. Для компьютеров пользователей используются индивидуальные ИБП с автономностью около

30 минут. В случае исчерпания их заряда и необходимости продолжить работу предусмотрены ноутбуки с автономностью до 4 часов.

SAN-хранилища SynologyRackStationRS18016xs+ форм-фактора 2U имеют в себе наборы из 12 высокоскоростных SAS-дисков со скоростью оборотов от 10000 до 15000 RPM и поддерживают их «горячую» замену. Диски работают в режиме массива RAID 6. RAID 6 массив продолжает работать при выходе из строя двух дисков без потери данных. Используется SSD-кэш для кэширования частых однотипных запросов к жестким дискам и ускорения чтения соответственно. Для надёжности хранилища используют два блока питания, подключенных к разным источникам питания (промышленным ИБП). Также сами SAN-устройства объединены в кластеры из двух хранилищ для отказоустойчивости (один сервер является пассивным и в случае отказа активного сразу принимает на себя работу). Хранилища поддерживают подключение дополнительных модулей для увеличения количества дисков и общего объема.

Компьютеры находятся во всех отделах организации и филиалах.

Таблица 1.1. – Характеристики сервера HPProLiantDL380 Gen9

Процессор	Intel Xeon E5-2620 v3 2.4 GHz (x2)
Жесткий диск	Внешний отказоустойчивый посредством сети 10 GbitEthernet и протокола ISCSI, объем не ограничен.
Оперативная память	DDR4 128GB ECC DIMM
Операционная система	VMWare VSphere ESXi 6.5
Монитор	KVM
Материнская плата	HP LGA2011-3 (2 CPU)

Таблица 1.2. – Характеристики типового закупаемого ПК
(Lenovo ThinkCentre S510 Small Form Factor)

Процессор	Intel Core i5-6400 Processor (6M Cache, 2.7 GHz, 3.3 GHz TurboBoost)
Жесткий диск	Western Digital 1Tb SATA-III
Оперативная память	DDR4 8GB DIMM
Операционная система	Windows 10 Pro x64
Монитор	23" AOC
Материнская плата	Lenovo LGA1151

Программная архитектура предприятия описана ниже.

В состав программного обеспечения на предприятии входят:

1. ПО «Зенит» (подсистема документооборота, созданная специально для регистраторов ценных бумаг компанией Элдис-Софт, с её помощью осуществляются все операции с основной базой данных MicrosoftSQLServer 2012).

2. MicrosoftSQLServer 2012 (в этой БД хранятся все необходимые для деятельности регистратора данные, из неё делаются выписки для эмитентов и др.).

3. ПО «Луч» (подсистема документооборота с эмитентами и трансфер-агентами посредством сети Интернет, защищенного посредством цифровой подписи).

4. 1С: БНФО 8.3 (Бухгалтерия некредитной финансовой организации КОПИ), 1С: ЗУП 8 (Зарплата и управление персоналом).

5. Гипервизоры VMWareVSphereESXi 6.5 + vCenter для централизованного управления и репликации.

6. OS Windows 7/8.1/10.

7. OS Linux.

8. OS FreeBSD.

9. OSCiscoIOS (в составе маршрутизаторов и управляемых коммутаторов Cisco).

10. Пакет MS Office 2007/2010.

11. Клиент-серверный антивирус SymantecEndpointProtection.
12. WindowsServer 2008R2.
13. WindowsServer 2012R2.
14. КриптоПРОСР.
15. OpenVPNServer (VPN для случаев когда не обязательно использовать ГОСТ шифрование).
16. Аппаратно-программный крипто-шлюз (АПКШ) Континент 3.7 (аппаратный VPN с применением ГОСТ шифрования для документооборота с филиалами).
17. ПО Континент-АП для соединения удаленных компьютеров с АПКШ.
18. MicrosoftTMG 2010 (пограничный с интернетом шлюз безопасности).
19. MicrosoftExchangeServer 2010.
20. MDaemonMailServer (пограничный почтовый сервер смарт-хост).
21. Microsoft SQL Server 2014.
22. Веб сервер (PHP+MySQL+Apache).

1.1.3. Сущность задачи автоматизации

В рассматриваемой организации используется несколько разнородных баз данных среди которых можно выделить MySQL и MSSQLServer2012, а также базу данных 1С предприятие. Данные между базами постоянно перемещаются, в частности при формировании отчетов о деятельности организации – данные из MSSQLServer переносятся в базу MySQL, которая связана с сайтом предприятия.

Для организации переноса данных между этими базами, сравнения изменений в них за определенное время, а также для обновления баз данных – предназначено разрабатываемое ПО.

В целом разрабатываемое ПО должно удовлетворять следующим требованиям:

- возможность работы с несколькими типами баз данных;
- возможность просмотра изменений в каждой из рассматриваемых баз по отношению к другой;
- возможность переноса недостающих данных из одной базы в другую;
- возможность сравнения схем рассматриваемых баз данных;
- возможность резервного копирования.

1.2. Концептуальное моделирование предметной области

Для построения концептуальной схемы предметной области мною был выбран стандарт IDEF0, который является методологией по функциональному моделированию (англ. «function modeling») и графической нотацией, предназначенной формализовать и описывать бизнес-процессы. Отличительная особенность IDEF0 — это её акцент на соподчинённости объектов. Стандарт занимается рассмотрением логических отношений между работами, а не их временной последовательности (потоки работ).

В IDEF0 организация представляется набором модулей, здесь присутствует правило — наиболее важную функцию следует помещать в верхний левый угол.

Описание схоже с «чёрным ящиком, имеющим входы, выходы, управление и механизм, детализация которого до нужного уровня происходит постепенно. Также в целях правильного понимания, присутствуют словари, описывающие активности и стрелки. С помощью этих словарей задаются описания смысла, вкладываемого в данные активности либо стрелки.



Методология IDEF0 описывается в рекомендациях Р 50.1.028-2001 "Информационные технологии поддержки жизненного цикла продукции. Методология функционального моделирования".

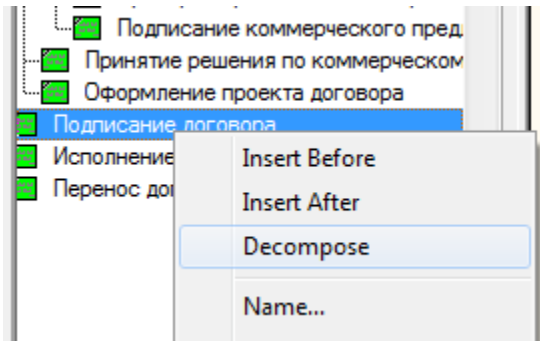
Осуществляется отображение всех сигналов управления, не отображаемых на DFD (диаграмма потоков данных). Данную модель используют для организации бизнес-процесса и проекта, которые берут за

основу моделирование всех процессов (административные и организационные).

Функциональная модель IDEF0 включает несколько элементов. В блоках находится процесс или действие. Стрелками слева указываются входные данные, стрелками справа указываются выходные данные процесса. Стрелками снизу указываются участники процесса, стрелками сверху – управляющие воздействия.

Главные объекты модели — это диаграммы, отображение всех организационных функций и интерфейсов между ними организовано в виде функциональных блоков и дуг. Место, соединения дуги и блока показывает тип интерфейса. Сверху в блок входит информация, которая управляет процессами, левая стороны содержит обрабатываемые входные данные в правой стороне располагаются результативные выходные данных. Механизмы (пользователи или АИС), осуществляющие операции, показывается снизу. Построение SADT - модели следует начинать представлением всей системы как исходного компонента - один блок и стрелка-дуга, изображающие воздействие извне системы. Подобная диаграмма носит название контекстной. Блок, представляющий систему как единый модуль, детализируется на следующей диаграмме при помощи нескольких блоков, которые соединены интерфейсными дугами. Такие блоки определяют центральные подфункции в исходной функции. Разработку функциональных моделей удобнее производить с использованием программного средства BPWin, реализующего метод IDEF0.

Блоки на диаграмме размещаются с помощью кнопки . Для размещения стрелок используется кнопка . Для того, чтобы разбить диаграмму на блоки (провести декомпозицию) необходимо нажать правой кнопкой на нее слева в окне Activity и выбрать Decompose в контекстном меню.



Для того, чтобы разветвить стрелку, нужно нажать на кнопку со стрелкой, щелкнуть в месте стрелки, от которого начинаем ответвление, а затем на часть блока, к которому идет стрелка.

Контекстная диаграмма работы компаратора баз данных представлена на рисунке 1.2.

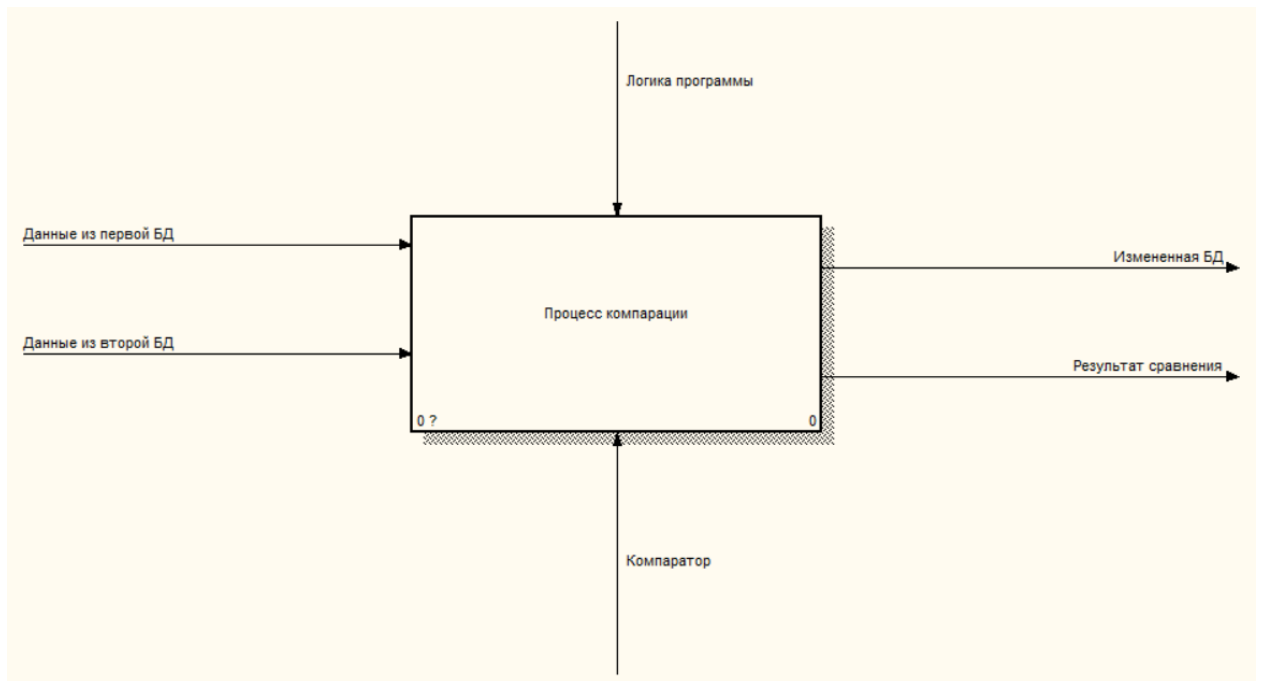


Рисунок 1.2. Контекстная диаграмма процесса компарации
Декомпозиция контекстной диаграммы показана на рисунке 1.3.

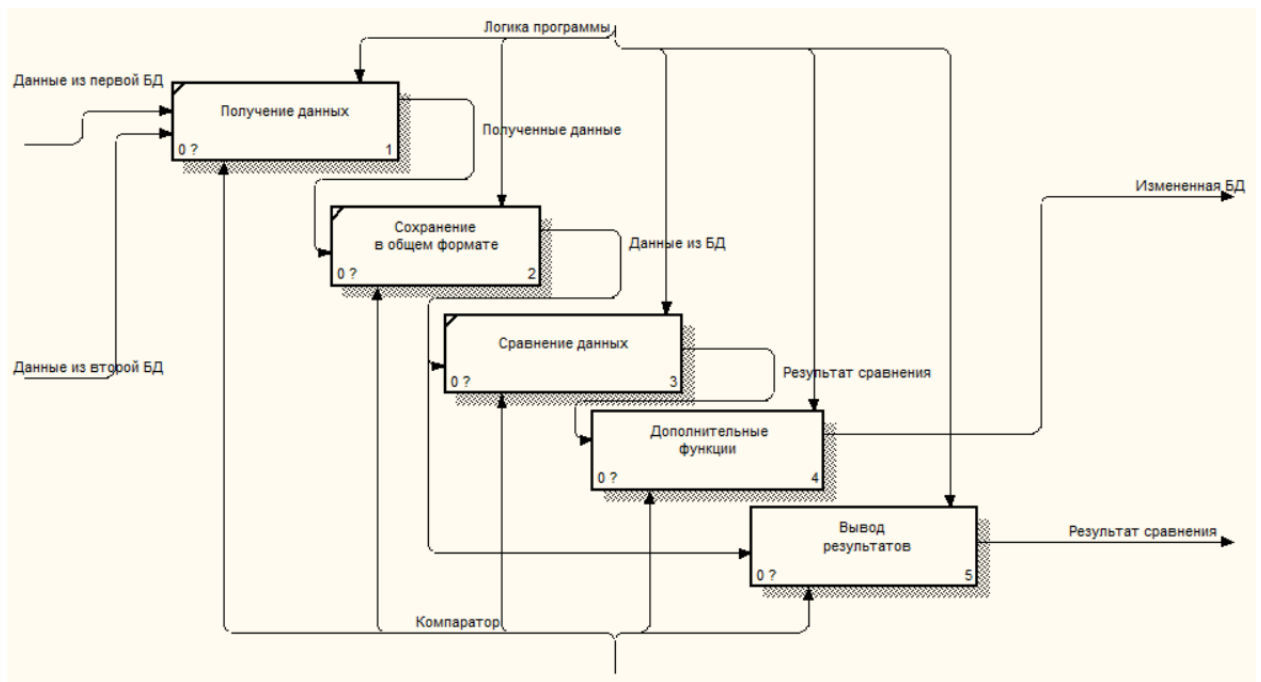


Рисунок 1.3. – Декомпозиция контекстной диаграммы

Согласно декомпозиции – сначала система получает данные из двух баз данных.

После этого данные необходимо сохранить в общем формате, привести к «общему знаменателю». Это может быть либо формат какой-либо из этих баз данных, либо сторонний формат, например XML.

После этого производится сравнение данных из обеих БД.

На заключительном этапе работы выполняются дополнительные функции, а также происходит форматирование и вывод результатов сравнения.

На рисунке 1.4. Показана декомпозиция задачи «Дополнительные функции». Согласно ей – по результатам сравнения может происходить миграция (перенос) данных из одной базы данных в другую, а также можно делать снимки схем этих баз данных, для сохранения их структуры.

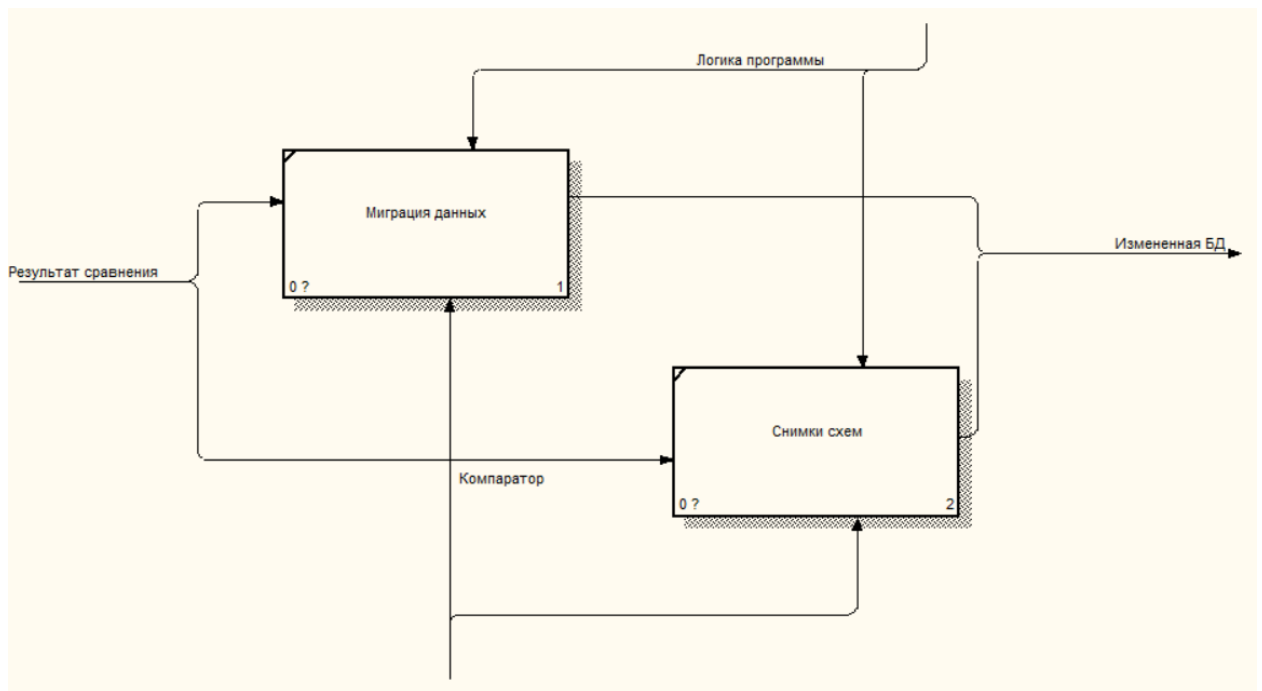


Рисунок 1.4. – Декомпозиция процесса «Дополнительные задачи».

