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

Кафедра Прикладная математика и информатика
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки / специальности)

Мобильные и сетевые технологии
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка модуля обработки и анализа данных системы телефонии "ООО Сам РЭК-Эксплуатация"»

Обучающийся А.С. Приб
(И.О. Фамилия) (личная подпись)

Руководитель к.т.н., доцент, Н.В. Хрипунов
(ученая степень, звание, И.О. Фамилия)

Консультант к.п.н., доцент, А.В. Егорова
(ученая степень, звание, И.О. Фамилия)

Тольятти 2025

Аннотация

Бакалаврская работа выполнена на тему «Разработка модуля обработки и анализа данных системы телефонии "ООО СамРЭК-Эксплуатация"».

Объектом исследования является система обработки данных телефонии компании «ООО СамРЭК-Эксплуатация», обеспечивающая хранение, обработку и передачу сведений о входящих и исходящих вызовах, зафиксированных телефонной станцией.

Предметом исследования выступает автоматизированный модуль, разработанный для анализа и структурирования бинарных файлов .ama, содержащих информацию о детализации звонков, с последующей передачей этих данных в базу данных Oracle.

Целью работы является разработка отказоустойчивого программного решения, позволяющего инженеру по данным автоматизировать рутинный процесс обработки .ama файлов, тем самым устранив человеческий фактор, повысив эффективность и надёжность бизнес-процесса.

Выпускная квалификационная работа состоит из введения, трёх глав, заключения и списка литературы. Общий объём работы составляет 55 страниц, содержит 25 рисунков.

Первая глава посвящена анализу деятельности компании и существующего бизнес-процесса обработки вызовов.

Во второй главе рассмотрены этапы проектирования модуля: от требований к логике взаимодействия компонентов и базы данных до архитектуры решения.

В третьей главе описана реализация модуля, составные компоненты, механизмы логирования, автозапуска и тестирования, подтверждающие стабильность работы.

Разработанный модуль будет внедрён на рабочем месте инженера по данным, обеспечивая анализ и обработку, загрузку данных в БД.

Abstract

The bachelor's thesis is titled «Development of a Data Processing and Analysis Module for the Telephony System of "SamREK-Operation" LLC».

The object of the study is the telephony data processing system of “SamREK-Operation” LLC, which is responsible for storing, processing, and transmitting information about incoming and outgoing calls recorded by the telephone exchange.

The subject of the study is an automated module developed for analyzing and structuring binary .ama files that contain detailed call records, with subsequent transmission of this data to an Oracle database.

The aim of this work is to develop a fault-tolerant software solution that automates the routine .ama file processing workflow for the data engineer, thereby eliminating human error and increasing the efficiency and reliability of the business process.

The thesis consists of an introduction, three chapters, a conclusion, and a list of references. The total volume is 55 pages and includes 25 figures, 1 scheme, and 7 diagrams.

The introduction justifies the relevance of the topic and defines the objectives, tasks, object, and subject of the study.

The first chapter analyzes the company's operations and the current call processing workflow.

The second chapter presents the design stages of the module: from requirements to interaction logic and data architecture.

The third chapter describes the implementation of the module, its components, logging mechanisms, auto-start scripts, and testing, which confirms the system’s stable operation.

The developed module is intended to be deployed at the workplace of the data engineer, providing automated analysis, processing, and data loading into the database.

Содержание

Введение

Современные системы телефонии являются важным инструментом для эффективного управления бизнес-процессами, особенно в компаниях, где коммуникация играет ключевую роль. Однако в процессе развития технологий появляются и реализуются все более новое, эффективное программное обеспечение (ПО), стандарты и оборудование, а внутренние процессы обработки данных соответственно становятся устаревшими и недостаточно гибкими. Кроме того, данный процесс может вызвать необходимость перестройки целой бизнес-логики или какую-то функцию внутри нее что в случае крупных организаций, может быть, не выгодно и побудит использовать что-то устаревшее, либо другое решение, не требующее глобальной перестройки логики организации.

Преддипломная практика проходила в ООО «СамРЭК-Эксплуатация», где существующие решения не обеспечивают достаточной гибкости и эффективности, что требует разработки специализированного модуля для обработки и анализа данных системы телефонии.

Актуальность темы обусловлена необходимостью оптимизировать процесс обработки данных системы телефонии. Разработка модуля, способного эффективно обрабатывать и анализировать данные, позволит не только улучшить текущие бизнес-процессы, но и повысить как качество обслуживания клиентов, так и, к примеру, юридическую защищенность компании. Кроме того, подобное решение может быть востребовано в других компаниях, столкнувшихся с аналогичной проблемой.

Целью бакалаврской работы является разработка модуля обработки и анализа данных системы телефонии для ООО «СамРЭК-Эксплуатация», который позволит оптимизировать бизнес-процессы и повысит эффективность работы с данными.

Для достижения поставленной цели необходимо:

- определить проблему, требования и рамки программного обеспечения для модуля;
- получить и ознакомиться с документацией оборудования у инженера по связи отдела эксплуатации;
- описать текущий бизнес-процесс по обработке и анализу, выделить проблему;
- описать архитектуру разработанной системы и основных модулей;
- разработать модуль согласно документации и установленных рамок и требований;
- протестировать модуль с описанием методов и результатов тестирования.

Объектом исследования работы является система обработки телефонии ООО «СамРЭК-Эксплуатация».

Предметом исследования – автоматизированный модуль обработки и анализа данных, поступающих из системы телефонии.

Для разработки модуля обработки и анализа данных выбраны методы, включающие анализ требований, проектирование архитектуры, разработку ПО и тестирование. Эти методы позволят обеспечить поэтапное создание модуля, начиная с изучения текущих проблем и заканчивая внедрением готового решения.

Практическая значимость бакалаврской работы заключается в том, что разработанный модуль позволит оптимизировать процесс обработки данных в системе телефонии ООО «СамРЭК-Эксплуатация», сокращая временные затраты и уменьшая вероятность ошибки. Кроме того, результаты работы могут быть применены в других компаниях.

Разработанный модуль будет внедрен в систему обработки телефонии ООО «СамРЭК-Эксплуатация», протестирован и в следствии результатов либо отправлен на доработку, либо заменит старое решение.

1. Анализ применения системы телефонии и деятельности ООО СамРЭК-Эксплуатация

1.1 Общая информация о предприятии

ООО «СамРЭК-Эксплуатация» является одной из дочерних компаний акционерного общества «СамРЭК» и основано 29 января 2013 года, а с 31 июля 2020 года входит в число системообразующих предприятий губернии.

В настоящее время существует двадцать пять объектов по Самарской области, большая часть которых расположена в малых городах и поселениях. Так, к примеру, компания обслуживает более 58 объектов водоснабжения и водоотведения в городском округе Жигулёвск, а их общая протяженность по Самарской области составляет более 356 км.

ООО «СамРЭК-Эксплуатация» занимается разными задачами:

– эксплуатация объектов жилищно-коммунального хозяйства, энергетики, тепло-, газо-, паро-, водоснабжения, водоотведения, технических средств контрольно-измерительных систем и автоматики;

– разработка эксплуатационной, конструкторской и проектной документации в сфере тепло-, водоснабжения и водоотведения;

– диагностика и обследование, текущий и капитальный ремонт объектов жилищно-коммунального хозяйства, теплообеспечения, водоснабжения и водоотведения, технических средств контрольно-измерительных систем и автоматики;

– выполнение работ по режимно-наладочным испытаниям;

– проведение электротехнических измерений;

– энергоаудит.

Предприятие ООО «СамРЭК-Эксплуатация» имеет функциональное деление и включает в себя множество департаментов и отделов. В рамках данной работы будут рассмотрены основные единицы структуры вплоть до

уровня отделов, а наиболее важные для работы будут описаны и разобраны подробнее.

На рисунке 1 представлена структура предприятия, входящее в АО «СамРЭК».