На рисунке 1.5. Показана декомпозиция процесса «Вывод результатов».

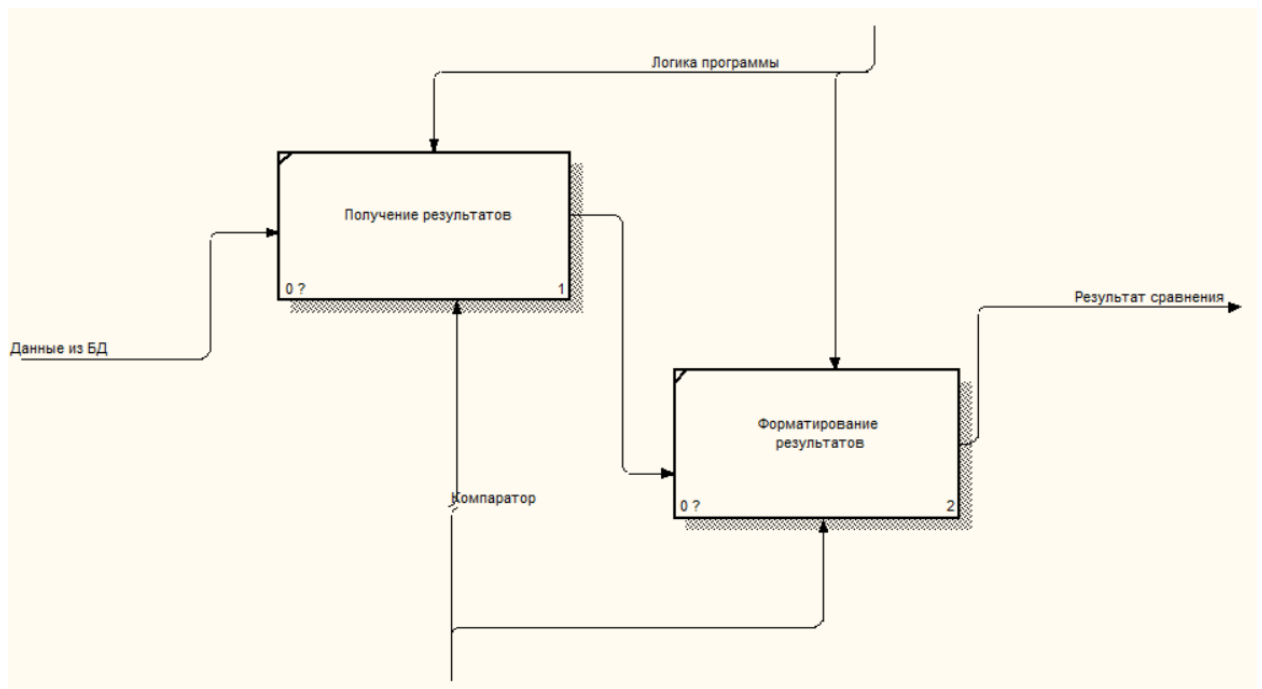


Рисунок 1.5. – Декомпозиция процесса «Вывод результатов»

1.3. Постановка задачи

1.3.1. Цель и назначение автоматизированного варианта решения задачи

Решение задачи сравнения баз данных с помощью разрабатываемого программного средства позволит упростить процесс сравнения однородных баз данных различных отделов предприятия, и объединять данные из таких баз.

Разрабатываемое решение должно обеспечивать компарацию баз SQLServer и сравнение схем этих баз данных.

1.3.2. Общая характеристика организации решения задачи на ЭВМ

Основная задача создаваемого приложения заключается в получении данных из двух похожих по структуре баз данных, сохранение этих данных и их сравнение друг с другом. Исходя из этого условно приложение возможно разделить на две части, которые занимаются следующим:

1. Получением данных из БД, их структуризацией и сохранением в файл.
2. Чтением данных и файлов в базе MySQL и предоставлением развернутого отчёта для разработчика.

Создание промежуточного этапа сохранения данных в файл и его чтения введено по той причине, что часто разработчик не имеет прямой доступ к базе данных клиента, и клиент к базе данных разработчика. Запуск первого компонента приложения необходимо осуществлять отдельно на машинах клиента и разработчика.

Сначала получают сравниваемые данные из базы данных, и они сравниваются для компарации. Данный процесс запускается для второй базы данных. После осуществляется компарация данных, которые сохранены, а также вывод результатов компарации для конечного пользователя. Прежде, чем будет запущен процесс компарации данных, эти данные должны быть

извлечены из двух файлов, которые были записаны на втором этапе. Эти четыре процесса выполняются компаратором.

Рассмотрим, по какой схеме будут передаваться данные между вышеописанными процессами.

Сперва пользователем должен быть указан критерий извлечения данных. На этапе, следующем за извлечением данных из двух баз данных, данные сохраняются. Это необходимо по следующим причинам:

1. Сохранение данных позволяет уменьшать число запросов, направляемых в базу данных при эталонном сравнении (сравнение нескольких различных наборов данных друг с другом).

2. Сохраненные данные могут передаваться на другую машину, с помощью которой произведется компарация, что может быть полезным в случаях, когда использовать данные базы данных может ограниченное число машин, доступ к этим машинам осуществляется с помощью консоли. При таком случае такие машины будут являться посредником в процессе извлечения данных.

Другая часть программы считывает сохраненные данные, после чего она сравнивает их друг с другом. Результат такого сравнения выводится на экран пользователя.

Первыми двумя хранилищами данных являются базы данных, данные которых необходимо сравнить друг с другом. Третье хранилище отводится для извлечённых данных из первых двух хранилищ.

Первоначально пользователем указывается критерии для сравнения. Он должен указать таблицы для сравнения, и также то, как эти таблицы связаны друг с другом. После этого компаратор, основываясь на полученной логике, извлечет данные первой базы данных и запишет эти данные в файл. Затем извлечет данных из второй базы и запишет их во второй файл. Это будет окончанием этапа извлечения данных.

Вторым этапом является компарация извлеченных данных. Пользователь должен выбрать два файла, которые были получены на первом

этапе. Компаратор загрузит данные из файлов снова в классы и сравнит их друг с другом согласно логике, которую на первом этапе указал пользователь. Результат сравнения данных предоставляется пользователю, затем пользователь анализирует результат компарации, и выбирает искомое различие этих в данных.

1.4. Анализ существующих разработок и обоснование выбора технологии проектирования

1.4.1. Определение критериев анализа

В качестве критериев для рассмотрения рассматриваемых ниже систем выбраны следующие:

Поддержка SQLServer. MSSQLServer является одной из самых популярных СУБД в мире на данный момент, и широко используется на рассматриваемом предприятии. Поддержка этой СУБД должна быть в рассматриваемых решениях.

Поддержка MySQL. Благодаря своей бесплатности, доступности, веб-ориентированности – MySQL получила очень широкое распространение в последнее время. Большим плюсом рассматриваемых решений будет поддержка также и этой СУБД.

Возможность сравнения с резервными копиями базы данных. Возможность сравнения резервных копий подразумевает возможность сравнения резервных копий без извлечения их из архива. Это очень важная особенность для ПО, поскольку на восстановление из резервных копий может уходить много времени и резервов.

Синхронизация двух баз данных между собой. Возможность синхронизации БД является важным параметром. Зачастую возникает необходимость перенести какие-либо данные из одной сравниваемой БД в другую, и здесь приходит на помощь синхронизация.

Восстановление данных из резервной копии. Данный параметр является не слишком важным, поскольку все СУБД поддерживают возможность восстановления. Тем не менее это полезная возможность.

Частичное восстановление данных для определённой таблицы из резервной копии может пригодиться в случае, когда необходимо сравнить только отдельные таблицы из БД.

Создание отчётов результатов компарации баз данных – это важный параметр, поскольку отчеты являются базовым документом при сравнении баз данных.

Встроенный редактор скриптов. Этот параметр не особо важен, но желателен.

Создание снимков схем. Эта возможность позволяет сделать быструю краткую резервную копию системы.

Возможность ограничить область компарации определёнными таблицами. Параметр является важным, поскольку рассматриваемые БД могут занимать значительные объёмы. И возможность выборочного сравнения может очень пригодиться для экономии времени и ресурсов.

1.4.2. Сравнительная характеристика существующих разработок

Невзирая на распространённость разнообразных компараторов баз данных, на русском языке нет научной литературы на эту тему. Исходя из этого ниже описаны существующие решения:

`dbForgeDataCompareforSQLServer` является мощным, быстрым и простым в эксплуатации инструментом сравнения и синхронизации баз данных SQL, способным использовать свои резервные копии SQLServer как источник метаданных.

Он дает возможность проводить операции:

- Сравнения данных в БД SQLServer и выявления внесённых в них изменений.
- Сравнения "рабочих" БД и резервных копий SQLServer.

- Анализа различий между двумя БД.
- Синхронизации двух рассинхронизированных БД.
- Восстановления данных конкретной таблицы с помощью резервной копии.
- Создания отчётов о сравнении данных (формат Excel и HTML).
- Копирования данных поиска из создаваемой БД в финальную.
- Автоматизации повседневных задач по синхронизации данных и интерфейса командной строки [50].

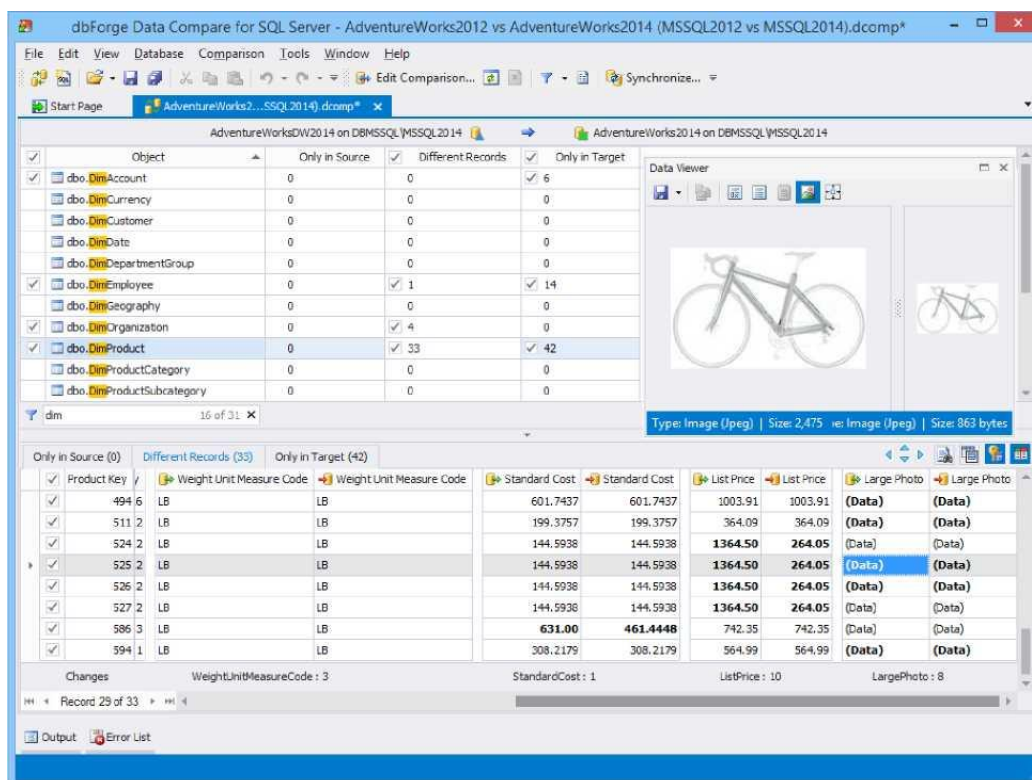


Рисунок 1.4. – dbForge Data Compare for SQL Server

ArxSQLDiff является решением, позволяющим проводить комплексное сравнение сценариев БД по большому количеству различных факторов, с включением уникальных особенностей: элементов кода, RegEx-фильтров, проектных особенностей и многого другого:

- сравнивать и синхронизировать структуры и содержимое баз данных;

- автоматизировать операции по сравнению БД при помощи интерфейса командной строки;
- использовать мастера синхронизации данных и структур;
- встраивать программы сравнения непосредственно в процесс разработки;
- создавать HTML-отчёты для данных и структур;
- осуществлять XML-экспорт данных;
- использовать возможности встроенного редактора скриптов, позволяющего просматривать уже запущенные скрипты.
- создавать снимки БД;
- использовать поддержку всех объектов SQL 2005;
- выполнять миграции из SQL 2000 в SQL 2005;
- сравнивать зашифрованные объекты баз данных SQL Server 7 и 2000 [11].

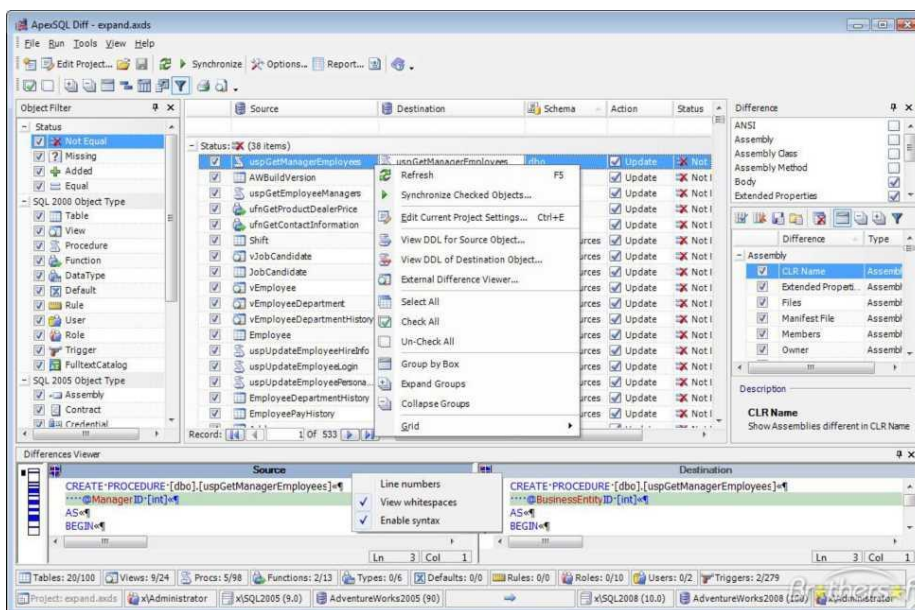


Рисунок 1.5. – ApexSQLDiff

xSQLObject является профессиональной сервисной программой по работе с SQLServer, выполняющей сравнение и синхронизацию SQLServer

баз данных. Эта программа дает возможности для SQLServer разработчиков и администраторов баз данных по слежению за синхронизированной работой их сред разработки, тестированию и производству с минимальными усилиями.

Программа отличается:

Полной поддержкой SQLServer2000 и SQLServer2005 - всех объектов.

Сравнением по версиям - сравнение баз данных SQLServer2000 и баз данных SQLServer2005 и генерированием сценариев с конкретными изменениями по версиям.

Снимками схем - поддержкой истории изменения схем и предоставлением безопасного и эффективного анализа схем, если это требуется.

Широким спектром опций, обеспечивающим частичный контроль поведения системы сравнения.

Расширенной фильтрацией объектов - сравнением и синхронизацией только необходимых объектов.

Сравнением результатов, отображаемых удобной решёткой, позволяющей пользователю не тратить время на подробности, а быстро работать со сценариями объектов, результатами фильтрации и т.д.