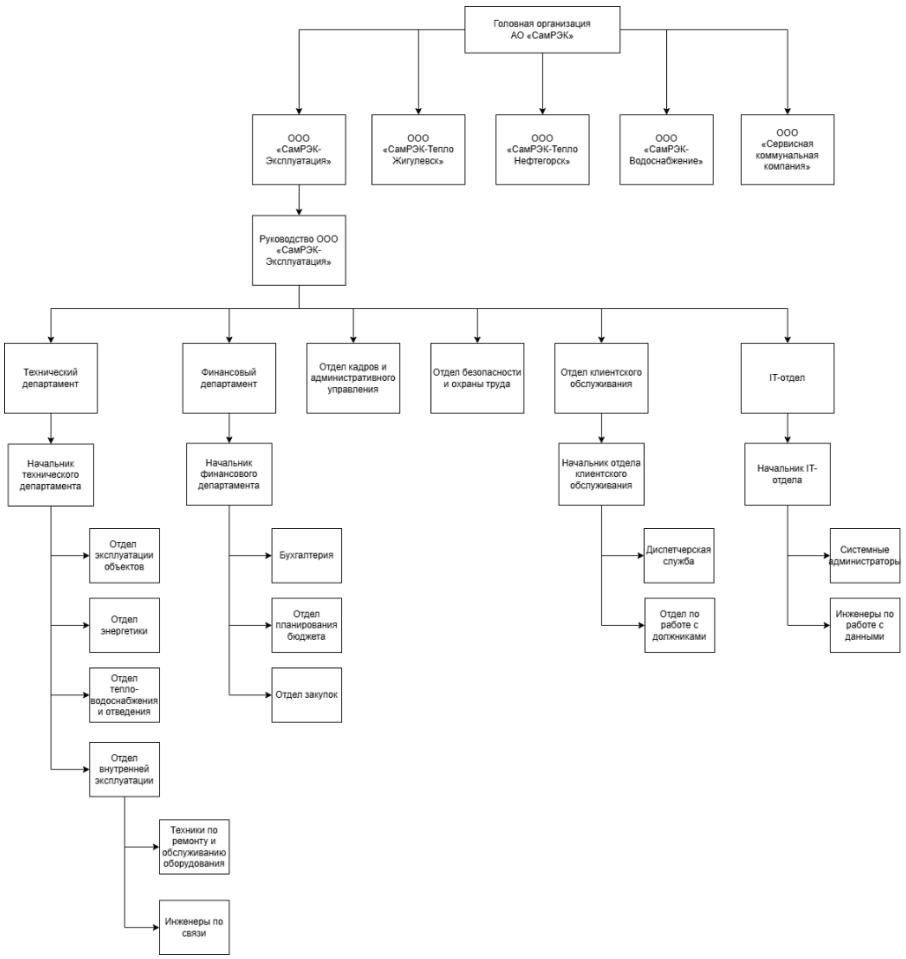


Рисунок 1 – Организационная схема компании ООО «СамРЭК-Эксплуатация»

Структура управления ООО «СамРЭК-Эксплуатация» является, как уточнялось ранее, функциональной, но, кроме того, линейной, что хорошо видно на рисунке 1.

Стоит разобраться какой департамент, отдел чем занимается в соответствии с вышеуказанным рисунком.

Технический департамент, исключая внутренний отдел эксплуатации, который следует рассмотреть отдельно, включает в себя:

– отдел эксплуатации объектов отвечает за общее техническое состояние объектов (зданий, сооружений, инфраструктуры) и их обслуживание;
– отдел энергетики отвечает за обеспечение объектов электроэнергией, контроль за энергопотреблением;
– отдел тепловодоснабжения и отведения отвечает за обеспечение объектов теплом, горячей и холодной водой, а также за отведение сточных вод.

Финансовый департамент включает в себя:

– бухгалтерия ведет учет финансовых операций компании, подготовку отчетности и соблюдения налогового законодательства;
– отдел планирования бюджета занимается планированием и контролем финансовых ресурсов компании;
– отдел закупок отвечает за приобретение товаров, услуг и оборудования.

Отдел кадров и административного управления отвечает за управление персоналом вроде отбора и найма сотрудников, проведения собеседований, а также юридические вопросы в которые входит оформление трудовых договоров и ведение документации кадров.

Отдел безопасности и охраны труда отвечает за обеспечения безопасности сотрудников и объектов компании, а именно за пожарную, промышленную безопасность, охрану объектов, разбор инцидентов и проведение инструктажей.

Отдел клиентского обслуживания отвечает за взаимодействие с клиентами компании, в него входит:

– Диспетчерская служба отвечает за оперативное реагирование на заявки клиентов, а именно принимает заявки, координирует выезд бригад, мониторит выполнение заявок и информирует клиентов;

– Отдел по работе с должниками занимается взысканием задолженностей за предоставленные компанией услуги.

IT-отдел необходимо рассмотреть отдельно, как и внутренний отдел эксплуатации из технического департамента.

Преддипломная практика проходила в IT-отделе, взаимодействуя с отделом внутренней эксплуатации технического департамента, отвечающего не только за телефонию, но и за обслуживание техники внутри организации.

IT-отдел отвечает за работу информационных технологий, автоматизацию бизнес-процессов и защиту данных компании. Данный отдел взаимодействует с множеством других, к примеру, обеспечивает автоматизацию учета и отчетности финансового департамента, автоматизацию работы диспетчерской службы.

Основные его функции:

– разработка и поддержка ПО: Создание и внедрение решений автоматизации бизнес-процессов вроде систем учета, обновление и поддержка существующих систем интеграция различные IT-систем между собой;

– управление цифровой инфраструктурой: Обслуживание серверов, баз данных, резервное копирование и восстановление данных;

– информационная безопасность: Защита данных от несанкционированного доступа, мониторинг и предотвращение кибератак;

– техническая поддержка: Помощь сотрудникам в решении технических проблем, например, с персональным компьютером (ПК), ПО или периферией наподобие принтера;

– аналитика данных: Сбор, обработка и анализ данных, а также разработка отчетов для руководства.

Отдел внутренней эксплуатации занимается управлением и поддержкой телекоммуникационных систем и, как упоминалось ранее, обслуживанием внутренней техники. За телекоммуникацию и все то, что с ней связано

отвечают инженера связи, а состояние внутреннего оборудования и ее обслуживание на техниках данного отдела.

Основные его функции:

– обеспечение связи: Поддержка и управление телефонными системами, VoIP и другими средствами связи;

– сетевое администрирование: Установка, конфигурация и обслуживание сетевого оборудование, вроде маршрутизаторов, коммутаторов и серверов;

– техническая поддержка: Решение проблем с оборудованием и ПО, связанных с телекоммуникацией;

– мониторинг: Постоянный мониторинг работы системы.

1.2 Модель применения системы телефонии

Для того, чтобы разобраться, где и в чем проблемы существующего бизнес-процесса, следует начать с более обобщенной картины, рисунок 2, DFD (Data Flow Diagram) [2].

DFD – графическая модель показывающая как данные перемещаются внутри системы, преобразуются и хранятся. Она помогает визуализировать: источники и получатели данных, процессы, потоки данных и хранилища данных. Это полезно при выявлении проблемных мест, для упрощения сложных процессов, оптимизации потоков и служит как основа для автоматизации, показывая какие процессы можно перевести в цифровой формат.

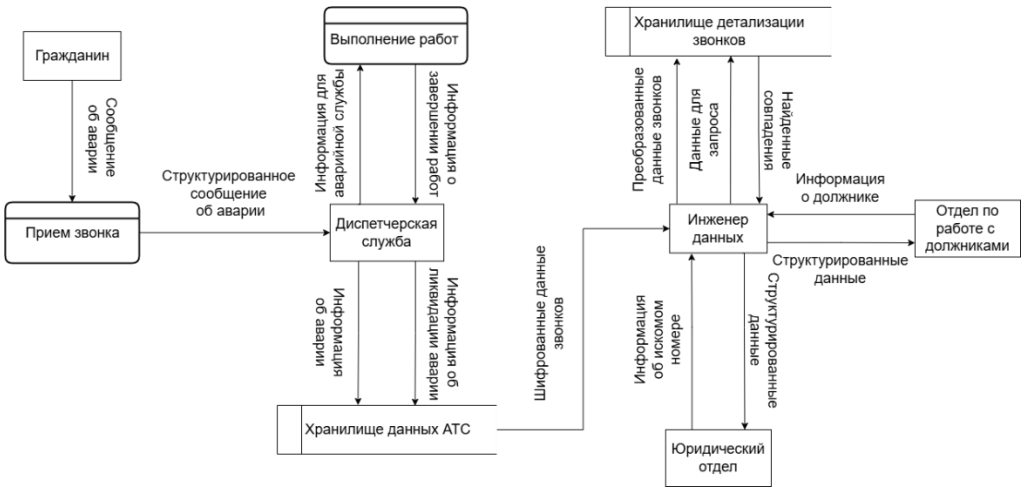


Рисунок 2 – AS IS DFD схема потоков данных

Данная схема, представленная рисунком 2, позволяет рассмотреть, как потоки текут между системами, базами данных, сущностями:

– поток начинается с гражданина, который через прием звонка передает диспетчерской службе структурированное сообщение об аварии;

– диспетчера передают необходимую информацию на выполнение работ;

– после окончания приема звонка, передачи информации на выполнение работ, то есть что после внешнего или что после внутреннего соответственно, на автоматической телефонной станции (АТС) формируется детализированная запись по совершенным звонкам;

– записи в шифрованном виде попадают с хранилища АТС к инженеру данных, где он их преобразует, заносит в хранилище детализации звонков;

– инженер по заявкам отделов обращается в хранилище звонков и возвращает необходимую информацию в структурированном виде, к примеру, отделу по работе с должниками или юридическому отделу.