Генерированием безопасных сценариев изменений обзора прежде, чем они будут исполняться в целевой базе.

Использованием утилиты командной строки, отличающейся возможностями по сравнению и синхронизации по расписанию.

Расширенными функциями создания сценариев - создание сценария по определенному объекту или же по всей базе данных, сценария схемы или же одновременно по схемам и данным [4].

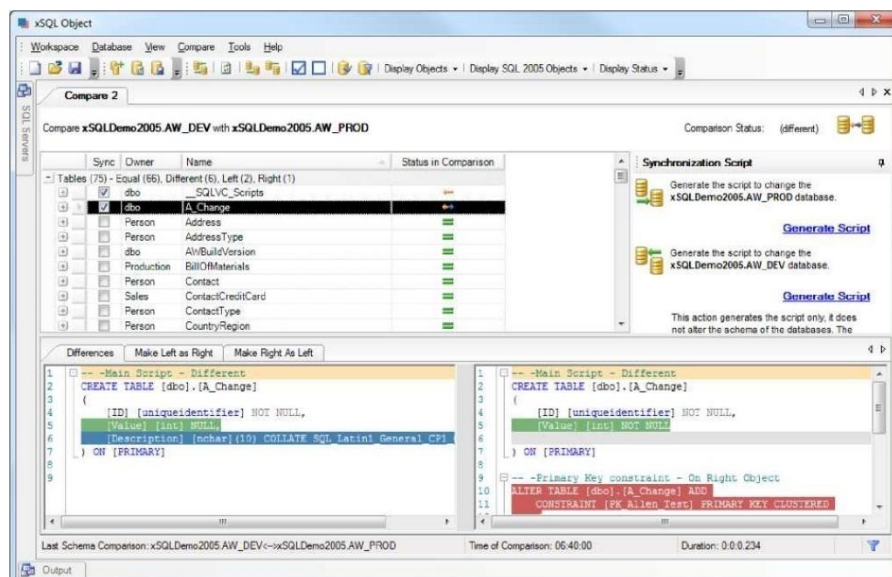


Рисунок 1.6. – xSQLObject

Теперь, после перечисления основных особенностей имеющихся компараторов баз данных, произведем сравнение из друг с другом по ключевым критериям. Сравнение представлено в виде таблицы, где по строкам располагаются ключевые особенности, а в столбцах находятся компараторы.

Таблица 1.1. – Сравнение существующих компараторов

Критерий	dbForge Data Compare for SQL Server	ApexSQL Diff	xSQL Object
Поддержка SQL Server	Есть	Есть	Есть
Поддержка MySQL	Есть	Есть	Нет
Возможность сравнения с резервными копиями базы данных	Есть	Нет	Нет
Синхронизация двух баз данных между собой	Есть	Есть	Есть
Восстановление данных из резервной копии	Есть	Нет	Нет
Частичное восстановление данных для определённой	Есть	Нет	Нет

Создание отчётов результатов компарации	Есть	Есть	Нет
Встроенный редактор скриптов	Нет	Есть	Нет
Создание снимков схем	Нет	Есть	Есть
Возможность ограничить область компарации	Нет	Нет	Есть

Исходя из данных вышеописанной таблицы, можно увидеть, что каждый компаратор имеет свои минусы. В основном предназначение всех рассмотренных компараторов – синхронизация данных в различных версиях баз данных, и восстановление при утере. Однако нам необходимо «вытащить» из базы данных данные, которые логически связаны с необходимым критерием. Поэтому данные приложения не подходят для использования.

С другой стороны, привязыванием компаратора с логикой информационной системы можно получить невозможность применения этого компаратора к другим базам данных, таким образом, чтобы не изменять логики самого приложения.

1.5. Выводы

В процессе выполнения первой главы работы были достигнуты следующие результаты:

1. Проведено обследование предприятия и его деятельности, дана краткая характеристика, для которого разрабатывается система;
2. Проведено концептуальное моделирование разрабатываемой системы;
3. Описаны цели и задачи создания системы, дана общая характеристика способов решения задачи;

4. Выбраны критерии для сравнения аналогичных систем и проведено их сравнение.

Несомненно, разработка программной системы для сравнения баз данных является достаточно сложной задачей. В следующей главе будет проведена практическая реализация описанной системы.

ГЛАВА 2. РАЗРАБОТКА И РЕАЛИЗАЦИЯ ПРОЕКТНЫХ РЕШЕНИЙ

2.1. Логическое моделирование предметной области

2.1.1. Логическая модель и ее описание

Преобразуем IDEF0 модель, приведенную в первой главе, в диаграмму UseCase.

Диаграмма вариантов деятельности для разрабатываемого приложения выглядит следующим образом.

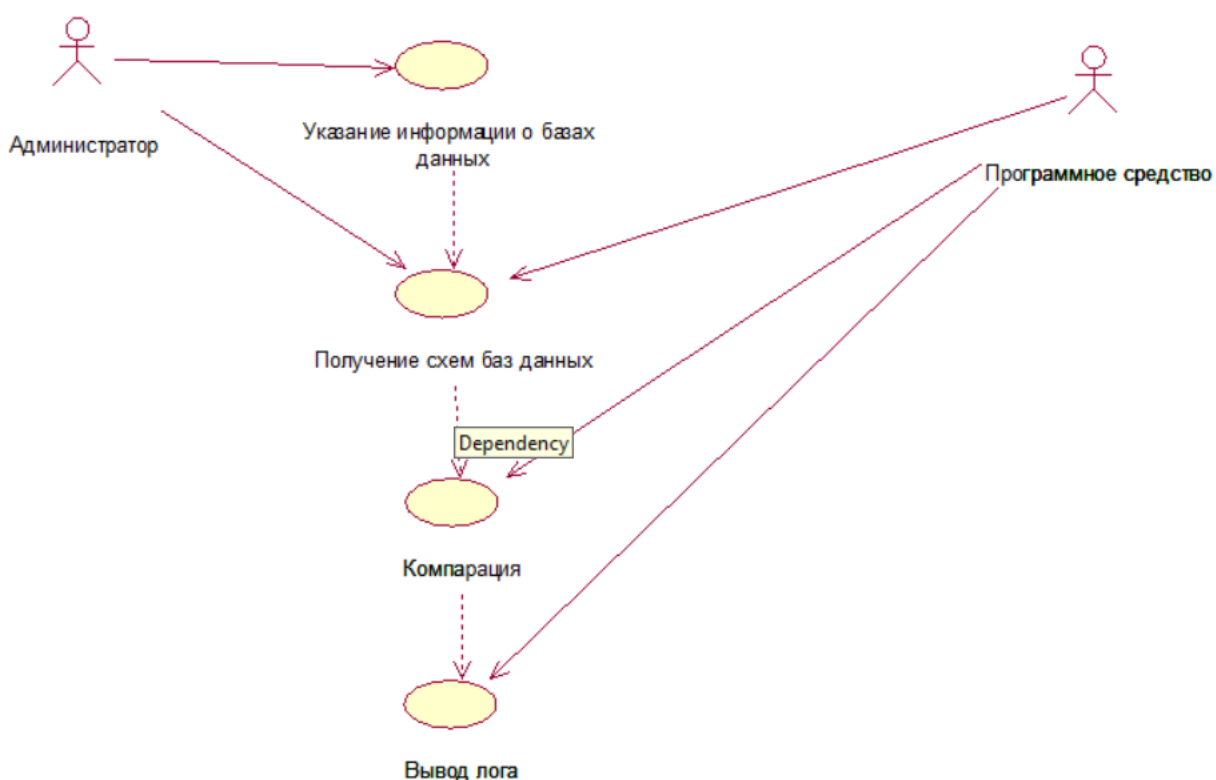


Рисунок 2.1. – Диаграмма вариантов деятельности

2.1.2. Используемые классификаторы и системы кодирования

Дополнительных классификаторов при разработке системы не использовалось.

2.1.3. Характеристика нормативно-справочной и входной оперативной информации

Входная информация в разрабатываемой системе – это две базы данных MSSQLServer и данные доступа к ним.

Промежуточной информацией являются файлы схем данных.

Использование схем в модели безопасности компонента DatabaseEngine упрощает взаимоотношения пользователей и объектов, из чего следует, что схемы имеют значительное влияние на процесс взаимодействия пользователя и компонента DatabaseEngine. Ниже рассмотрена значимость схем для безопасности компонента DatabaseEngine. Первый подраздел описывает взаимодействие между схемами и пользователями, второй описывает все три инструкции для языка Transact-SQL, которые применяются с целью создания и модификации рассматриваемых схем.

Схема является коллекцией объектов базы данных, имеющей одного владельца и формирующей одно пространство имен (двумя таблицам в одной и той же схеме не может даваться одинаковое имя.) Компонентом DatabaseEngine поддерживаются именованные схемы при использовании понятия принципала (principal). Принципал может быть индивидуальным принципалом и групповым принципалом.

Индивидуальным принципалом представляется один пользователь, например, как регистрационное имя или учетная запись пользователя Windows. Групповой принципал может быть группой пользователей, например, ролью или группой Windows. Принципалы обладают схемами, но обладание схемой может быть свободно отдано другому принципалу, при этом имя схемы не изменится.

Разграничение пользователей базы данных и схем имеет некоторые преимущества:

- один принципал может владеть несколькими схемами;

- несколько индивидуальных принципалов имеют возможность владеть одной схемой через членство в ролях или группах Windows;

- при удалении пользователя базы данных не требуется переименование объектов, которые содержатся в схеме данного пользователя.

По умолчанию у каждой базы данных есть схема, используемая для того, чтобы определять имена объектов, ссылки на такие объекты создаются без уточнения полных имен. В схеме по умолчанию есть название первой схемы, в которой сервер базы данных в последствии будет выполнять поиск с целью разрешения имен объектов. Настройка и изменение схемы по умолчанию осуществляется с помощью параметра `DEFAULT_SCHEMA` инструкции `CREATE USER` или же `ALTER USER`. В случае, когда схема по умолчанию `DEFAULT_SCHEMA` не является определенной, по умолчанию как схеме пользователю базы данных выделяется схема `dbo`.

Выходной информацией является лог, отображенный как описание различий двух баз данных.

2.1.4. Характеристика базы данных

Опишем словесно сущности, которые должны быть представлены в базе данных, которая используется для тестирования системы.

Контрагенты проводят операции с ценными бумагами, которые отражаются в реестре операций.

Операции проводятся на счетах.

Каждая ценная бумага относится к определенной категории, виду, имеет определенную форму выпуска.

Все операции относятся к определенному виду.

Таким образом, сущности, которые необходимо реализовать в базе данных:

- Контрагенты;

- Ценные бумаги;
- Реестр операций;
- Счета;
- Категории ценных бумаг;
- Виды ценных бумаг;
- Формы выпуска ценных бумаг;
- Виды операций.

К справочным сущностям (справочникам) в данном списке относятся следующие:

- Категории ценных бумаг;
- Виды ценных бумаг;
- Формы выпуска ценных бумаг;
- Виды операций.

Построим концептуальную схему БД на основе этих данных.

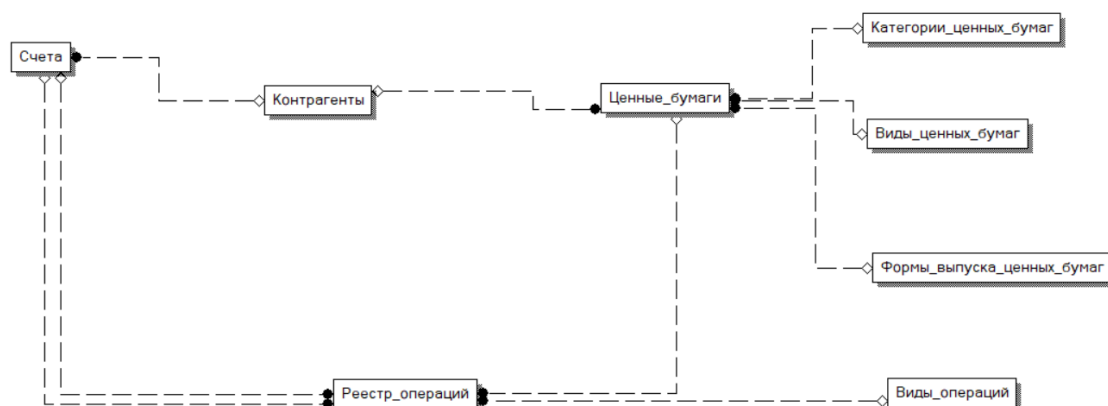


Рис. 2.2. – Схема взаимосвязи сущностей базы данных

Путем добавления атрибутов к сущностям концептуальной модели была получена логическая модель базы данных.

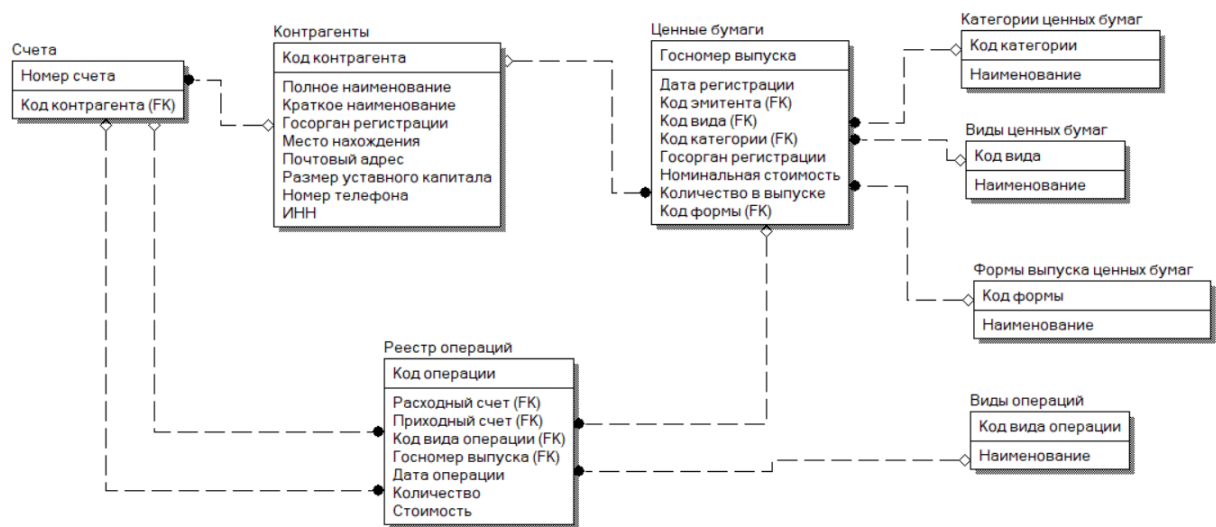


Рис. 2.3. – Логическая схема базы данных

Далее, путем добавления типов данных к атрибутам логической модели – была получена физическая схема базы данных.

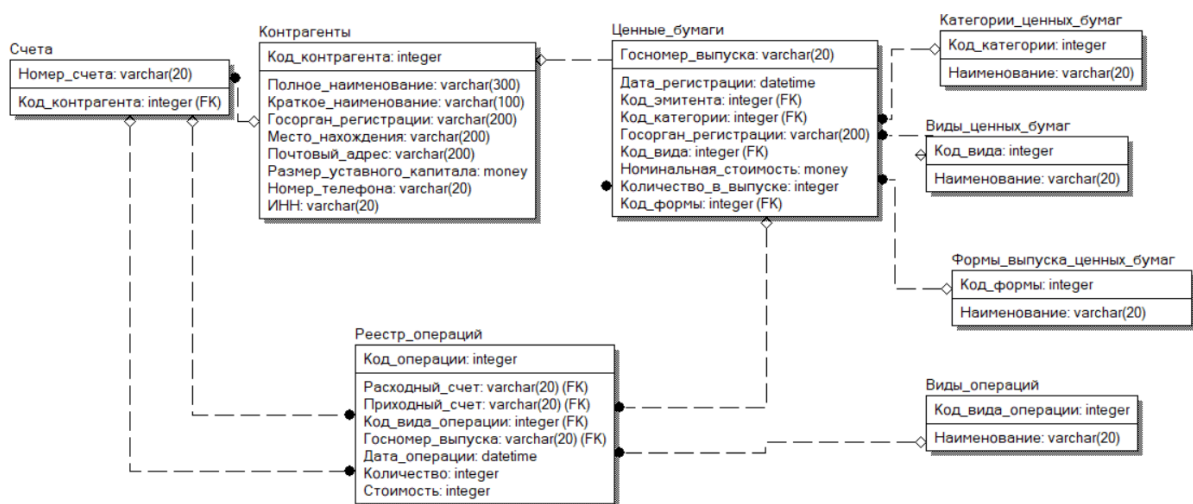


Рис. 2.4. – Физическая схема базы данных

После этого, в CASE-средстве ERWin был получен сценарий создания базы данных, который показан в ПРИЛОЖЕНИИ 2.