Следует отметить, что инженер сам, вручную проводит каждый файл в хранилище детализации звонков.

В целях более детального отображения, включающего работу инженера, следует воспользоваться BPMN (Business Process Model and Notation) [6], которая используется для моделирования бизнес-процессов.

BPMN - система условных обозначений, которая отображает процессы в одном из направлений, слева направо или сверху вниз, с помощью блок-схем имеющих определенное начало и конец. Она позволяет детально и наглядно отобразить последовательность и логику взаимосвязи действий, событий, исполнителей и объектов бизнес-процесса, а также выявить узкие места. В основе этой системы лежат объекты потока управления, соединяющие объекты, пулы, дорожки и артефакты:

– объектом потока управления является событие, действие или логические операторы в виде развилок;

– соединяющие объекты позволяют установить взаимосвязь и поток между событиями, например, позволяет через ассоциацию связать артефакт с объектом события потока управления или определить обмен сообщениями между элементами бизнес-процессов, которые могут находиться на разных дорожках или пулах;

– потоком является идущая по установленному направлению связь между различными элементами процесса, отображающая последовательность;

– процессом именуется пул, являющийся пространством, где формируется модель самого процесса, он при необходимости может быть разбит на дорожки;

– дорожками разделяют пул для указания зон ответственности исполнителей;

– исполнитель может быть как отделом компании, конкретным сотрудником организации, так и информационной системой, ее модулем.

В рамках данной работы необходимо разобрать бизнес-процессы связанные с системой обработки звонков, обработкой детализированных

записей о вызове. На рисунке 3 представлена схема обозначенных бизнес-процессов.

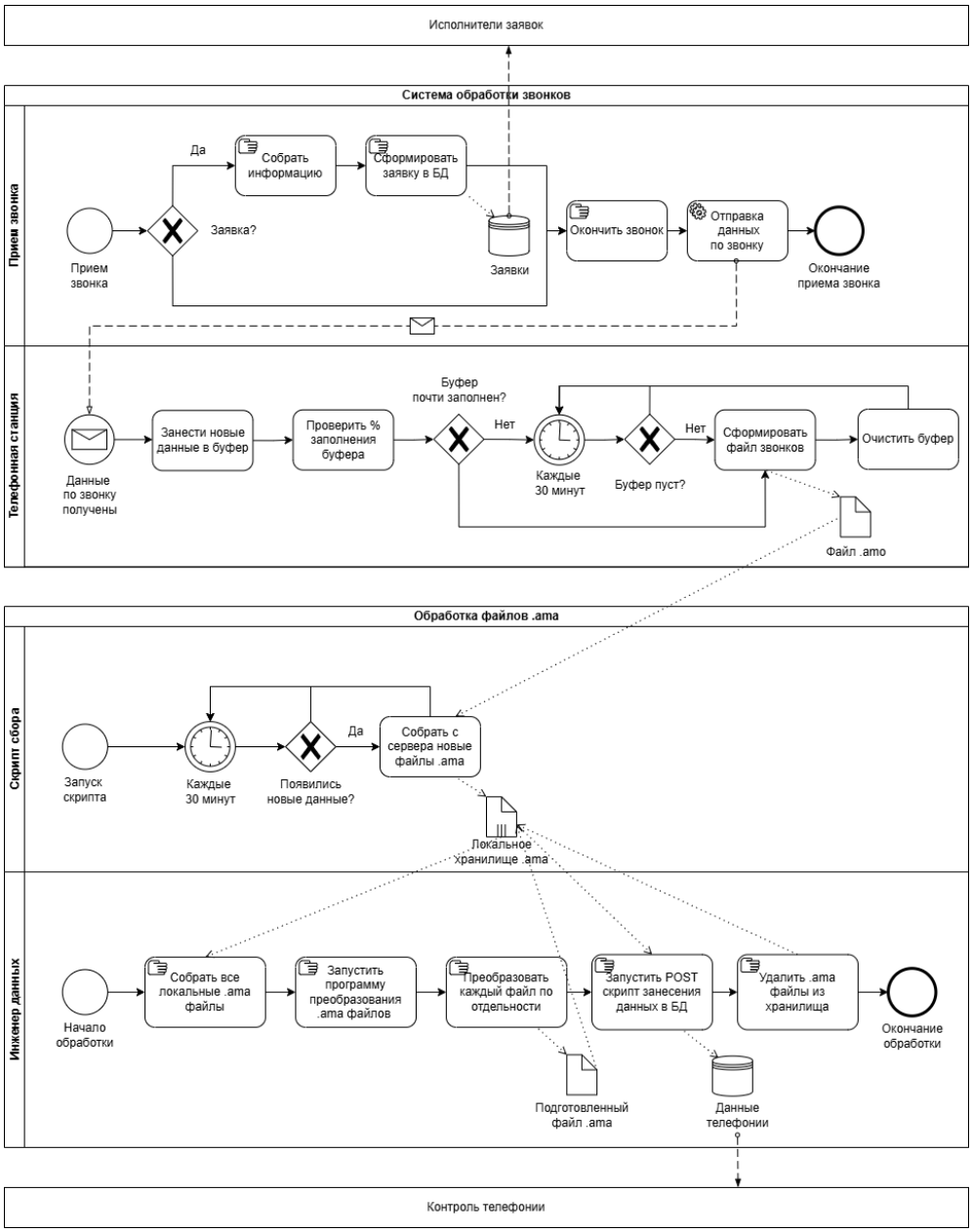


Рисунок 3 – BPMN схема бизнес-процессов

Рисунок 3 представляет из себя описание двух бизнес-процессов, процесса обработки звонков и файлов расширения .ama, получаемых в следствии поступления какого-либо внутреннего или внешнего звонка на любое подключенное внутри организации устройство связи. Под внешним и внутренним в данном контексте подразумевается, что звонок поступил,

например, от клиента в диспетчерскую и от диспетчерской к исполнителям клиентских заявок соответственно. В качестве отправной точки на схеме системы обработки звонков взята работа диспетчерской службы из отдела клиентского обслуживания. Рассмотрим подробнее каждый описанный схематикой процесс.

В рамках системы обработки звонков сначала фиксируется любой из вышеуказанных вызовов, если это заявка, в данном случае на диспетчерскую, то вручную, по отработанным шаблонам вопросов собирается информация, формируется заявка и заносится в соответствующую базу данных. Уже на телефонной станции каждый раз по окончанию звонка поступают данные, которые заносятся в буфер. По истечению таймера в 30 минут проверяется буфер данных на пустоту, если он пуст, то снова устанавливается таймер, иначе формируется файл, расширения .ama, полной детализации звонков взятых с буфера, а после буфер отчищается. Для того, чтобы данный буфер не переполнился запускается проверка оставшегося свободного места буфера после внесения новых данных и, если его недостаточно, то таймер пропускается и процесс сразу переходит к формированию вышеописанного файла с последующей отчисткой буфера.

Процесс обработки файлов .ama работает параллельно и начинается с запуска скрипта сбора на ПК инженера по работе с данными, IT-отдела. Данный скрипт каждые 30 минут проверяет появились ли на станции новые файлы .ama в определенном каталоге системы и собирает их на локальный компьютер инженера, если же он оказался пуст, то снова устанавливается таймер. Затем инженер вручную запускает программу, которая преобразует каждый .ama файл только по отдельности. Полученные подготовленные файлы инженер заносит в базу данных (БД) и удаляет из локального хранилища, оканчивая процесс обработки.

Проблема существующего бизнес процесса заключается в том, что инженер данных выполняет множество ручных действий в обработке файлов .ama. Данные множественная ручная работа несет за собой усложнение,

замедление, некоторые критические ситуации, которые необходимо рассмотреть и решить.

Можно перечислить следующие недостатки:

– ручной труд,
– человеческий фактор,
– низкая эффективность,
– риск накопления данных.

Здесь выделены недостатки, которые происходят друг из друга.

Ручной труд означает что инженер своими руками производит множество действий, которые отнимают время и требуют внимательности.

Исходя из внимательности, особо проявляется человеческий фактор. Данный фактор подразумевает что ответственный инженер, являясь человеком, может случайно удалить, не обработать, не занести в БД.

Множество действий привносит низкую эффективность. Проделываемый труд над обработкой, занесением данных в БД несоизмеримо велик с цифровой автоматизацией по коэффициенту полезного действия (КПД) за одинаковый промежуток времени. Так, в данном ключе, при отстроенной автоматизации инженер сможет повысить свою эффективность, занимаясь другими задачами.

Возвращаясь к человеческому фактору, существует риск накопления данных. Человек может забыть про необходимость провести файлы в БД, а также подвержен болезням и в следствии продолжительно отсутствовать что при неимении заменяющего может стать критической точкой.

По итогу рассмотрения DFD, BPMN, определения проблемы и рассмотрения недостатков, получается следующий вид потоков данных, представленный рисунком 4, который призван устранить недостатки, избавив существующий бизнес-процесс от проблемы.

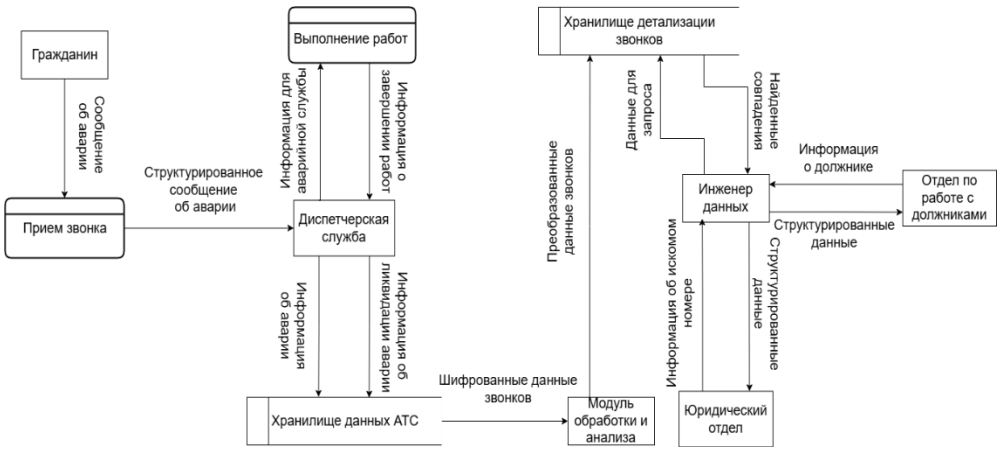


Рисунок 4 – TO BE DFD схема потоков данных

На рисунке 4 те же потоки, процессы, сущности и хранилища что отмечались ранее, за исключением новой сущности - модуля обработки и анализа данных. Этот модуль берет на себя ответственность за преобразования данных звонков, снимая ее с инженера данных.

Подытоживая, автоматизация повысит общий КПД IT-отдела за счет перенаправления времени и труда на другие задачи у инженера, а также избавит бизнес-процесс от человеческого фактора, минимизируя ошибки конвертации и потери данных.

1.3 Требования к программному обеспечению

Из беседы с инженером по данным были вынесены следующие важные пункты по ПО:

а) Не должно иметь пользовательского интерфейса;

б) Возможность настраивания;

в) Должно запускаться с ПК инженера;

г) Ограничения по использованию ресурсов ПК нет;

д) Должно работать без установки сторонних программ на ПК;

е) Не важен применяемый язык программирования (ЯП);

ж) Работа с Oracle PL/SQL Developer;

з) Должно само поддерживаться и в случае прекращения работы перезапускаться и\или уведомлять об случившемся.

Пункт б будет решен по средству предварительного создания файла формата .ini с названием config. Он позволит настраивать некоторый функционал ПО, но необходимо будет предварительно проверять целостность настроек и только после считывать установленный шаблон.

Из-за простоты, количества созданных библиотек и личного обширного опыта – выбор остановился на Python [15] по пункту е.

Однако, по пункту д, подходят компилируемые языки потому как такие ЯП генерируют самодостаточные исполняемые файлы, которые не требует установки сторонних программ, а Python же является интерпретируемым, требующий установки самого Python интерпретатора. Для решения данной проблемы можно воспользоваться PyInstaller [14] – инструментом для упаковки приложения на ЯП Python в исполняемый файл.

Для работы с Oracle PL/SQL Developer [12] можно воспользоваться библиотеками Python-oracledb [16] или SQLAlchemy [17]. SQLAlchemy является универсальной библиотекой, которая работает со множеством БД что в следствии не позволяет углубленно пользоваться той или иной БД, кроме того, привносит много излишнего кода в ПО, увеличивая итоговый вес. В свою очередь Python-oracledb лишен перечисленных недостатков и нацелен на работу конкретно с Oracle. По указанным причинам для пункта ж выбор останавливается на Python-oracledb.

Пункт в введен потому как работа с БД не должна зависеть от работы телефонной станции. ПО должно быть совместимо с операционной системой ПК IT-отдела, в данном случае это Windows 7.

В целях решения пункта з можно использовать дополнительный, параллельный процесс, который следит за работой главного, перезапуская при необходимости и, как вариант, делать это перекрестно. Под перекрестно имеется ввиду что главный точно так же проверяет дополнительный на

существование. Кроме того, необходимо добавить исполняемый файл в автозапуск системы.

Поводя итог: ПО должно функционировать на Windows 7 само поддерживаясь через дополнительный процесс или перекрестную проверку, ЯП будет Python, для работы с Oracle будет использоваться Python-oracledb, а итоговый результат будет скомпилирован через PyInstaller.

В первом разделе была проведена всесторонняя оценка структуры и бизнес-процессов предприятия, в частности - процессов, связанных с системой телефонии. С помощью DFD и BPMN была визуализирована текущая модель обработки звонков и .ama файлов, выявлены проблемы: ручной труд, уязвимость к человеческому фактору, низкая эффективность и риски накопления необработанных данных. На основе выставленных требований и анализа была заложена основа для проектирования решения.

2. Проектирование модуля обработки и анализа данных системы телефонии «ООО СамРЭК-Эксплуатация»

2.1 Логическое проектирование приложения

Логическое проектирование приложения – важный этап разработки программного обеспечения, на котором определяются ключевые взаимодействия пользователей с системой. В данном разделе будут рассмотрены две основные UML-диаграммы: диаграмма вариантов использования и диаграмма последовательности [19, 18]:

– диаграмма вариантов использования отображает функциональные возможности системы и взаимодействие между Actor и прецедентами. Под Actor или актор подразумевается пользователь или внешняя система, а прецедент это вариант использования. Она представлена на рисунке 5;

– диаграмма последовательности показывает взаимодействие объектов в рамках одного сценария, отражая порядок сообщений между ними. Отображена рисунком 6.