2.1.5. Характеристика результатной информации

Результатная информация в системе представляет собой сообщения, выдаваемые пользователю системы.

В системе предусмотрено три вида сообщений:

«Пожалуйста получите информацию по обеим базам данных» – в случае если не получена схема баз данных перед запуском процедуры компарации.

«Необходимо создать таблицу» <Запрос для создания>

«Модифицируйте таблицу» <Запрос для модификации>

«Измените таблицу» <Запрос для изменения>

2.2. Физическое моделирование АИС

2.2.1. Выбор архитектуры АИС

Клиентский компьютер и серверный компьютер могут работать вместе для предоставления пользователям прикладного программного обеспечения. Прикладная архитектура — это способ, которым функции программного обеспечения прикладного уровня распределяются между клиентами и серверами в сети.

Работу, выполняемую любой прикладной программой, можно разделить на четыре основные функции. Первое — это хранение данных. Большинство прикладных программ требуют хранения и извлечения данных, будь то небольшой файл, такой как заметка, созданная текстовым процессором, или большая база данных, такая как бухгалтерские записи организации. Вторая функция — это логика доступа к данным, обработка, необходимая для доступа к данным, что часто означает запросы к базе данных в SQL (язык структурированных запросов). Третья функция - логика приложения (иногда называемая бизнес-логикой), которая также может быть простой или сложной, в зависимости от приложения. Четвертая функция - логика представления; представление информации пользователю и принятие пользовательских команд. Эти четыре функции, хранение данных, логика доступа к данным, логика приложения и логика представления, являются основными строительными блоками любого приложения.

Есть много способов, которыми эти четыре функции могут быть распределены между клиентскими компьютерами и серверами в сети. В настоящее время используются четыре основные архитектуры приложений. В архитектуре на основе хоста сервер (или хост-компьютер) выполняет практически всю работу. В клиентских архитектурах клиентские компьютеры выполняют большую часть работы. В клиент-серверных архитектурах работа распределяется между серверами и клиентами. В одноранговых архитектурах компьютеры являются как клиентами, так и серверами и, таким образом, совместно используют работу. Клиент-серверная архитектура является доминирующей архитектурой приложений.

Существует много разных типов клиентов и серверов, которые могут быть частью сети, и различия между ними со временем становятся немного сложнее. Вообще говоря, есть четыре типа компьютеров, которые обычно используются в качестве серверов:

- Мэйнфрейм — это большой компьютер общего назначения (обычно с высокой стоимостью), который способен выполнять много одновременных функций, поддерживать много одновременных пользователей и хранить огромные объемы данных;

- Микрокомпьютер — это тип используемого нами компьютера. Микрокомпьютеры, используемые в качестве серверов, могут варьироваться от небольшого микрокомпьютера, аналогичного настольному, который мы можем использовать, до более дорогого;

- Кластер — это группа компьютеров (часто микрокомпьютеров), связанных между собой так, что они действуют как один компьютер. Запросы поступают в кластер (например, веб-запросы) и распределяются между компьютерами, чтобы ни один компьютер не был перегружен. Каждый компьютер является отдельным, поэтому в случае сбоя кластер просто обходит его. Кластеры сложнее, чем отдельные серверы, потому что работа должна быть быстро скоординирована и распределена между

отдельными компьютерами. Кластеры очень масштабируемы, потому что всегда можно добавить еще один компьютер в кластер;

- Виртуальный сервер — это один компьютер (часто микрокомпьютер), который действует как несколько серверов. Используя специальное программное обеспечение, несколько операционных систем устанавливаются на одном физическом компьютере, поэтому один физический компьютер отображается в сети как несколько разных серверов. Эти виртуальные серверы могут выполнять одинаковые или отдельные функции (например, сервер печати, веб-сервер, файловый сервер). Это повышает эффективность (когда каждый сервер используется не полностью, нет необходимости покупать отдельные физические компьютеры) и может повысить эффективность (если происходит сбой одного виртуального сервера, это не приводит к сбою других серверов на том же компьютере).

Существует пять часто используемых типов клиентов:

- Микрокомпьютеры является наиболее распространенным типом клиентов сегодня. К ним относятся настольные и портативные компьютеры, а также планшетные ПК, которые позволяют пользователю писать инструментом, похожим на ручку, а не печатать на клавиатуре;

- Терминал — это устройство с монитором и клавиатурой, но без центрального процессора (ЦП). «Глупые» терминалы, названные так потому, что они не участвуют в обработке отображаемых данных, имеют минимум, необходимый для работы в качестве устройств ввода и вывода (экран и клавиатура). В большинстве случаев, когда символ набирается на немом терминале, он передает символ через схему на сервер для обработки. Каждое нажатие клавиши обрабатывается сервером, даже такие простые действия, как стрелка вверх;

- Сетевой компьютер предназначен в первую очередь для связи с использованием интернет-стандартов (например, HTTP, Java), но не имеет жесткого диска. Он имеет только ограниченную функциональность;

- Терминал транзакций предназначен для поддержки определенных бизнес-операций, таких как банкоматы, используемые банками. Другими примерами транзакционных терминалов являются терминалы торговых точек в супермаркете;

- Карманный компьютер, Personal Digital Assistant или мобильный телефон также можно использовать в качестве сетевого клиента.

Для решения проблемы сохранения целостности существуют следующие архитектуры.

Архитектура хост/терминал.

Самые первые сети передачи данных, разработанные в 1960-х годах, были основаны на хостах, а сервер (обычно это большой мэйнфрейм) выполнял все функции. Клиенты (обычно терминалы) позволяли пользователям отправлять и получать сообщения с хост-компьютера. Клиенты просто фиксировали нажатия клавиш, отправляли их на сервер для обработки и принимали от сервера инструкции о том, что отображать.

Эта простая архитектура часто работает хорошо. Прикладное программное обеспечение разрабатывается и хранится на одном сервере вместе со всеми данными. Существует одна точка контроля, потому что все сообщения проходят через один центральный сервер. Теоретически, существует эффект масштаба, поскольку все компьютерные ресурсы централизованы.

Есть две фундаментальные проблемы с сетями на основе хоста. Во-первых, сервер должен обработать все сообщения. По мере того, как растет спрос на сетевые приложения, многие серверы перегружаются и не могут быстро обработать все запросы пользователей. Приоритизация доступа пользователей становится затруднительной. Время отклика становится медленнее, и сетевым менеджерам приходится тратить больше денег на модернизацию сервера. К сожалению, обновления до мэйнфреймов, которые обычно являются серверами в этой архитектуре, являются «кусковыми». То есть обновления идут большими шагами и стоят дорого.

В отношениях терминал / хост обработка работы происходит на хосте, а отображение управляется терминалом. Терминальное устройство не выполняет обработку приложений; оно просто отображает экран с хоста и принимает нажатия клавиш от пользователя для отправки в приложение, работающее на хост-компьютере.

В этом случае текстовый процессор на основе хоста переформатирует абзац (в памяти на хост-машине) и перерисует видеоизображение на терминальном устройстве. Приложение базы данных работает на хосте, и между двумя компьютерами происходит обмен только символами текста.

Преобладающая информация, которой обмениваются коммутаторы в разговоре терминал / хост, символично-ориентирована. Обработка выполняется на хосте, символы вводятся и отображаются на терминале.

Терминал / хост – это символично-ориентированный обмен информацией.

Управление экранами приложений — это другой тип клиент / сервера. Предположим, что между клиентским приложением и компонентом сервера установлен диалог клиент / сервер. В этом случае сервер собирается выполнить обработку видеоизображения. «Команды», отправляемые на сервер, будут выглядеть как «Переместить курсор вверх на одну строку» или, возможно, «Поместить курсор в строку 12, столбец 18». Сервер хранит виртуальную копию «экрана» в памяти, чтобы ему было известно, что команда «Переместить курсор вниз на одну строку» получена, когда курсор находится в строке 12, а в столбце 18 курсор теперь находится в строке 13 (одна строка вниз). Работа, выполняемая этим «сервером», является управлением экраном прикладной программы.

Компонент «клиент» в этом примере просто отвечает за отображение экрана (которым управляет «сервер»). Это действительно делает упрощенную «клиентскую» программу. По сути, это терминальное устройство. Прикладная программа (текстовый процессор или база данных) работает на том же компьютере, что и программа-сервер для экрана.

«Клиент» — это просто терминальная программа. Сервер Telnet обслуживает клиенту один экран. Программное обеспечение сервера Telnet находится на главном компьютере вместе с прикладной программой. Клиент Telnet находится на компьютере пользователя для отображения экрана и приема символов с клавиатуры.

Архитектура подключения Telnet Client. Так работают программы Telnet и RLOGIN в мире протоколов TCP / IP. Многие UNIX-системы используют связь клиент / сервер Telnet и хост / терминал. Telnet реализует диалог между терминалами.

Преобладающая информация, которой обмениваются в разговоре Telnet, символьно-ориентирована. В начале диалога клиент и сервер обмениваются некоторыми командами настройки. Диалог представляет собой взаимодействие между терминалами и хостами, но на уровне программы организовано само соединение между процессом клиента и сервера. Только программист, пишущий код Telnet (или RLOGIN, или другой подобный тип), должен знать об этой взаимосвязи. Для обычных пользователей Telnet — это простой разговор между хостом и терминалом.

Преимущества: дешевые терминалы, невысокая загрузка сети (трафик), эффективнее решается проблема целостности.

Недостатки: мощность ограничена хостом, если в качестве терминала используются персональные компьютеры, то их ресурсы не используются.

Данная архитектура применима для больших и очень больших корпоративных сетей, построенных, как правило, на базе ОС Unix

Клиентские архитектуры

В конце 1980-х гг. произошел взрыв использования микрокомпьютеров и локальных сетей на основе микрокомпьютеров. Сегодня более 90 процентов общей вычислительной мощности компьютеров большинства организаций в настоящее время находится в локальных сетях на основе микрокомпьютеров, а не в централизованных компьютерах мэйнфреймов. Часть этого расширения была вызвана рядом недорогих, очень популярных

приложений, таких как текстовые процессоры, электронные таблицы и графические программы для презентаций. Это также было вызвано недовольством менеджеров прикладным программным обеспечением на основных компьютерах. Большая часть программного обеспечения для мэйнфреймов не так проста в использовании, как программное обеспечение для микрокомпьютеров, намного дороже, и на ее разработку могут уйти годы. В конце 1980-х гг. многие крупные организации имели задержки в разработке приложений от двух до трех лет; то есть на создание любой новой прикладной программы для мэйнфрейма уходили годы. В противовес к этому, менеджеры могут купить микрокомпьютерные пакеты или разработать приложения на основе микрокомпьютера за несколько месяцев.

В клиентских архитектурах клиенты являются микрокомпьютерами в локальной сети, а сервер обычно является другим микрокомпьютером в той же сети. Прикладное программное обеспечение на клиентских компьютерах отвечает за логику представления, логику приложения и логику доступа к данным; сервер просто хранит данные.

Эта простая архитектура часто работает очень хорошо. Если вы когда-либо использовали текстовый процессор и хранили свой файл документа на сервере (или писали программу на Visual Basic или C, которая работает на вашем компьютере, но хранит данные на сервере), вы использовали клиентскую архитектуру.

Основная проблема в клиентских сетях состоит в том, что все данные на сервере должны передаваться клиенту для обработки. Например, предположим, что пользователь желает отобразить список всех сотрудников со страховкой компании. Все данные в базе данных (или все индексы) должны передаваться с сервера, где база данных хранится по сетевому каналу, клиенту, который затем проверяет каждую запись, чтобы определить, соответствует ли она данным, запрошенным пользователем. Это может привести к перегрузке сетевых каналов, поскольку с сервера клиенту передается гораздо больше данных, чем клиенту на самом деле нужно.

Архитектура файл/сервер.

Архитектура файл-сервер является обычным продуктом системы баз данных, которая использует архитектуру компьютеров, а затем расширяется для размещения нескольких одновременно работающих пользователей. Она характеризуется разделением СУБД от данных сети. Переход от архитектуры компьютера к архитектуре файл-сервер, как правило, очень прост и не требует специальных знаний базы данных. Тем не менее, это вводит существенное ограничение производительности.

СУБД отвечает за управление каждым аспектом данных. В этой архитектуре СУБД должна манипулировать данными в сети. Поскольку скорости передачи сетевых данных существенно ниже, чем скорости передачи данных на локальном диске, отделение СУБД от данных создает существенное «узкое место». Можно представить, что происходит, когда СУБД необходимо ответить на информационный запрос, требующий проверки каждой записи в списке клиентов. Предположим, что нужно изучить миллион записей клиентов, и только 100 из них соответствуют критериям, указанным в информационном запросе. В архитектуре мэйнфреймов, где СУБД имеет прямой доступ к данным, она может быстро прочитать все записи и отправить только 100 соответствующих записей по сети. Однако в архитектуре файл-сервера все миллионы записей должны быть отправлены по сети для оценки СУБД. СУБД проверяет данные по мере их поступления, а затем сохраняет только соответствующие данные для отображения пользователю.

Снижение производительности не ограничивается только одним пользователем приложения базы данных. Поскольку сеть становится узким местом в этом процессе, все пользователи, которые совместно используют сегменты сети с пользователем приложения базы данных, также будут испытывать сетевые задержки. Более того, если другие пользователи, которые совместно используют сегменты сети с пользователем приложения

базы данных, вовлечены в интенсивное использование сети, производительность приложения базы данных будет еще ниже.

Архитектура файл-сервер представляет другую проблему по сравнению с архитектурой мэйнфрейма. В этой архитектуре есть одна СУБД, которая управляет хранилищем данных. Однако в архитектуре файл-сервер каждый дополнительный пользователь добавляет еще один экземпляр СУБД для управления данными. Чтобы это работало, СУБД должны координировать действия друг с другом для обеспечения целостности и безопасности данных. Например, если один пользователь в настоящее время редактирует информацию о конкретном продукте, экземпляр СУБД, который использует пользователь, должен предупредить другие экземпляры, дабы не допустить, чтобы эта же информация была отредактирована другим пользователем до завершения. Этот процесс обычно работает достаточно хорошо; однако проблемы возникают, когда экземпляр СУБД не может предупредить другие экземпляры о том, что его пользователь завершил работу с конкретным ресурсом данных, из-за проблем с сетевым подключением или других проблем операционной системы.

Преимущество этой архитектуры состоит в том, что при очень небольшом знании баз данных пользователь может настроить приложение базы данных для одновременного использования несколькими пользователями. Если система не управляет большим объемом данных или если пользователям редко приходится отвечать на сложные вопросы о данных, это недорогой путь к многопользовательскому параллелизму.

Преимущества архитектуры файл-сервер:

- Низкая цена;
- Проста в настройке;
- Недорогой вариант для многопользовательского параллелизма.

Недостатки архитектуры файл-сервер:

- Жизнеспособна только для ограниченного числа пользователей, возможно, 10 или меньше;

- Производительность запросов быстро снижается по мере увеличения объема данных;
- Многократная установка как прикладного программного обеспечения, так и СУБД может привести к проблемам управления версиями.

Данная архитектура применяется в простых коробчатых вариантах программного обеспечения (например, 1:С).

Наиболее популярными СУБД, которые применяют данную архитектуру, являются такие продукты, как FoxPro, DBase, Paradox, Access.

Архитектура клиент/сервер.

Серверы — это мощные компьютеры или процессы, предназначенные для управления файлами (файловые серверы), принтерами (серверы печати) или сетевым трафиком (сетевые серверы). Клиентами являются ПК или рабочие станции, на которых пользователи запускают приложения. Клиенты полагаются на серверы для получения ресурсов, таких как файлы, устройства и вычислительная мощность. Клиент-серверная архитектура, архитектура компьютерной сети, в которой многие клиенты (удаленные процессоры) запрашивают и получают обслуживание от централизованного сервера (хост-компьютера).

Клиент-серверная архитектура — это система с общей архитектурой, в которой нагрузки клиент-сервер разделены. Клиент-серверная архитектура представляет собой централизованную систему ресурсов, где сервер содержит все ресурсы. Сервер получает многочисленные возможности для совместного использования ресурсов для своих клиентов по запросу. Сервер чрезвычайно стабилен и масштабируем для того, чтобы возвращать ответы клиентам. Эта архитектура ориентирована на обслуживание, это означает, что обслуживание клиента не будет прервано в экстренных случаях. Клиент-серверная архитектура подавляет сетевой трафик, отвечая на запросы клиентов, а не на полную передачу файлов, восстанавливает файловый сервер с сервером базы данных.

Клиентские компьютеры реализуют связь, позволяющую пользователю компьютера запрашивать услуги сервера и получать результаты, которые возвращает сервер. Серверы ждут запросов от клиентов и затем возвращают их. Сервер обычно предоставляет стандартизированный простой интерфейс для клиентов, чтобы избежать путаницы аппаратного / программного обеспечения. Клиенты расположены на рабочих местах или на персональных компьютерах, при этом серверы будут находиться где-то в сети. Эта архитектура полезна, главным образом, когда клиенты и сервер имеют отдельные задачи, которые они выполняют. Многие клиенты могут получать информацию о сервере одновременно, в то время как клиентский компьютер может выполнять другие задачи.

Типы клиент-серверной архитектуры

1-уровневая архитектура. В этой категории настройки клиент-сервер пользовательский интерфейс, маркетинговая логика и логика данных присутствуют в одной системе. Этот вид услуг является разумным, но им трудно управлять из-за дисперсии данных, которая позволяет реплицировать работу. Одноуровневая архитектура состоит из слоев.

Например, уровни представления, бизнеса и доступа к данным в одном программном пакете. Данные обычно хранятся в локальной системе или на общем диске. Приложения, которые обрабатывают все три уровня, такие как MP3-плеер и MS Office, входят в одноуровневое приложение.

2-х уровневая архитектура. В этом типе клиент-серверной среды пользовательский интерфейс хранится на клиентском компьютере, а база данных хранится на сервере. Логика базы данных и бизнес-логика хранятся либо на клиенте, либо на сервере, но ее необходимо поддерживать. Если бизнес-логика и логика данных собираются на стороне клиента, это называется архитектурой тонкого сервера толстого клиента. Если бизнес-логика и логика данных обрабатываются на сервере, это называется архитектурой толстого сервера тонкого клиента.

В двухуровневой архитектуре клиент и сервер должны входить в прямую интеграцию. Если клиент вводит данные на сервер, промежуточного звена быть не должно. Это сделано для быстрых результатов и во избежание путаницы между разными клиентами. Например, программное обеспечение для онлайн-бронирования билетов использует эту двухуровневую архитектуру.

3-х уровневая архитектура. В этой архитектуре используется дополнительное промежуточное программное обеспечение, которое означает, что клиентский запрос направляется на сервер через промежуточный уровень, и ответ сервера сначала получает промежуточное программное обеспечение, а затем клиент. Эта архитектура защищает двухуровневую архитектуру и обеспечивает наилучшую производительность. Эта система имеет высокую цену, но проста в использовании. Промежуточное программное обеспечение хранит всю бизнес-логику и логику передачи данных. Идея промежуточного программного обеспечения заключается в том, чтобы организовать базу данных, поставить в очередь, выполнить приложение, составить расписание и т. д. Промежуточное программное обеспечение повышает гибкость и обеспечивает наилучшую производительность.

Трёхуровневая архитектура разделена на 3 части: уровень представления (уровень клиента), уровень приложения (уровень бизнеса) и уровень базы данных (уровень данных). Клиентская система управляет уровнем презентации; сервер приложений заботится о уровне приложений, а система сервера контролирует уровень базы данных.

Особенности архитектуры клиент-сервер:

- Клиент / сервер использует сетевые ресурсы более эффективно, чтобы уменьшить сетевой трафик и отвечать на запросы, сохраняя при этом безопасность данных;
- Архитектура клиент / сервер эффективно разделяет функции СУБД между клиентом и сервером;

- Сервер действует как ядро базы данных, поддерживающее и администрирующее доступ к данным;
- Когда сервер получает запрос от клиента, он сканирует базу данных на наличие данных на сервере и возвращает только запрошенный фрагмент, а не целый блок;
- Обработка происходит на сервере, поэтому сервер отправляет данные в окончательной форме в качестве ответа на запрос;
- Язык структурированных запросов (SQL) — это инструмент, позволяющий пользовательскому интерфейсу приложения взаимодействовать с ядром базы данных сервера.

Спрос на быстрые ответы и качественные услуги растет. Поэтому сложная клиентская архитектура имеет решающее значение для деловой активности. Компании обычно изучают возможности для поддержания качества обслуживания и соответствия своему рынку с помощью клиент-серверной архитектуры. Архитектура повышает производительность благодаря практике экономичных пользовательских интерфейсов, улучшенному хранению данных, расширенным возможностям подключения и безопасным услугам.

Преимущества: использование ресурса, как клиента, так и сервера.

Недостатки: при очень большой загрузке (при высоком трафике) падает производительность.

Данная архитектура применяется в масштабе предприятия.

В настоящее время на рынке имеются огромное количество СУБД, применяющих данную архитектуру. Перечислим наиболее популярные: ORACLE, SyBase, MS SQL, InterBase.

Выбор архитектуры

Каждая из описанных архитектур имеет определенные недостатки и преимущества. Во многих случаях архитектура просто дана; организация имеет определенную архитектуру, и ее просто нужно использовать.

В других случаях организация приобретает новое оборудование и пишет новое программное обеспечение и имеет возможность разработать новую архитектуру, по крайней мере, в некоторой части организации. Существует как минимум три основных набора факторов, которые необходимо учитывать

Стоимость инфраструктуры. Одной из самых сильных сторон, побуждающих компании к архитектуре клиент-сервер, является стоимость инфраструктуры (аппаратное и программное обеспечение и сети, которые будут поддерживать систему приложений). Проще говоря, микрокомпьютеры более чем в 1000 раз дешевле мэйнфреймов при той же вычислительной мощности. Микрокомпьютеры на наших рабочих столах сегодня имеют больше вычислительной мощности, памяти и места на жестком диске, чем мэйнфреймы начала 1990-х гг., и стоимость микрокомпьютеров составляет лишь часть стоимости мэйнфрейма. Поэтому стоимость клиент-серверных архитектур ниже, чем у серверных архитектур, которые используют мэйнфреймы. Клиент-серверные архитектуры также, как правило, дешевле, чем клиентские архитектуры, поскольку они создают меньшую нагрузку на сети и, следовательно, требуют меньшей пропускной способности сети.

Стоимость разработки. Стоимость разработки прикладных систем является важным фактором при рассмотрении финансовых преимуществ клиент-серверных архитектур. Разработка прикладного программного обеспечения для клиент-серверных архитектур может быть сложной. Разработка прикладного программного обеспечения для хост-архитектур обычно обходится дешевле. Разница в стоимости может измениться по мере того, как компании приобретают опыт работы с клиент-серверными приложениями, по мере разработки и совершенствования новых клиент-серверных продуктов и по мере развития стандартов клиент-сервер. Однако, учитывая внутреннюю сложность клиент-серверного программного обеспечения и необходимость координировать взаимодействие

программного обеспечения на разных компьютерах, вероятно, сохранится разница в стоимости.

Даже обновление сети с новой версией программного обеспечения является более сложным. В сети на основе хоста есть одно место, в котором хранится прикладное программное обеспечение; чтобы обновить программное обеспечение, просто заменяют его там. В клиент-серверных сетях необходимо обновить все клиенты и все серверы. Например, предположим, что вы хотите добавить новый сервер и переместить некоторые существующие приложения со старого сервера на новый. Все прикладное программное обеспечение на всех толстых клиентах, которые отправляют сообщения в приложение на старом сервере, теперь должно быть изменено для отправки на новый сервер. Хотя это не является концептуально сложным, это может привести к административному хаосу.

Масштабируемость. Масштабируемость означает способность увеличивать или уменьшать емкость вычислительной инфраструктуры в ответ на изменяющиеся потребности в емкости. Наиболее масштабируемая архитектура — это клиент-серверные вычисления, поскольку серверы могут быть добавлены (или удалены) к архитектуре, когда требуется изменить обработку. Например, в четырехуровневой архитектуре клиент-сервер можно иметь десять веб-серверов, четыре сервера приложений и три сервера базы данных. Если серверы приложений начинают перегружаться, можно просто добавить еще два или три сервера приложений.

Кроме того, типы аппаратного обеспечения, которые используются в настройках клиент-сервер (например, мини-компьютеры), как правило, могут быть обновлены в темпе, который наиболее близко соответствует росту приложения. В отличие от этого, архитектуры на основе хоста в основном полагаются на аппаратное обеспечение мэйнфреймов, которое необходимо масштабировать большими, дорогостоящими шагами, а архитектуры на основе клиентов имеют пределы, выше которых приложение не может расти, поскольку увеличение использования и данных может привести к

увеличению сетевого трафика до такой степени, что производительность становится недопустимо низкой.

Наиболее подходящей для реализации системы является клиент-серверная архитектура.

2.2.2. Функциональная схема проекта

Функциональная схема проекта приведена на рисунке ниже.

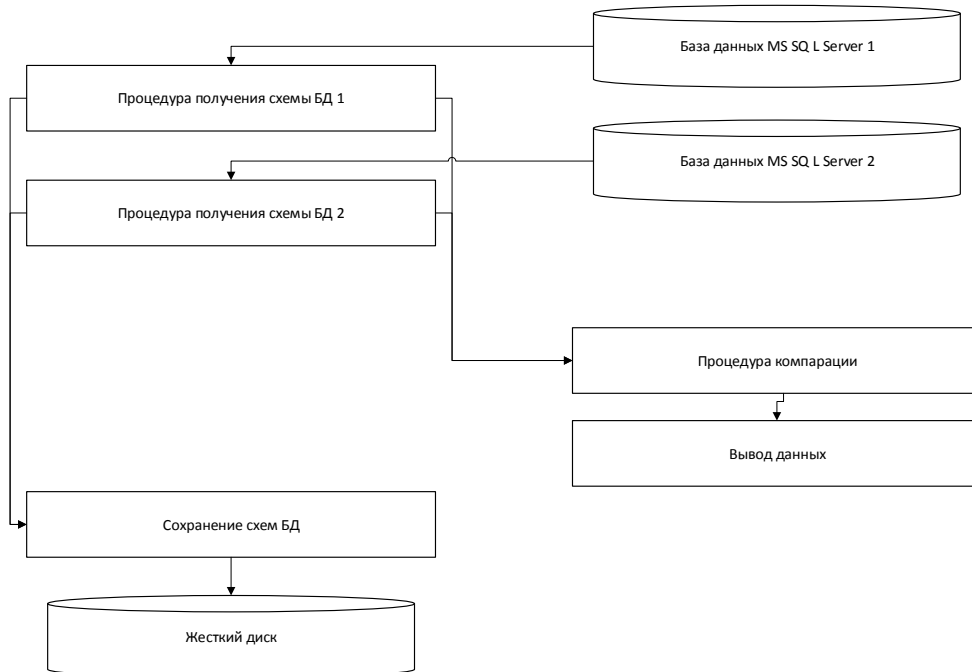


Рисунок 2.5. – Функциональная схема проекта

2.2.3. Структурная схема проекта

Структурная схема проекта показана на рисунке ниже.

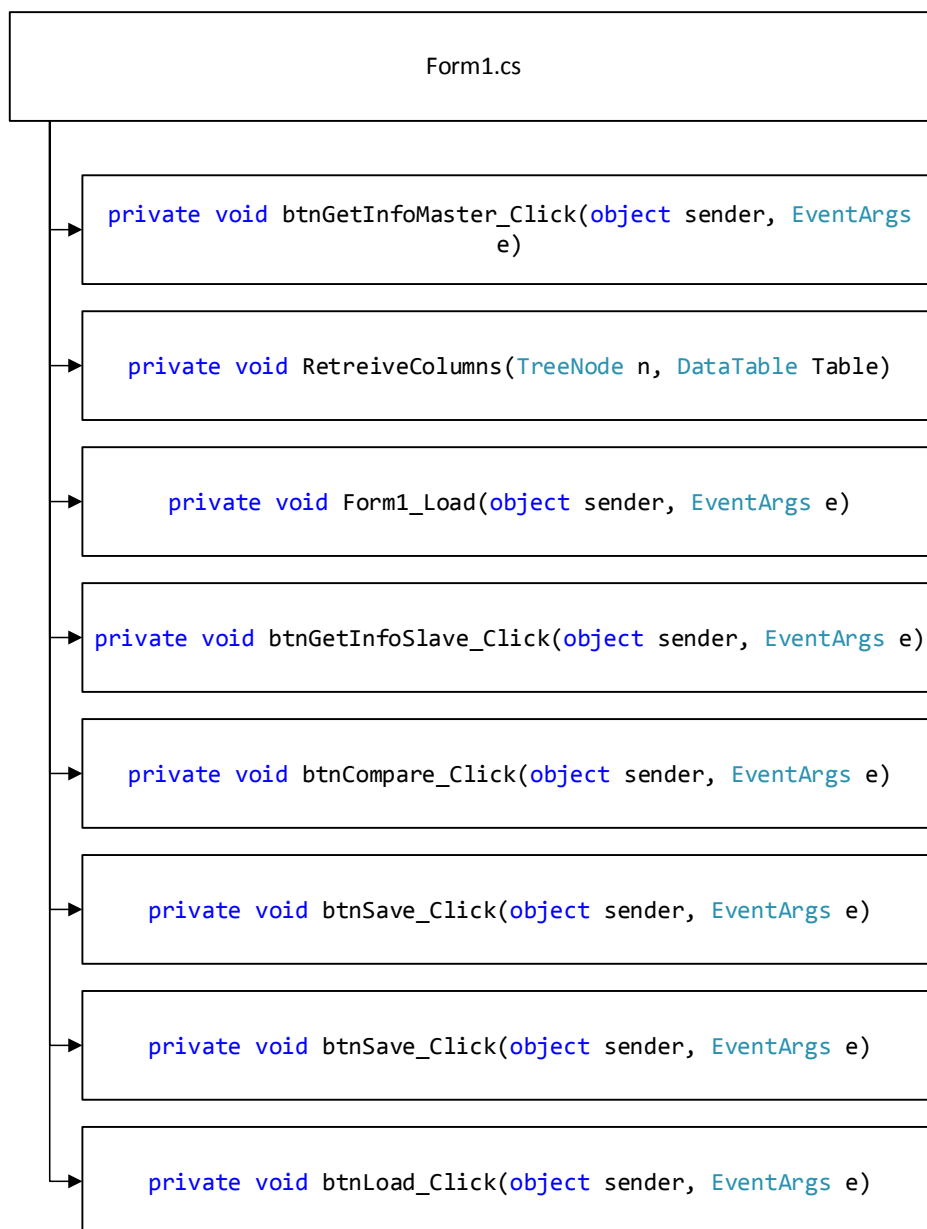


Рисунок 2.7. – Структурная схема проекта

2.2.4. Описание программных модулей

Основным модулем системы является модуль Form1.cs.

В начале работы процедуры компарации происходит проверка, загружена ли информация о схемах используемых баз данных.

Если информация не найдена – выводится сообщение об ошибке и выход из процедуры.

Далее последовательно проверяется наличие каждой из таблиц в обеих схемах данных. Если таблица найдена – происходит последовательное

сравнение всех полей данных в них. При отсутствии или необходимости модификации поля – выдается соответствующее сообщение.

Если таблица не найдена – выдается сообщение о необходимости создать соответствующую таблицу.

Схема работы данной процедуры показана на рисунке ниже.