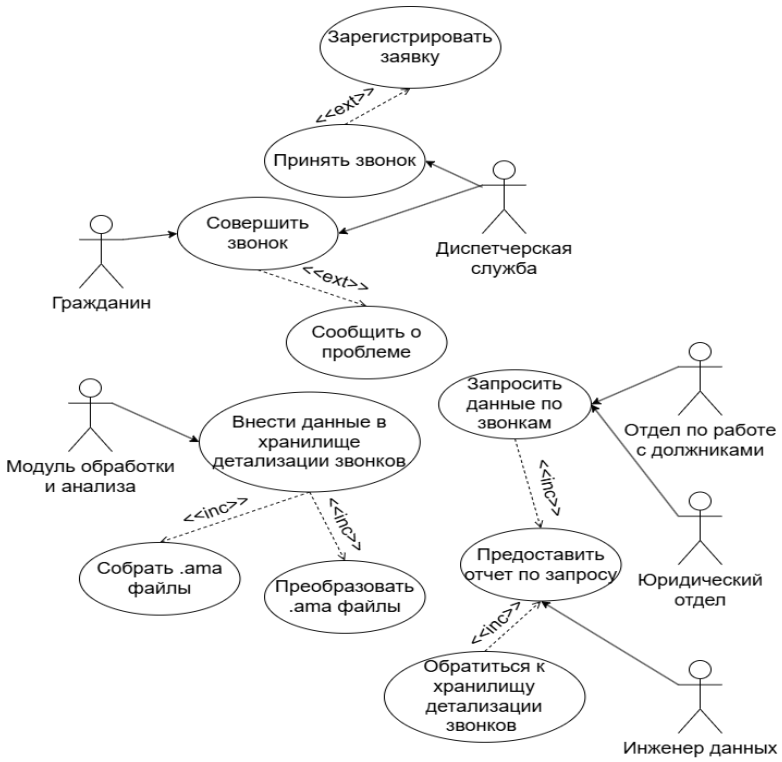


Рисунок 5 – Диаграмма вариантов использования

Диаграмма рисунка 5 представляет допустимое взаимодействие с системой различных акторов, обеспечиваемое благодаря автоматизации процесса обработки .ama файлов.

Так, после получения данных с телефонной станции на ПК инженера и обработки через ПО, запись их в БД, различные пользователи системы могут выполнять запросы в соответствии со своими задачами. Так, к примеру, запросить факт о состоявшемся звонке может юридический отдел и отдел по работе с должниками для доказательной базы.

Для примера можно рассмотреть последовательность сценария запроса данных по звонку юридическим отделом. Данный разбор представлен рисунком 6.

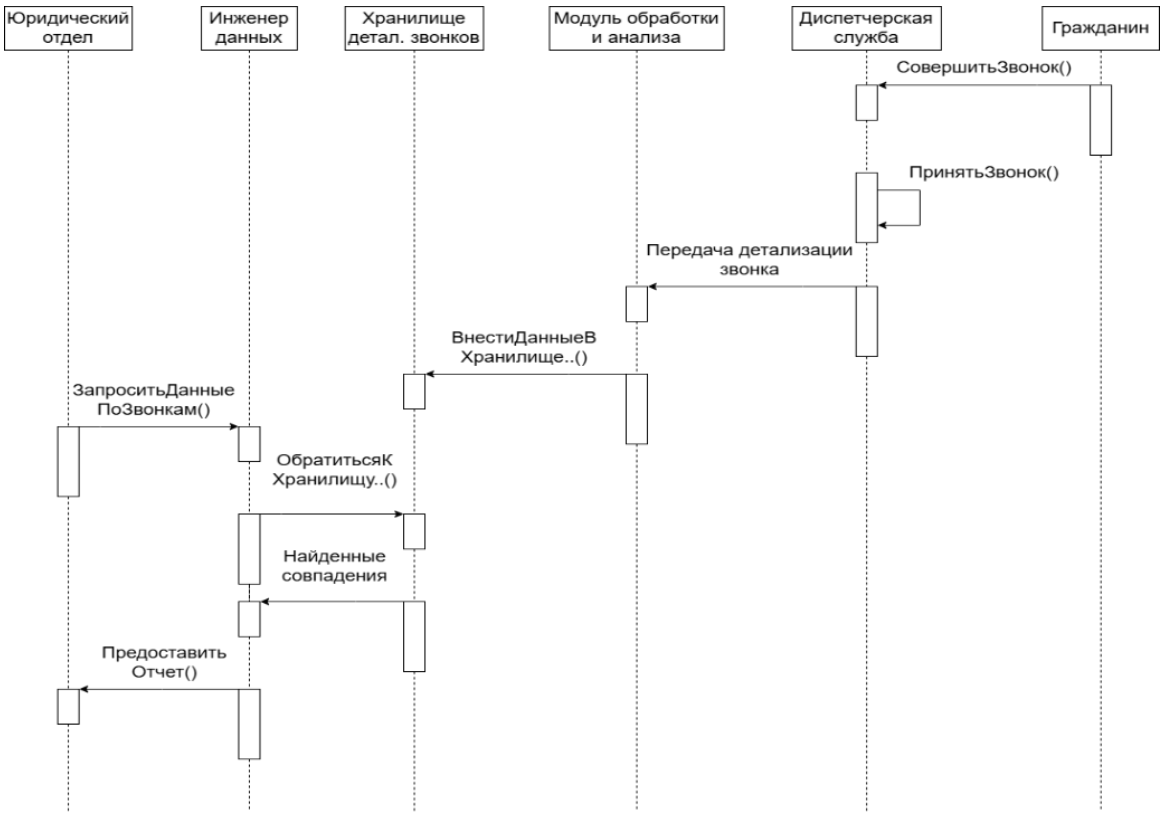


Рисунок 6 – Диаграмма прецедента юридического отдела

Диаграмма с рисунка 6 представляет разбор в виде последовательности действий при прецеденте с запросом данных по звонкам юридическим отделом.

Данная последовательность начинается с того, что гражданин совершает звонок, диспетчерская служба его принимает.

По окончанию звонка, телефонная станция, сформировав файлы формата .ama, передает эти файлы в локальное хранилище инженеру данных, где модуль сам преобразовывает файлы и вносит в хранилище детализации звонков, где данные хранятся для дальнейшего использования.

Юридический отдел при необходимости делает запрос звонков инженеру данных.

Инженер данных обращается к хранилищу звонков, формирует ответ и отправляет отчет запрашиваемому.

Полученный отчет может использоваться для выявления факта звонка или того, что звонка не поступало. В свою очередь это может являться важным элементом защиты в суде.

Диаграмма иллюстрирует полный цикл обработки звонков – от их фиксации на телефонной станции до использования для юридических целей.

2.2 Проектирование данных

Проектирование базы данных является также одним из ключевых этапов разработки. Оно включает в себя следующие этапы моделирования [9]:

– Концептуальное моделирование [11], определяющее основные сущности, их атрибуты и взаимосвязи без привязки к конкретной системе управления баз данных (СУБД);

– Логическое моделирование [4], где концептуальная модель преобразуется в схему базы данных с определением таблиц, полей, типов данных и связей;

– Физическое моделирование [5] уже учитывает особенности конкретной СУБД, к примеру, индексы, ключи и ограничения.

Следует начать с концептуального моделирования, двигаясь к последнему, который целесообразнее разобрать при непосредственной реализации ПО.

В рамках данной работы модель БД является простой по той причине, что она обособленна, предназначена лишь для хранения данных и не привязывается к какой-либо другой базе данных или таблицы. Данное хранилище обязательно для ранее разобранных прецедентов, поэтому играет важную роль в организации. В следствии ее простоты имеет смысл сразу представить логическую и физическую модель, что сделано рисунком 7.

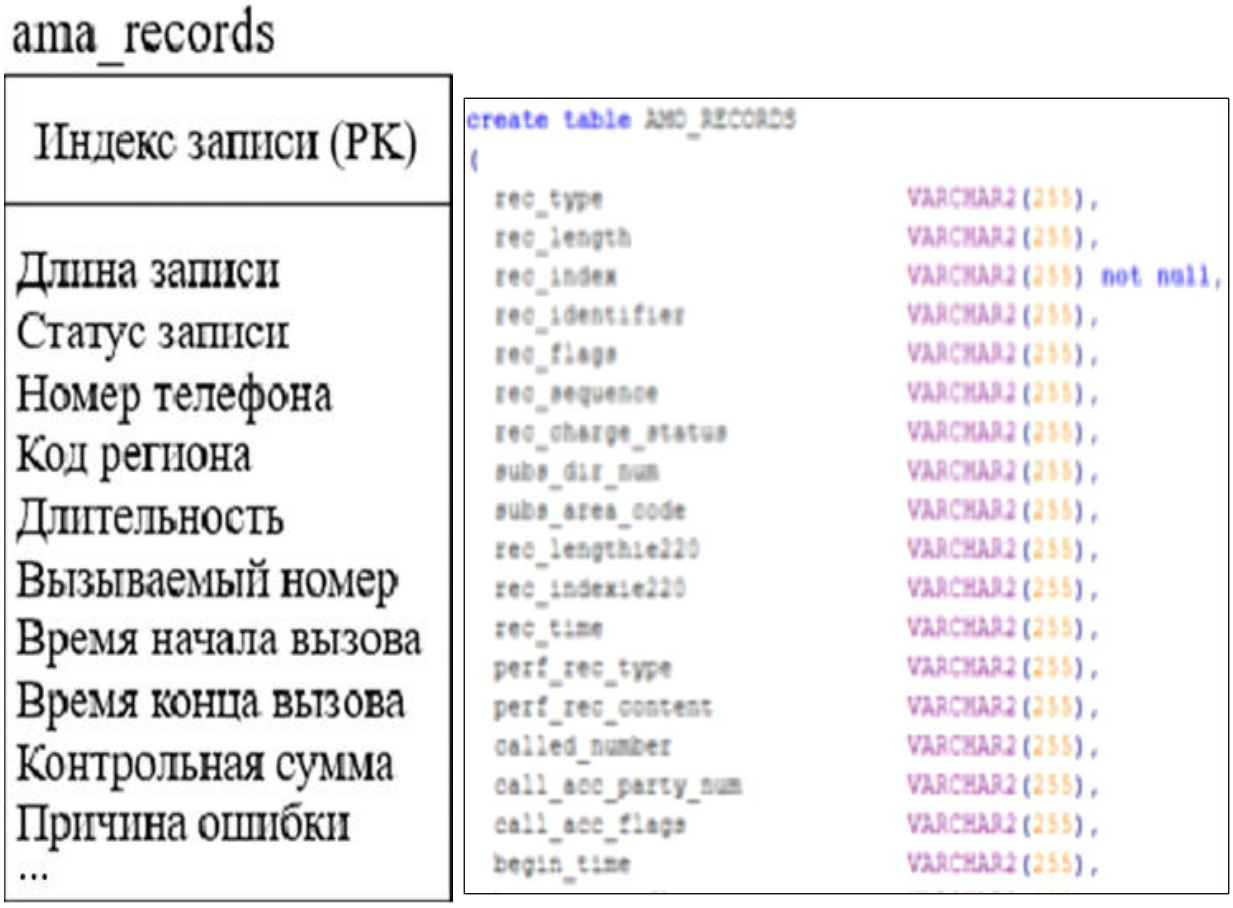


Рисунок 7 – Структура таблицы ama_records

Рисунок 7 отображает сильно упрощенную структуру таблицы ama_records, включающую малую долю из 212 атрибутов. Полная версия содержит дополнительные технические и бизнес-поля, необходимые того или иного прецедента что указывались ранее.

Из представленных полей можно выделить несколько:

– индекс записи является первичным ключом, а PK (Primary Key) указывает на это. Он представляет собой уникальный идентификатор каждой записи, который обязательно должен быть объявленным. Обычно для PK работает автоинкремент, механизм, который при добавлении новой записи автоматически увеличивает значение числового идентификатора, но в данном случае каждой полученной на АТС записи присвоен свой уникальный идентификатор;

– контрольная сумма является значением, которое служит для быстрой проверки подлинности и целостности файла путем сопоставления сумм и ее проверки до/после соответственно. Она рассчитывается по набору данных путем применения определенного алгоритма.

Во втором разделе были спроектированы логическая архитектура и структура базы данных создаваемого модуля. Построены диаграммы вариантов использования и последовательности, отражающие взаимодействие пользователей с системой. Выполнено логическое и физическое проектирование БД, учитывающее особенности хранения данных звонков. Кроме того, определена архитектура приложения, его компоненты и взаимодействие между ними, что обеспечило прочную основу для реализации отказоустойчивого и расширяемого программного модуля.

3. Разработка модуля обработки и анализа данных системы телефонии «ООО СамРЭК-Эксплуатация»

3.1 Выбор средств разработки

В ходе выяснения требований к программному обеспечению, которые были указаны в итоге главы 1.3, было выбрано что для разработки ПО на Windows 7 будет использоваться ЯП Python, библиотека Python-oracledb для работы с Oracle PL/SQL Developer, а для компиляции в исполняемый файл PyInstaller, который позволит не устанавливать интерпретатор Python на конечное оборудование.

Для использования Python-oracledb, исходя из документации, будет необходим интерпретатор Python версии 3.8 и выше.

Под Windows 7 последний поддерживаемый интерпретатор был Python версии 3.8, а также ее минорные обновления вроде 3.8.1, 3.8.2.

PyInstaller по документации работает с Python версии 3.8, кроме того, указывается что поддерживает официально Windows 8 и новее, но должен работать и с Windows 7.

Исходя из вышеуказанного для разработки подходит Python 3.8, который удовлетворяет всем требованиям.

3.2 Разработка приложения

Для начала нужно определить чем является файл .ama, пример которого представлен на рисунке 8. Это зашифрованный определенным образом файл, который несет в себе детализацию по всем вызовам, к примеру: кто кому позвонил, с какого номера на какой, сколько длился разговор, какие линии связи были задействованы. Данная информация может использоваться с разными целями. Например, информация о длительности разговора может послужить для отслеживания оставшегося количества времени на

неделю/месяц для того или иного номера и, в случае если время израсходовано, поставить ограничение на определенное время с целью заботы о средствах компании.

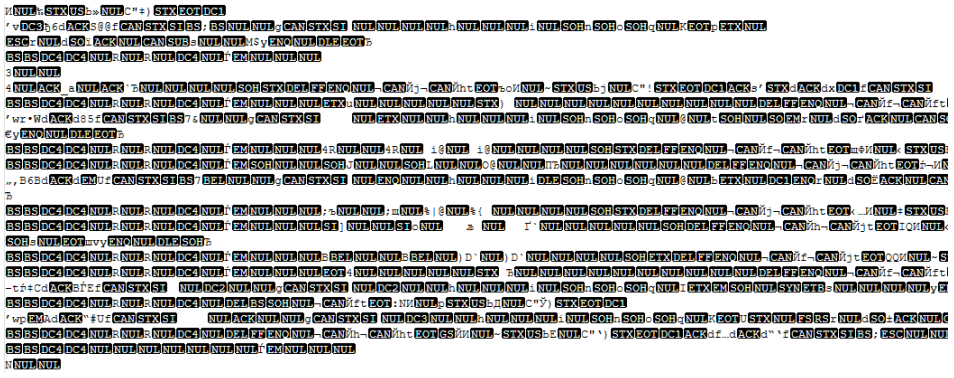


Рисунок 8 – Пример файла AMA

Попытка чтения файла, представленного на рисунке 8 – безрезультатна. Открытый файл имеет структуру, которую невозможно разобрать и полностью не читаемые данные.

В итоге необходимо написать модуль, который проанализирует и обработает данный тип файлов. Для этого инженером по связи, отдела внутренней эксплуатации, была выдана техническая документация.

Документация представляет из себя список используемых аббревиатур, разобранный список возможных записей в самом файле ama, указывая их идентификационный номер. Существуют номера записей (200, 210, 211, 212) и в диапазоне [100, 159]

Выданное техническое описание записей звонков имеет описание разбора данных различных форматов записей. В данной работе будет разрабатываться модуль для взаимодействия с бинарными файлами, которые подразумевают считывание определенное количество байт и его разбор соответственно описанной структуре какой-то записи.

Для примера разберем одну из таких записей в документации, а именно IE200, где 200 означает номер записи, а IE означает Information Element, представленную на рисунке 9.

Field	Posi.	Length	Field name								Format
1	1	1	Record type identifier (200)								bin
2	2	2	Record length								bin
3	4	4	Record index								bin
4	8	4	Call identifier								bin
5	12	3	F8	F7	F6	F5	F4	F3	F2	F1	bin
			F16	F15	F14	F13	F12	F11	F10	F9	
			Reserved			F21	F20	F19	F18	F17	
6	15	1	Sequence				Charge status				bin
7	16	1	Area code length			Subscriber number length					bin
8	17	n	Area code and subscriber number of the record owner								BCD

Рисунок 9 – IE200 в документации

На рисунке 9 можно увидеть наименования столбцов, которые гласят следующее:

– Field обозначает номер поля записи;

– Posi несет информацию о номере позиции поля;

– Length отображает сколько необходимо прочесть байтов для того или иного поля на позиции;

– Field Name это наименование считанного поля;

– Format показывает на то, в каком виде необходимо обработать поле.