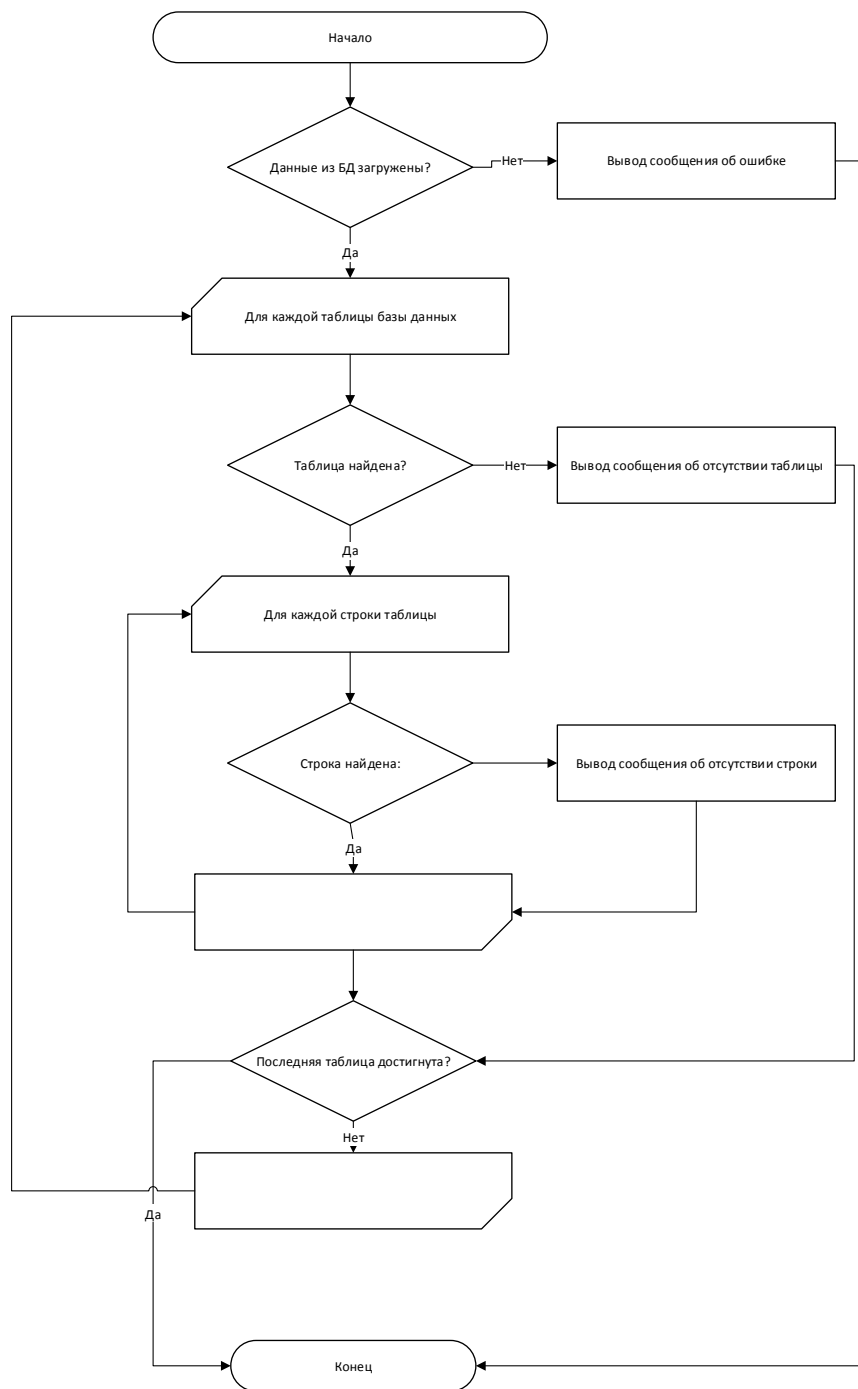


Рисунок 2.8. – Схема работы процедуры компарации

2.2.5. Схема взаимосвязи программных модулей и информационных файлов

Схема взаимосвязи программных модулей показана на рисунке ниже.

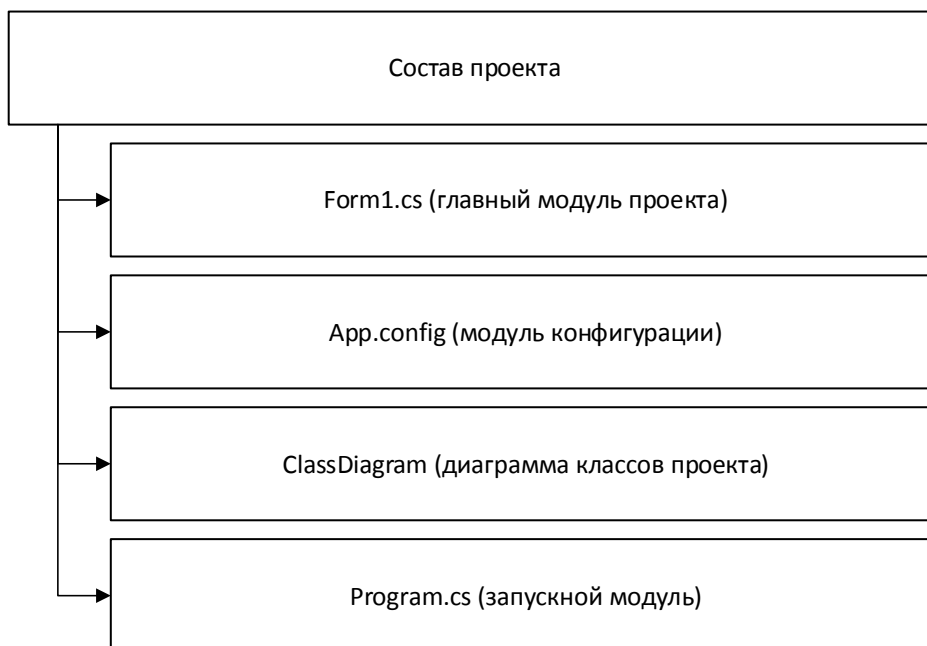


Рисунок 2.6. – Схема взаимосвязи программных модулей

2.3. Технологическое обеспечение задачи

2.3.1. Организация технологии сбора, передачи, обработки и выдачи информации

Технологическом обеспечением проекта автоматизации включены в себя процессы:

- получения первичной информации,
- обработки информации,
- выдачи результатной информации (формирования и передачи).

Состав операций, которые выполняются после получения первичной информации, включает в себя операции:

1. Съема первичной информации – процесса по получению количественного значения показателя, который отражает характеристику объектов и процессов предприятия. Вход решаемой задачи получает

первичную информацию из описания заказчиком желаемых услуг, на основании этой информации производятся расчеты всех необходимых параметров: количества, цены, и прочего.

2. Регистрация первичной информации – внесение всех количественных характеристик на какой – либо носитель. В процессе решения поставленной задачи автоматизации регистрация осуществляется механическим способом, то есть вводится с клавиатуры в экранные формы.

3. Сбор информации – операция получения пакета сообщений, набора первичных документов или файла на машинном носителе для последующей передачи и обработки. Эта операция в рамках решаемой задачи автоматизации происходит автоматическим способом, централизованно – то есть сбор информации происходит по мере ее возникновения в источниках.

При сборе первичной информации получают документы, данные из которых используются для корректировки нормативно–справочной информации и документы, представляющие оперативную информацию, используемую для расчетов.

При этом важно уделить внимание контролю входной информации следующими способами:

- визуальный контроль на экране дисплея,
- метод верификации, при котором осуществляется сверка ранее введенных и сохраненных в БД данных и данных первичных документов, вводимых оператором.

4. Передача информации на обработку – формальная технологическая процедура, поскольку все операции по регистрации и сбору первичной информации производятся на одном рабочем месте.

Технологический процесс обработки информации – это определенный комплекс операций, выполняемых в строго регламентированной последовательности с использованием определенных методов обработки данных и инструментальных средств.

В процессе создания и ведения БД используются следующие режимы

обработки:

- пакетный – для задач со слабой разветвленностью алгоритма, отсутствием необходимости вмешательства пользователя в процесс решения экономической задачи, с большим объемом исходных данных, длительным временем решения и получения результатов;

- диалоговый (интерактивный) – для выполнения функций управления диалогом, информирования пользователей, вывода информационных сообщений, обработки с их помощью прикладных программ и выдачи результатов;

- смешанный.

Пакетный режим обработки данных предполагает предварительный сбор пакета документов или подготовку входного файла первичной информации, с которых осуществляется наполнение основных файлов первичной информации или его обновление. Пакетный режим используется для работы с файлами оперативной информации в тех случаях, когда требуется ведение централизованной БД из локальных источников первичной информации при невозможности подключения этих источников к ИС. Применение пакетного режима позволяет уменьшить вмешательство оператора в процесс решения задачи, требует только предварительного ввода данных, исключает возможность вмешательства пользователя и, таким образом, изменяет последовательность выполняемых действий. Однако за счет этого появляется более полная загрузка оборудования, которое начинает работать по жесткому графику. В некоторых случаях для решения задачи выполняется и параллельная обработка данных. Пакетный режим более тесно связан с бумажной технологией.

Диалоговый режим предполагает активное вмешательство пользователя в процесс работы комплекса и ориентацию на безбумажную технологию. В ходе его выполнения отсутствует заранее установленная последовательность операций обработки данных и дополнительного их ввода. Особенностью диалогового режима обработки данных является ввод

или обновление отдельных записей файлов по мере необходимости.

С учетом специфики разрабатываемой проблемной области и предметной технологии решения задачи, выбор смешанного режима обработки обоснован следующими положениями:

- источники первичной информации подключены к ИС, однако имеет место бумажная технология сбора первичной информации;

- обработка информации начинается по мере поступления данных, хотя отчетные формы начинают составляться в момент получения всего объема данных за отчетный период;

- существует четкая последовательность технологических операций обработки информации, однако вмешательство пользователя в процесс обработки данных допускается.

Технологический процесс выдачи результатной информации происходит в двух направлениях:

- вывод результатных документов на печать;

- вывод результатных документов на экран.

Оба этих технологических направлений выдачи результатов решения поставленной задачи не исключают сохранения результатных данных в информационной базе. Таким образом, происходит ее пополнение, сохраненные данные являются исходными для решения аналогичных задач последующих периодов.

В рассматриваемой системе ввод информации происходит на основании схемы баз данных, полученных из системных таблиц базы данных.

Результатная информация формируется путем SQL-запросов к базе данных, позволяющих получать данные об отсутствующих таблицах, полях в этих таблицах, и необходимых изменениях в них.

2.3.2. Схема технологического процесса сбора, передачи, обработки и выдачи информации

Схема технологического процесса показана на рисунке ниже.

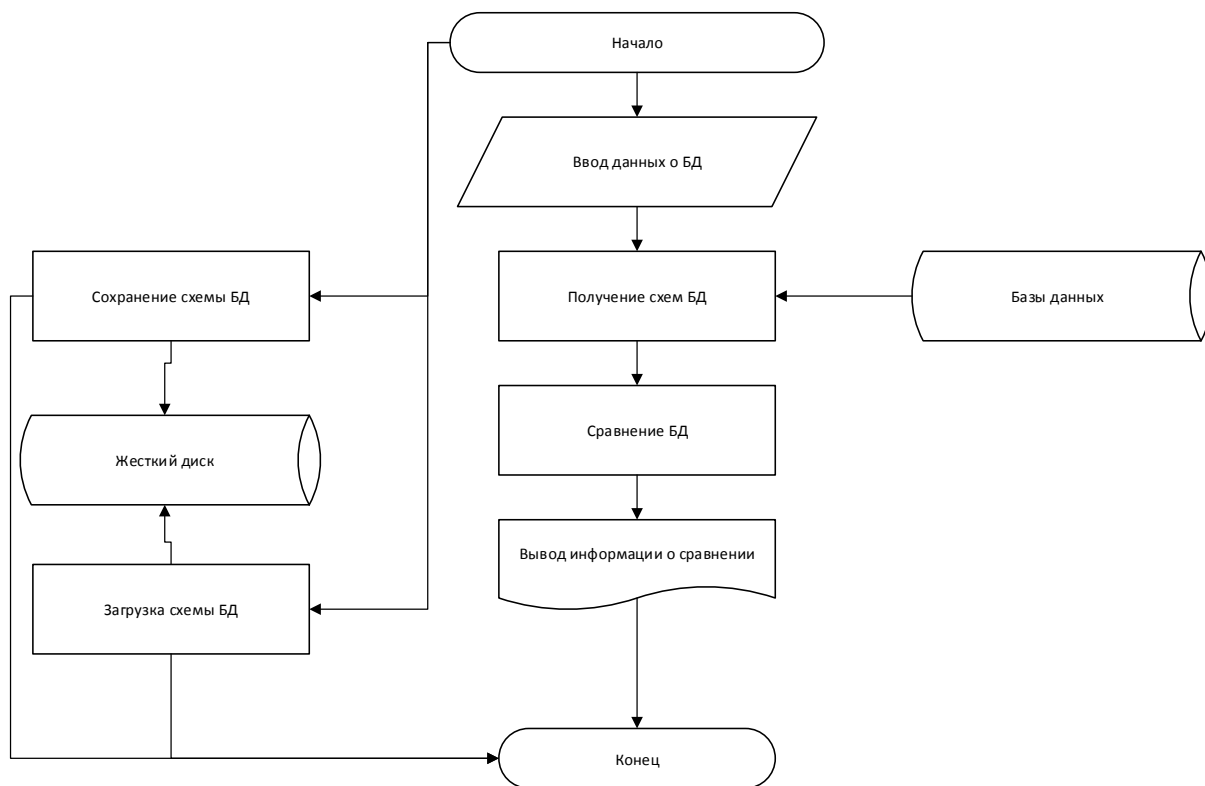


Рисунок 2.8. – Схема технологического процесса

2.4. Контрольный пример реализации проекта

После запуска исполняемого файла SQLComparator.exe на экран выводится главное окно приложения.

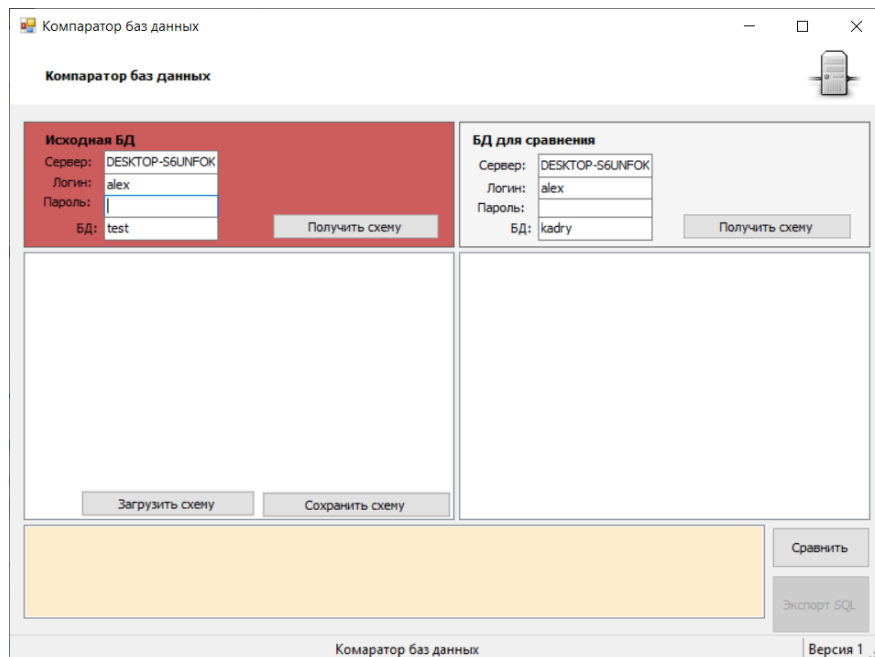


Рисунок 2.9. – Главное окно приложения

Вводим пароль для первой базы данных и получаем схему данных.

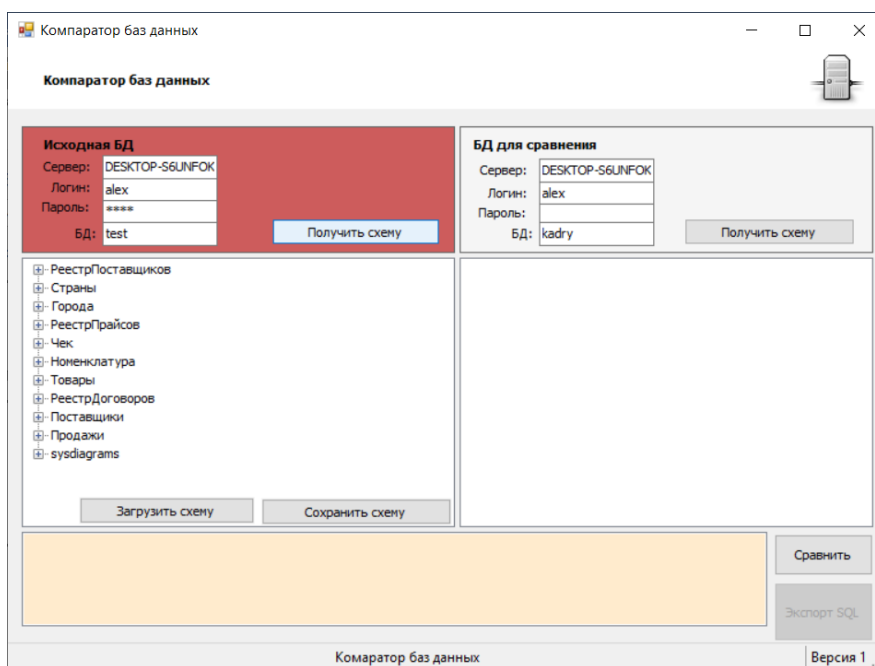


Рисунок 2.10. – Получение схемы первой БД

Вводим пароль для второй БД и получаем схему базы данных.

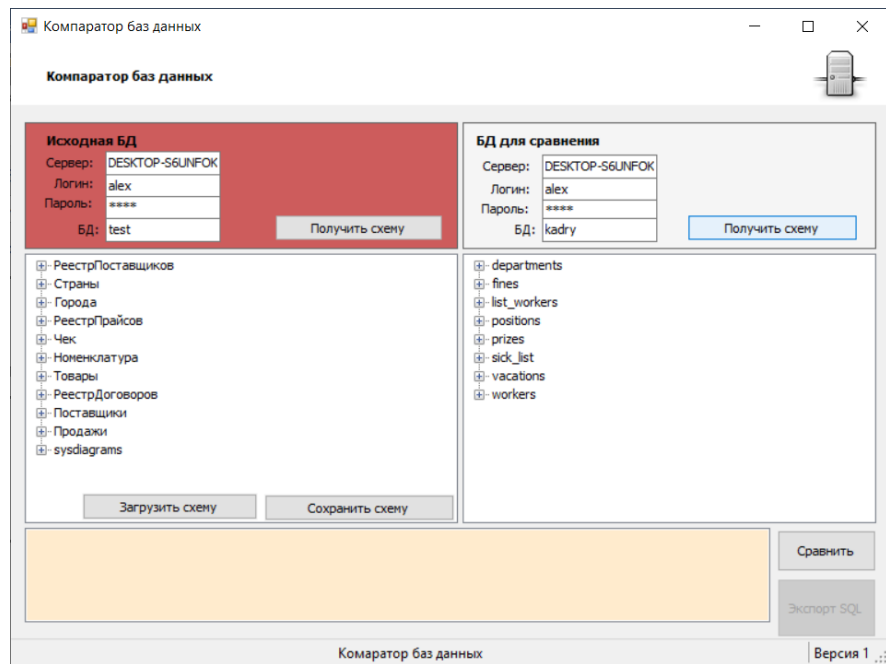


Рисунок 2.11. – Получение схемы второй БД

Кнопкой «Сравнить» производим сравнение баз данных. Сообщение о различиях выводится в лог-файл.

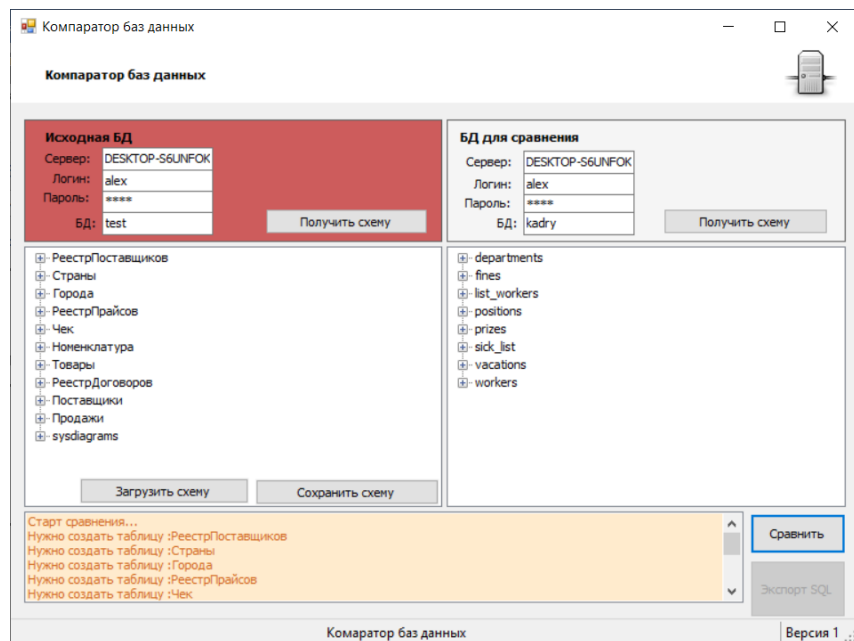


Рисунок 2.12. – Результат сравнения

По результатам сравнения можно выполнить создание или изменение таблиц в указанных базах данных.

ГЛАВА 3. ОЦЕНКА И ОБОСНОВАНИЕ ЭКОНОМИЧЕСКОЙ ЭФФЕКТИВНОСТИ ПРОЕКТА

3.1. Выбор и обоснование методики расчёта экономической эффективности

Все рабочие места специалистов отдела автоматизированы, в деятельности отдела используются информационные системы, но задачи по обработке информации выполняются с минимальным использованием информационных систем. Операции по выдаче дисков выполняются вручную. В связи с этим возникает проблема избыточной трудоемкости выполняемых задач, в результате имеются ошибки и неточности, а также относительно высока длительность процесса.

Так, в месяц работники выполняют обработку порядка 200 заявок на обработку информации. На обработку этой информации тратится в среднем 24 часа. Т.е. в месяц на процесс начисления дополнительных премий сотрудникам затрачивается около 288 часов. При этом около 50% трудовых затрат (144 часа в месяц) считаются избыточными, что в стоимостном выражении с учетом тарифа оплаты труда сотрудника (85,23 руб./час) составит около 12 273 руб./мес.

На основе проведенного анализа, основной проблемой предметной области является избыточная трудоемкость процесса, которая оценивается как 144 час/мес или 12 273 руб/мес.

Таким образом, можно определить основную цель проекта, как сокращение трудовых затрат на размещение оборудования на 50% на базе автоматизированных решений. В результате в качестве основного фактора экономической эффективности проекта выделена экономия на оплату труда сотрудников. Ожидаемые результаты проекта в предметной области отражены в таблице 3.1.

Таблица 3.1. – Результаты проекта в предметной области

Факторы улучшения бизнес-процессов в результате проекта	Факторы экономической выгоды, получаемые в результате проекта	Количественное выражение
Снижение количества ошибок	Экономия на оплату труда работников в части размещения оборудования (экономия трудозатрат)	Не менее 12 000 руб./мес (144 час/мес)
Уменьшение времени на ввод данных		

На основе типовой методики разработано информационно-методическое обеспечение для оценки экономической эффективности проекта, которое включает следующие показатели.

К трудовым показателям относятся следующие:

1. Трудозатраты на размещение оборудования по базисному варианту (Т0):

$T_0 = [\text{Средняя продолжительность по базисному варианту, час/начисл}] * [\text{Количество начислений в мес}]. \quad (3.1)$

2. Трудозатраты на размещение оборудования по проектному варианту (Т1):

$T_1 = [\text{Средняя продолжительность по проектному варианту, час/начисл}] * [\text{Количество начислений премии}]. \quad (3.2)$

3. Абсолютное снижение трудовых затрат (ΔT) на размещение оборудования:

$$\Delta T = T_0 - T_1, \quad (3.3)$$

при этом значение показателя должно быть не меньше 144 час/мес.

4. Коэффициент относительного снижения трудовых затрат (Кт):

$$K_T = \Delta T / T_0 * 100\% \quad (3.4)$$

5. Индекс снижения трудовых затрат или повышение производительности труда (Y_T):

$$Y_T = T_0 / T_1 \quad (3.5)$$

К стоимостным показателям относятся: абсолютное снижение стоимостных затрат (ΔC), коэффициент относительного снижения стоимостных затрат (Кс), индекс снижения стоимостных затрат (Yс), рассчитываемые аналогично.

1. Стоимостные затраты на размещение оборудования по базисному варианту (C0), руб/мес:

$$C0 = Cc * T0, \quad (3.6)$$

где Cc – тарифная ставка оплаты труда сотрудника отдела, руб./час.

2. Стоимостные затраты на размещение оборудования по базисному варианту (C1), руб/мес:

$$C1 = Cc * T1, \quad (3.7)$$

3. Абсолютное снижение стоимостных затрат (ΔC) на размещение оборудования, руб/мес:

$$\Delta C = C0 - C1 \quad (3.8)$$

при этом значение показателя должно быть не меньше 12000 руб/мес.

4. Коэффициент относительного снижения стоимостных затрат (Kc)

$$Kc = \Delta C / C0 * 100\% \quad (3.9)$$

5. Индекс снижения стоимостных затрат (Yc):

$$Yc = C0 / C1 \quad (3.10)$$

Для определения трудовых и стоимостных затрат на предметную область в базовом и проектном вариантах используем данные таблицы 3.3. Представленные данные получены в результате анализа предметной области до внедрения информационной подсистемы и на основе результатов тестовой эксплуатации ИПС.

Таблица 3.2. – Показатели эффективности проекта

Показатель	До проекта	После проекта
Средняя продолжительность операций по выдаче дисков	24	4
Тарифная ставка оплаты труда сотрудника отдела, руб.	85,23	
Количество операций в месяц	12	

Определим показатели изменения трудовых затрат на начисление дополнительной премии сотрудникам:

1. Определим общие трудовые затраты работников отдела по базисному варианту:

$$T0 = 24 \times 12 = 288 \text{ чел.-часа}$$

2. Общие трудовые затраты отдела поддержки специальных программ по проектному варианту:

$$T_1 = 4 \times 12 = 48 \text{ чел.-часов}$$

3. Рассчитаем абсолютное снижение трудовых затрат на выполнение задачи (формула 3.3.):

$$\Delta T = 288 - 48 = 240 \text{ чел.-часов}$$

4. Коэффициент относительного снижения трудовых затрат (формула 3.4) составит:

$$K = 240 / 288 \times 100\% = 83,3\%$$

5. Индекс снижения трудовых затрат или повышение производительности труда (формула 3.5):

$$Y = 288 / 48 = 6$$

Определим показатели изменения стоимостных затрат:

1. Определим общие стоимостные затраты работников отдела по базисному варианту:

$$C_0 = 288 \times 85,23 = 24546,24 \text{ руб/мес}$$

2. Общие стоимостные затраты работников отдела по проектному варианту:

$$C_1 = 48 \times 85,23 = 4091,04 \text{ руб/мес}$$

3. Рассчитаем абсолютное снижение трудовых затрат на выполнение задачи:

$$\Delta C = 24546,24 - 4091,04 = 20455,2 \text{ руб/мес}$$

4. Коэффициент относительного снижения стоимостных затрат составит:

$$K_c = 20455,2 / 24546,24 \times 100\% = 83,3\%$$

5. Индекс снижения трудовых затрат или повышение производительности труда:

$$Y_c = 24546,24 / 4091,04 = 6$$

Таким образом, наблюдается явное снижение трудовых и стоимостных затрат на размещение оборудования (на 83,3%) при использовании предлагаемой информационной системы.

3.2. Расчет показателей экономической эффективности

К показателям экономического эффекта и экономической эффективности относятся:

1. Годовой экономический эффект (Э):

$$\text{Э} = \Delta C - E_n * K, \quad (3.11)$$

где K_n – затраты на создание и внедрение проекта, руб, E_n – нормативный коэффициент эффективности единовременных затрат на создание ИП (0,15).

2. Срок окупаемости затрат на создание и внедрение проекта ($T_{ок}$), рассчитываемый в годах:

$$T_{ок} = K_n / \Delta C. \quad (3.12)$$

При этом затраты на создание и внедрение проекта (K_n) определяются:

$$K_n = [\text{Стоимость лицензионного ПО, руб.}] + [\text{Время разработки и отладки программы, час}] * [\text{Тариф оплаты труда программиста, руб/час}]. \quad (3.13)$$

Срок окупаемости должен быть не более трех лет (исходя из нормативного коэффициента эффективности единовременных затрат на создание ИП).

Информационно-методическое обеспечение оценки экономической эффективности проекта в части критериев отражено в таблице 3.3.

Таблица 3.3. – Показатели экономической эффективности проекта

Показатели эффективности	Критерии эффективности
Экономия на оплату труда работников (ΔC)	Не менее 12 000 руб./мес
Абсолютное снижение трудовых затрат работников (ΔT)	Не менее 144 чел-час/мес
Годовой экономический эффект (Э)	Больше 0
Срок окупаемости затрат на создание и внедрение проекта ($T_{ок}$)	Не более 3 лет

Представленные критерии оценки базируются на стоимостной оценке проблемной области и общеэкономической логике показателей.

Далее необходимо определить затраты на разработку информационной системы. К ним относятся:

- время разработки и отладки программы;
- затраты на операционную систему;
- заработная плата программиста в день.

Нужно рассчитать базовые показатели для вычисления затрат на разработку и отладку программы. Эти показатели включают время разработки и отладки программы, часовая тарифная ставка программиста, а также затраты на покупку необходимого для разработки программного и технического обеспечения. Стоит отметить, что организации имеется рабочее место, оснащенное операционной системой. Однако пришлось приобрести лицензионную версию программы.

Таблица 3.4. – Основные показатели для вычисления затрат на разработку и отладку программы

Показатель	Значение показателя
Время разработки и отладки программы, дн.	14
Часовая тарифная ставка программиста, руб.	105
Стоимость лицензионной версии программы SQLServer, руб	128 300

Основываясь на этих базовых показателях, мы можем рассчитать стоимость работы разработчика. Она определяется как время разработки, умноженное на зарплату в месяц:

$$K=14 \times 8 \times 105 = 11\,760 \text{ руб.}$$

Общие затраты на создание проекта (проектирование и внедрение) составят:

$$K_{\text{п}}=11\,760+128\,300=140\,060 \text{ руб.}$$

Экономический эффект рассчитываем по формуле 3.6:

$$\text{Э} = 12 \times 20\,455,2 - 0,15 \times 58\,060 = 224\,453,4 \text{ руб.}$$

Срок окупаемости единовременных затрат, рассчитываем по формуле 3.7:

$$\text{Ток} = 140\,060 / 224\,453,4 = 0,62 \text{ года} = 7,5 \text{ месяцев}$$

Полученные показатели можно представить графически в виде диаграмм.

На рисунке 3.1. представлена диаграмма, показывающая изменение трудозатрат в связи с внедрением проекта.

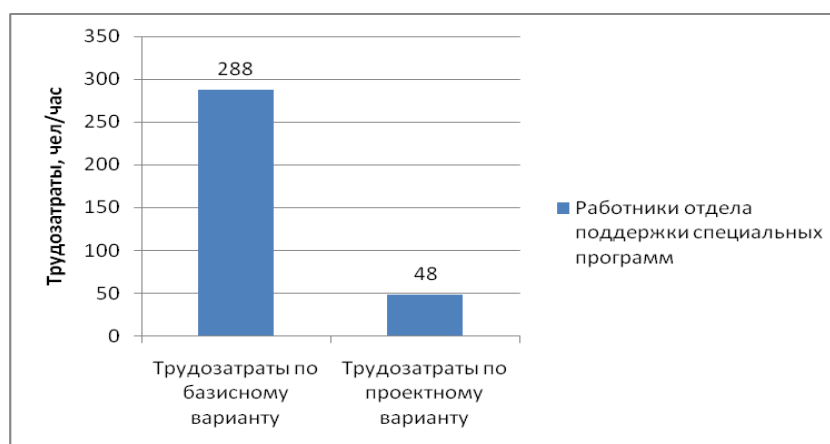


Рис. 3.1. – Трудозатраты до и после внедрения проекта

На рисунке 3.2. представлена диаграмма, показывающая изменение стоимостных затрат предприятия в связи с внедрением проекта.

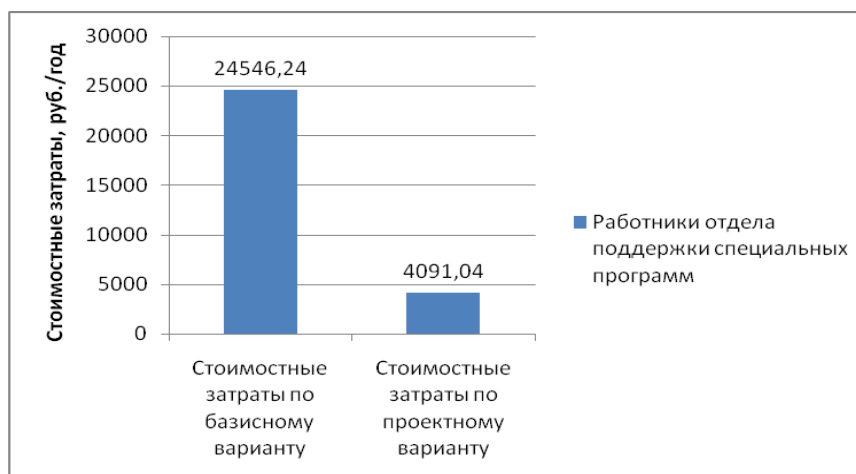


Рис. 3.2. – Стоимостные затраты предприятия до и после внедрения проекта

Таким образом, все полученные показатели соответствуют норме и свидетельствуют об экономической выгоде проекта.