Кроме того, Field с номером 5 требует считать 3 байта побитово (24 бита) и разобрать их на флаги (Flags), значения которых закодированы 1 или 0. Пример: 1000000101…1. Reserved – обозначает зарезервированное поле, бит или байт под что-то, которое следует пропустить. Иначе говоря, из 24 бита полезная информация лежит с 1 по 21 бит.

В вышеуказанных Flags лежит информация, указанная на рисунке 10.

- **Flags**

Flag	value	Meaning
F1	1	Call record
	0	Not a call record
F2	1	Facility usage (FAU) record
	0	Not a FAU record
F3	1	Facility input by subscriber (FAIS) record
	0	Not a FAIS record
F4	1	Call / service successful
	0	Call / service unsuccessful
F5	1	Call charging by meters
	0	No call charging by meters
F6	1	Call charging by AMA records
	0	No call charging by AMA records
F7	1	Immediate AMA
	0	Not immediate AMA
F8	1	Detailed billing (DEB)
	0	Not DEB
F9	1	Immediate DEB
	0	Not immediate DEB
F10	1	Originating calls meter observation (OMOB)
	0	Not OMOB
F11	1	Terminating calls meter observation (TMOB)
	0	Not TMOB
F12	1	Preventive meter observation (PMOB)
	0	Not PMOB
F13	1	Immediate PMOB
	0	Not immediate PMOB
F14	1	Reversed charging
	0	Not reversed charging
F15	1	Record of a call that was active at system switchover / failure The meaning of this flag depends on the product and configuration: - SI2000 V5: In non duplicated systems and in duplicated systems without the preservation of the calls at the system switchover the flag indicates a call released due to system restart or switchover. - SI2000 V5: In duplicated systems with the preservation of the calls at the system switch over the flag indicates a call preserved at system switchover. - SI3000 V6: The flag indicates a call preserved at system switchover.
	0	It is not a record of a call that was active at system switchover / failure
F16	1	Terminating charge recording
	0	It is not terminating charge recording
F17	1	A Centrex call
	0	Not a Centrex call
F18	1	Prepaid call / service
	0	Not a prepaid call / service
F19	1	Record for the purposes of Statistics CDRs
	0	Not for the purposes of statistics CDRs
F20	1	Online accounting for this call failed
	0	No information about Online Accounting
F21	1	FPH record (Free Phone Call)
	0	No FPH record

Рисунок 10 – Пример объяснения Flags

Рисунок 10 содержит информацию о значениях флагов с 1 по 21, каждый из которых кодирует определенные аспекты вызова или услуги. Например, значение 1 в Field 4 указывает на успешность вызова или услуги, а значение 0 на его неудачу. Field 5 определяет, применяется ли тарификация вызова, где значение 1 означает, что тарификация ведется, а Field 14 сигнализирует о реверсивной тарификации: значение 1 означает, что плата за вызов возлагается на принимающую сторону, а 0 что тарификация остается без изменений.

В Field 6, 7 следует отметить существование несколько подполей в одном байте записи, которые разбираются следующим образом: Для Field 6 необходимо прочитать 1 байт, или же, 8 бит, которые разбиваются пополам на Sequence и Charge Status в виде 1001 и 0001 соответственно.

Field 8 необходимо прочесть в виде BCD (Binary Coded Decimal), описание которого на рисунке 11.

8.1.2 Binary Coded Decimal (BCD)

The fields that contain binary coded decimal digits are marked with "BCD".

In one octet there are two digits (a binary coded digit is represented by four bits), whereat "*" is written as B and "#" as C. The digits in sequence in odd positions are written in the higher part of the octet, whereas the digits in sequence in even positions are written in the lower part of the octet. A field containing the number of digits in sequence is added to each field that contains binary coded decimal digits. If there are an odd number of digits in sequence, then the lower half of the last octet in the field has an undefined value.

Example 1:
Sequence of digits: 012345

Higher part of octet	Lower part of octet	
0	1	octet 1
2	3	octet 2
4	5	octet 3

Example 2:
Sequence of digits: 0123456

Higher part of octet	Lower part of octet	
0	1	octet 1
2	3	octet 2
4	5	octet 3
6	undefined value	octet 4

Рисунок 11 – Объяснение BCD

На рисунке 11 объясняется что BCD это поля, содержащие двоично-кодированные десятичные цифры. В одном октете находятся две цифры (двоично-кодированная цифра представлена четырьмя битами), при этом символ * обозначается как B, а # как C. Цифры последовательности в нечетных позициях записываются в старшей части октета, в то время как цифры в четных позициях записываются в младшей части октета. Каждому полю, содержащему двоично-кодированные десятичные цифры, добавляется поле, содержащее количество цифр в последовательности. Если количество цифр в последовательности нечетное, то в нижней половине последнего октета в поле будет неопределенное значение.

После завершения ознакомления с документацией можно сделать следующие выводы:

– необходимо предусмотреть способ чтения побайтово и перевод байта в целочисленное число;

– после перевода числа выходит значение, которое обязательно соответствует одному из номеров идентификаторов записей;

– сделать реализацию перевода байтов в биты;

– нужно обработать все встречающиеся флаги в соответствии с документацией;

– реализовать BCD считывание согласно данному примеру.

По итогу ознакомления реализуем модуль на ЯП Python с использованием библиотеки python-oracledb для составления SQL [3] запросов к БД.

Учитывая требования к программному обеспечению и требующиеся функции, получается следующая диаграмма компонентов, представленная рисунком 12.

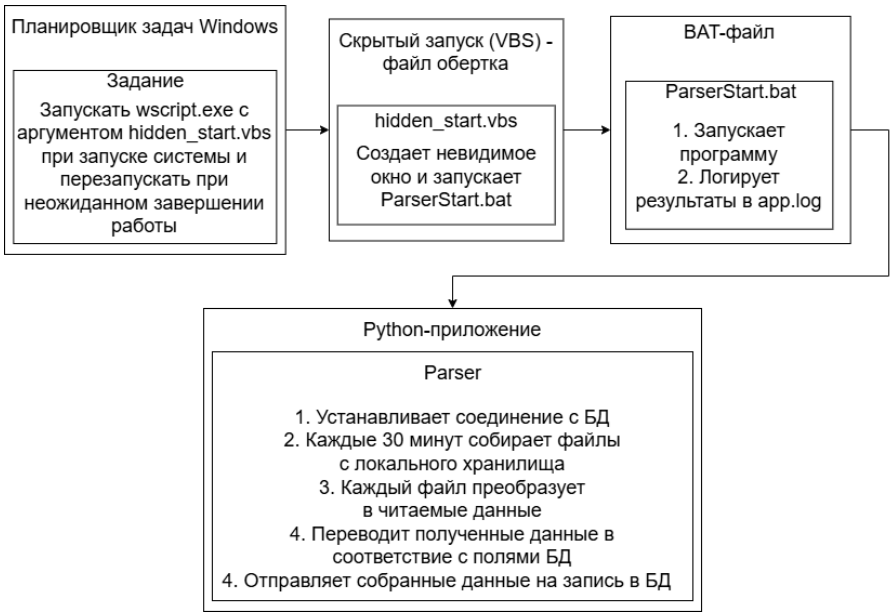


Рисунок 12 – Диаграмма компонентов

На рисунке 12 представлены компоненты, которые функционально разделены и запускаются каскадным способом, от планировщика Windows до Python-приложения:

– python-приложение устанавливает соединение с базой данных, каждые 30 минут собирает файлы с локального хранилища, каждый файл

преобразует в читаемые данные, переводит полученные данные в соответствие с полями базы данных, а затем отправляет данные на запись;

– приложение запускается через BAT файл (Batch file), которое отслеживает чтобы приложение при аварийной остановке заново запускалось;

– BAT файл обернут через VBS (Visual Basic Scripting), позволяя скрыть работу процесса отслеживающего состояние приложения;

– через планировщика задач Windows настраивается автозагрузка VBS файл при входе в систему и, кроме того, планировщик сам по себе следит за тем, чтобы процесс созданный VBS работал.