Показатели экономии на оплату труда работников по проектному варианту составят 20 455,2 руб./мес. По этим показателям проект полностью удовлетворяет поставленным перед ним требованиям (не менее 12 000 руб/мес).

Показатели трудозатрат также снизились, они составляют 48 чел-час/мес, снижение произошло на 240 чел-часов. Данные показатели также соответствуют поставленным критериям (не менее, чем на 144 чел-часа).

Снижение трудовых затрат персонала позволит сократить время внеурочной работы, снизить утомляемость, а значит повысить эффективность труда работников.

Экономический эффект от внедрения программного продукта составил 224 453,4 руб., что также удовлетворяет предъявленным требованиям (больше нуля). Срок окупаемости единовременных затрат составит 7,5 месяцев. Этот срок удовлетворяет предъявленным требованиям (не более трех лет). Чтобы проект был эффективен на практике, необходимо ввести новые должностные инструкции, которые будут обязывать работников отдела вести обработку данных с помощью новой информационной подсистемы.

ЗАКЛЮЧЕНИЕ

Развитием вычислительной техники и появлением вместительных внешних ЗУ прямого доступа было предопределено быстрое развитие автоматизированных систем, имеющих разное назначение и масштаб, которое особенно заметно в сфере бизнес-приложений. Подобные системы обрабатывают большие объемы информации, зачастую обладающую достаточно сложной структурой, требующей оперативности при обработке, часто обновляющейся и в то же время требующей длительного срока хранения. Такие системы являются примерами автоматизированных систем управления предприятием, банковских систем, систем резервирования и продаж билетов и т. д. Другие направления, стимулировавшие развитие, с одной стороны - это системы по управлению физическими экспериментами, занимающиеся обеспечением сверхоперативной обработки в реальном времени больших потоков данных, передаваемых датчиками, а с другой — информационно-поисковые библиотечные системы.

Экономические тенденции современной жизни во многом опираются на эффективность управления, которая определяет производительность работы предприятия в целом. Важным моментом, как в организации управления, так и в общей эффективности деятельности учреждения являются системы обработки информации. Требования к ним включают следующее: оперативное получение информации, обеспечение отчетности по итогам работы в общей и детализированной форме, производство полномасштабного точного анализа данных, определение тенденций изменения ключевых показателей.

Большинство существующих СУБД работают в системе Windows, поскольку именно эта операционная среда обеспечивает более полную реализацию возможностей компьютера, чем DOS. Появление высокопроизводительной вычислительной техники на уровне персональных компьютеров (ПК) и снижение стоимости таковых, обусловило широкую популяризацию операционной среды Windows, в которой создание

программного обеспечения (ПО) не требует от разработчика заботиться в полной мере о распределении системных ресурсов. Дополнительно эта операционная система снижает зависимость ПО, в частности СУБД, от аппаратных ресурсов ЭВМ.

Широкая распространенность компараторов баз данных, организующих синхронизацию и восстановление данных, не является залогом распространения компараторов, основой для которых служит логический анализ компарируемых данных, они не распространены широко. Исследованием в данной области возможно сэкономить время для разработчика, которое требуется на поиски решений каких-либо проблем, связанных с несоответствием данных.

В ходе данной работы были выделены основные компоненты разрабатываемого приложения и связь между ними. Также был определён минимальный набор функций приложения.

Были разработаны основные модули компаратора баз данных. Был разработан модуль получения схемы БД, описана база данных, под которую разрабатывается компаратор, описана последовательность компарации данных и приведены основные характеристики программы.

В последней части работы обоснована экономическая эффективность решения.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Агальцов В.П. Базы данных. – М.: Мир, 2002. – 376 с.
2. Бажин И.И. Информационные системы менеджмента. – М.: ГУ-ВШЭ, 2000. – 688 с.
3. Барановская Т.П., В.И. Лойко, М.И. Семенов, А.И. Трубилин. Информационные системы и технологии в экономике. Учебник. 2-е изд., доп. и перераб. М.: Финансы и статистика, 2005. – 416 с.
4. Бальжинов А.В., Михеева Е.В. Анализ и диагностика финансово-хозяйственной деятельности предприятия: учеб. пособие, 2008. – 128 с.
5. Гагарина Л.Г. Разработка и эксплуатация автоматизированных информационных систем: Учебное пособие / Л.Г. Гагарина. - М.: ИД ФОРУМ: НИЦ Инфра-М, 2013. - 384 с.
6. Гвоздева В.А. Введение в специальность программиста: Учебник / В.А. Гвоздева. - 2-е изд., испр. и доп. - М.: ИД ФОРУМ: ИНФРА-М, 2011. - 208 с.
7. Голицына О.Л., Попов И.И. Программирование на языках высокого уровня: Учебное пособие / О.Л. Голицына, И.И. Попов. - М.: Форум, 2008. - 496 с.
8. Дубейковский В.И. Эффективное моделирование с AllFusionProcessModeler (BPWin) 4.1.4 / В.И. Дубейковский. – М.: Диалог-МИФИ, 2007 – 384 с.
9. Емельянова Н.З., Партыка Т.Л., Попов И.И. Проектирование информационных систем: учеб.пособие / Н.З. Емельянова, Т.Л. Партыка, И.И. Попов. - М.: Форум, 2009. - 432 с.
10. Заботина Н.Н. Проектирование информационных систем: Учебное пособие / Н.Н. Заботина. - М.: ИНФРА-М, 2011. - 331 с.
11. Колдаев В.Д. Структуры и алгоритмы обработки данных: Учебное пособие / В.Д. Колдаев. - М.: ИЦ РИОР: НИЦ ИНФРА-М, 2014. - 296 с.

- 12.Култыгин О. П. Администрирование баз данных. СУБД MS SQL Server: учеб. пособие / О. П. Култыгин. - М.: МФПА, 2012. - 232 с.
- 13.Мокеев В.В. Методология моделирования данных в среде ERWin: Учебное пособие для лабораторных работ. – Челябинск: Изд. ЮУрГУ, 2003. – 45 с.
- 14.Немцова Т.И. Программирование на языке высокого уровня. Программир. на языке Object Pascal: Учеб.пос. / Т.И. Немцова и др; Под ред. Л.Г. Гагариной - М.: ИД ФОРУМ: НИЦ Инфра-М, 2013 – 496 с.
15. Румянцева Е.Л., Слюсарь В.В. Информационные технологии: Учебное пособие / Е.Л. Румянцева, В.В. Слюсарь: Под ред. Л.Г. Гагариной. - М.: ИД ФОРУМ: НИЦ Инфра-М, 2013. - 256 с.
- 16.Сибилёв В.Д. Проектирование баз данных: Учеб. пособие. — Томск: Томский межвузовский центр дистанционного образования, 2007. — 201 с.
- 17.Уткин В.Б. Математика и информатика: Учебное пособие / В.Б. Уткин, К.В. Балдин, А.В. Рукоосуев. - 4-е изд. - М.: Дашков и К, 2011. - 472 с.
- 18.Федотова Е.Л., Федотов А.А. Информатика: Курс лекций. Учебное пособие / Е.Л. Федотова, А.А. Федотов. - М.: ИД ФОРУМ: ИНФРА-М, 2011. - 480 с.
- 19.Нортроп Т.Е. Основы разработки приложений на платформе Microsoft.NET Framework. Учебный курс Microsoft. Перевод с англ./ Т. Нортроп, Ш. Уилдермьюс, Б. Райан. - М.: «Русская редакция», 2007. - 864 с.
- 20.Робинсон С.А. С# для профессионалов. Том 1. / С. Робинсон, О. Корнес, Д. Глин, Б. Харвей. - М.:Лори, 2003. - 1002 с.
- 21.Робинсон С.А. С# для профессионалов. Том 2. / С. Робинсон, О. Корнес, Д. Глин, Б. Харвей. - М.:Лори, 2003. - 998 с.
- 22.Троелсен Э.Л. Язык программирования С# 2005 и платформа.NET 2.0, 3-е издание.: Пер с англ. / Э. Троелсен. - М.: ООО «И.Д. Вильямс», 2007. - 1168 с.

23.Ватсон К.А. С#. / К. Ватсон, М. Беллиназо, О. Корнс, Д. Эспиноза. - М.: Лори, 2005. - 863 с.

ПРИЛОЖЕНИЕ

Исходный код модуля компарации Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Xml.Serialization;

namespace SQLComparator
{
    public partial class Form1 : Form
    {
        clsDatabase _Master;
        clsDatabase _Slave;
        public Form1()
        {
            InitializeComponent();
        }

        private void btnGetInfoMaster_Click(object sender, EventArgs e)
        {
            try
            {
                clsDatabase Master = new clsDatabase();

                Master.open(txtMasterServer.Text, txtMasterUserID.Text,
                    txtMasterPassword.Text, txtMasterDB.Text);

                SQLComparator.Properties.Settings.Default.MasterDB =
                    txtMasterDB.Text.Trim();
                SQLComparator.Properties.Settings.Default.MasterUserID
                    = txtMasterUserID.Text.Trim();
                SQLComparator.Properties.Settings.Default.MasterServer
                    = txtMasterServer.Text.Trim();
                SQLComparator.Properties.Settings.Default.Save();

                for (int i = 0; i < Master.Tables.Length; i++)
                {
                    TreeNode n = new TreeNode();
                    n.Name = "T" + i.ToString();
                    n.Text = Master.Tables[i].TableName;
                    MasterTree.Nodes.Add(n);
                    RetrieveColumns(n, Master.Tables[i]);
                }
            }
            catch { }
        }
    }
}
```

```

        }
        _Master = Master;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    }
}

private void RetrieveColumns(TreeNode n, DataTable Table)
{
    for (int i = 0; i < Table.Columns.Count; i++)
    {
        TreeNode col = new TreeNode();
        col.Name = n.Name + "col" + i.ToString();
        col.Text = Table.Columns[i].ColumnName;
        n.Nodes.Add(col);
    }
}

private void Form1_Load(object sender, EventArgs e)
{
    txtMasterDB.Text =
    SQLComparator.Properties.Settings.Default.MasterDB;
    txtMasterServer.Text =
    SQLComparator.Properties.Settings.Default.MasterServer;
    txtMasterUserID.Text =
    SQLComparator.Properties.Settings.Default.MasterUserID;

    txtSlaveDB.Text =
    SQLComparator.Properties.Settings.Default.SlaveDB;
    txtSlaveServer.Text =
    SQLComparator.Properties.Settings.Default.SlaveServer;
    txtSlaveUserID.Text =
    SQLComparator.Properties.Settings.Default.SlaveUserID;
}

private void btnGetInfoSlave_Click(object sender, EventArgs e)
{
    try
    {
        clsDatabase Slave = new clsDatabase();
        Slave.open(txtSlaveServer.Text, txtSlaveUserID.Text,
        txtSlavePass.Text, txtSlaveDB.Text);

        SQLComparator.Properties.Settings.Default.SlaveDB =
        txtSlaveDB.Text.Trim();
    }
}

```

```

        SQLComparator.Properties.Settings.Default.SlaveUserID
= txtSlaveUserID.Text.Trim();
        SQLComparator.Properties.Settings.Default.SlaveServer
= txtSlaveServer.Text.Trim();
SQLComparator.Properties.Settings.Default.Save();

for (int i = 0; i < Slave.Tables.Length; i++)
    {
    TreeNode n = newTreeNode();
        n.Name = "T" + i.ToString();
        n.Text = Slave.Tables[i].TableName;
SlaveTree.Nodes.Add(n);
RetreiveColumns(n, Slave.Tables[i]);
    }
    _Slave = Slave;
}
catch (Exception ex)
    {
    MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK,
    MessageBoxIcon.Error);
    }
}

privatevoid btnCompare_Click(object sender, EventArgs e)
    {
    List<DataTable> _tables = newList<DataTable>();
    clsResultUI ui = newclsResultUI();
    string sql = "";
    if (_Slave == null || _Master == null)
        {
        MessageBox.Show("Пожалуйстаполучитеинформациюпообеимбазамданных", "", Me
ssageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
        }
    lstLog.Items.Clear();
    lstLog.Items.Add("Стартсравнения...");
    foreach (TreeNode n in MasterTree.Nodes)
        {
        bool bFound = false;
        for (int i = 0; i < SlaveTree.Nodes.Count; i++)
            {
            if (n.Text == SlaveTree.Nodes[i].Text)
                {
                for (int x = 0; x < n.Nodes.Count; x++)
                    {
                    bool bColFound=false;
                    for (int y = 0; y < SlaveTree.Nodes[i].Nodes.Count; y++)
                        {

```

```

if (n.Nodes[x].Text.ToUpper() ==
SlaveTree.Nodes[i].Nodes[y].Text.ToUpper())
    {

bColFound = true;
break;

    }

if (!bColFound)
    {

string tp =
_Master.Tables[n.Index].Columns[n.Nodes[x].Text].DataType.ToString();
System.Data.DataColumn
c=_Master.Tables[0].Columns[0];

sql = sql + "\n" + "Alter Table " + n.Text + " Add Column (" +
n.Nodes[x].Text + " " + tp + " NOT NULL Default('t') )";

lstLog.Items.Add("Модифицируйтетаблицу " + n.Text + " : ADD Column ("
+ n.Nodes[x].Text+ " " + c.ToString() + ")");
    }

bFound = true;
break;

    }

if (!bFound)
    {
lstLog.Items.Add("Нужносоздатьтаблицу :" + n.Text);
sql = sql + "\n" + "Create Table " + n.Text + " { ";
for (int j=0; j < n.Nodes.Count; j++)
    {
sql = sql + " " + n.Nodes[j].Text + " char(20) , ";
    }
    sql = sql + "} \n";
    }

    ui.textBox1.Text = sql;
// ui.ShowDialog();
lstLog.Items.Add("Сравнениезавершено.");

}

privatevoid btnSave_Click(object sender, EventArgs e)
    {
saveFileDialog1.ShowDialog();
if (saveFileDialog1.FileName == "") return;
StreamWriter stWriter = null;
XmlSerializer xmlSerializer;

```

```

try
    {
xmlSerializer = newXmlSerializer(_Master.GetType());
stWriter = newStreamWriter(saveFileDialog1.FileName);
bool includeNameSpace = true;
if (!includeNameSpace)
    {
        System.Xml.Serialization.XmlSerializerNamespaces
xs =
newXmlSerializerNamespaces();
//To remove namespace and any

//other inline information tag

xs.Add("", "");
xmlSerializer.Serialize(stWriter, _Master, xs);
    }
else
    {
xmlSerializer.Serialize(stWriter, _Master);
    }
MessageBox.Show("Схемасохраненауспешно", "", MessageBoxButtons.OK,
MessageBoxIcon.Information);
    }
catch (Exception exception)
    {
MessageBox.Show(exception.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    }
finally
    {
if (stWriter != null) stWriter.Close();
    }
}

privatevoid btnLoad_Click(object sender, EventArgs e)
    {
openFileDialog1.ShowDialog();
if (openFileDialog1.FileName != "")
    {
XmlSerializer xmlSerializer;
FileStream fileStream = null;
try
    {
if (_Master==null) _Master = newclsDatabase();
xmlSerializer = newXmlSerializer(_Master.GetType());
fileStream = newFileStream(openFileDialog1.FileName,
FileMode.Open, FileAccess.Read);
object objectFromXml =

```

```

xmlSerializer.Deserialize(fileStream);
        _Master = (clsDatabase)objectFromXml;
MasterTree.Nodes.Clear();

for (int i = 0; i < _Master.Tables.Length; i++)
    {
TreeNode n = newTreeNode();
        n.Name = "T" + i.ToString();
        n.Text = _Master.Tables[i].TableName;
MasterTree.Nodes.Add(n);
RetreiveColumns(n, _Master.Tables[i]);
    }
MessageBox.Show("Загруженоуспешно!", "", MessageBoxButtons.OK,
MessageBoxIcon.Information);

    }
catch (Exception Ex)
    {
MessageBox.Show(Ex.Message, "Error Loading", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    }
finally
    {
if (fileStream != null) fileStream.Close();
    }
    }

privatevoid button1_Click(object sender, EventArgs e)
    {

    }

privatevoid toolStripStatusLabel1_Click(object sender, EventArgs e)
{

    }
}
}

```