Сама архитектура python-приложения древовидная, представлена рисунком 13.

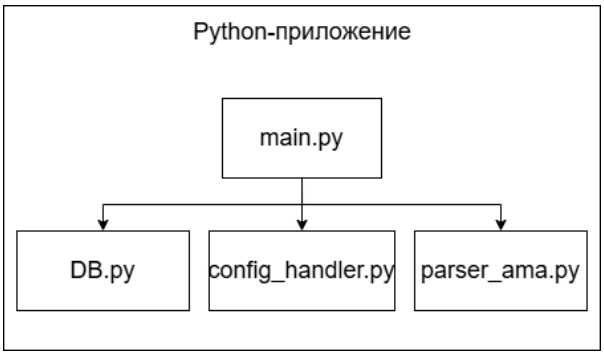


Рисунок 13 – Архитектура приложения

Архитектура с рисунка 13 состоит из 4 компонентов, которые объединяются в главном, main.py. Исходя из названий:

– DB.py отвечает за работу с БД (базой данных);

– parser_ama.py отвечает за обработку файлов .ama;

– config_handler.py отвечает за загрузку настроек работы модуля.

Модуль будет работать в среде, представленной рисунком 14.

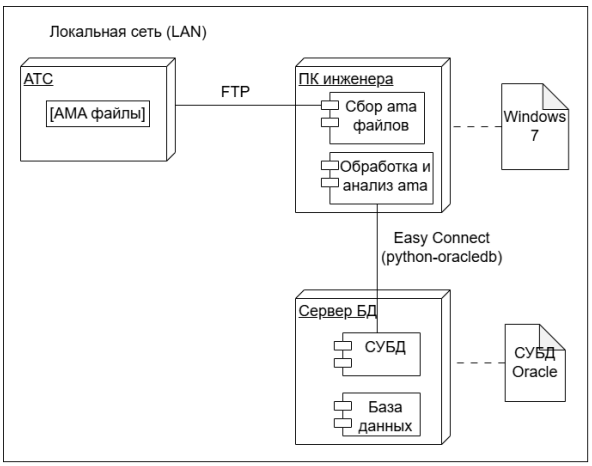


Рисунок 14 – Диаграмма развертывания

Рисунок 14 содержит диаграмму развертывания [10], которая отображает расположение компонентов и артефактов в развёрнутой системе, показывая, как части распределены по физическим или виртуальным ресурсам. Под артефактами подразумеваются программные элементы, такие как файлы или библиотеки. Следует отметить, что все части системы находятся в локальной сети.

На АТС хранятся файлы .ama, через FTP (File Transfer Protocol) собираются в локальное хранилище на ПК инженера под управлением Windows 7, там же обрабатываются модулем обработки и анализа, пересылаются через Easy Connect python-oracledb позволяющий подключиться к Oracle через TCP/IP (Transmission Control Protocol/Internet Protocol), указывая хост, порт и имя службы для подключения.

После этого следует перейти к самой разработке. Компонент Parser представлен рисунком 15. Как и следует из названия, будет анализировать синтаксис, иначе говоря парсить, файл по байтам и возвращать в результате полученные данные из зашифрованного файла ama.

Рисунок 15 – Реализация Parser

На рисунке 15 можно увидеть поля ama_fields, ama_fields_to_db_map, flags_map:

– ama_fields описывает все реализованные поля согласно документации. На рисунке 16 представлена часть их;

– ama_fields_to_db_map описывает отношения описанных полей в ama_fields к полям в ДБ. Данные отношения необходимы для конвертации. На рисунке 17 представлена часть отношений;

– flags_map несет в себе описание всех возможных полей с Flags для конвертации полученных значений битов, оговоренных ранее с примером, в какое-то осмысленное значение. Рисунком 18 описаны одни из множеств флагов.

Кроме того, можно заметить множество функций, которые будут рассмотрены подробно позже.

Рисунок 16 – Часть описания структур записей

Рисунком 16 затрагивается лишь часть элементов и на нем представлена структура в виде пар ключ-значение, где каждый ключ уникален. Данный вид иначе называется словарем.

Словарь позволяет отобразить иерархию записей, получить быстрый и удобный доступ к полям и помимо всего прочего словарь наиболее удобен для передачи данных. Изначально, при создании объекта Parser, все поля инициализированы None – пустотой.

Рисунок 17 – Часть описания отношений полей

Рисунок 17 описывает те же поля что затрагивались ранее. Данный словарь служит для того, чтобы перевести поля считанных данных в соответствующий таблице БД формат, который был установлен раньше физической моделью данных, рисунок 7.

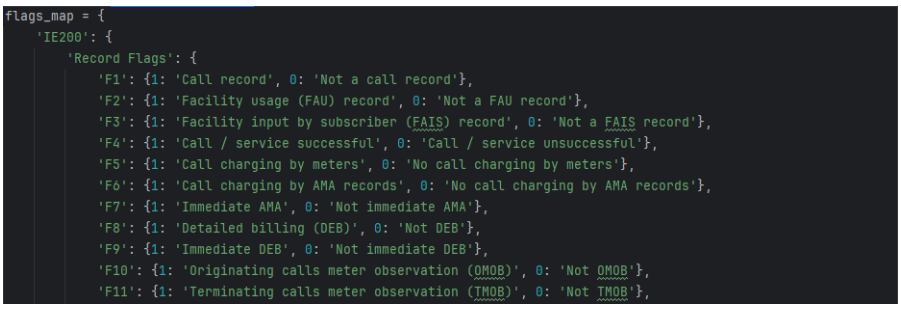


Рисунок 18 – Описание флагов на примере IE200

На рисунке 18 отображены некоторые флаги, принадлежащие записи под номером 200. В данной структуре, в зависимости от считанных данных с файла .ama, будут устанавливаться строго определенные значения.

Для реализации считывания байтов из файла необходимо создать функцию, которая будет принимать путь к директории с файлами и, перебирая каждый файл, идти от начала до конца каждого, считывая байты, определяя необходимую обработку для того или иного полученного значения. Она представлена рисунком 19.

```
def Parse(self):
    listFile = os.listdir(self.dir_path)

    amo = []
    for file in listFile:
        records = []
        first_f = False
        with open(os.path.join(self.dir_path, file), "rb") as f:
            while True:
                byte = self.logging_bytes_reader(f, length: 1)
                tt = self.to_int(byte)
                if tt == 200 and first_f:
                    self.FlagsConvert()
                    records.append(copy.deepcopy(self.amo_fields))
                    self.clearAmoFields()
                if tt == 0:
                    break
                first_f = True
                if tt in [200, 220] or tt in range(100, 159):
                    method_to_call = getattr(self, f'IE{tt}')
                    method_to_call(f)
                continue
        amo.append(records)
    return amo
```

Рисунок 19 – Реализация считывания байтов

На рисунке 19 реализован метод Parse(), который выполняет разбор бинарных файлов .ama.

Логика работы следующая:

– получение через библиотеку os [13] списка всех файлов в listFile, находящихся в указанной директории. Данная библиотека является встроенной и предназначена для взаимодействия с операционной системой;

– инициализация итогового списка ama для записей и для каждой записи создается временный список records;

– для каждого файла из списка открывается чтение по байтам при помощи флага rb - read byte, а затем происходит побайтовой чтение в byte, в tt перевод полученного значения в int через функцию. Обе функции представлены на рисунке 20.

– если встречается tt равный 200, то это либо первая запись, либо начало следующей, так как каждая запись имеет одно единственное начало - IE200.

– если же tt равен 0, то завершаем обработку файла.

– если tt не 200 и не 0, то проверка нахождения tt в допустимых значениях и, в случае успеха, через getattr находит функцию отвечающую за определенный номер записи, вызывает;

– сохраняет результат из records в ama, переходя к следующему файлу;

– по завершению возвращаем ama со всеми собранными данными.

```
def to_int(self, byte):
    return int.from_bytes(byte, byteorder='big')


def checksum(self, byte_seq):...


137 usages
def logging_bytes_reader(self, f, length):
    byte = f.read(length)
    return byte
```

Рисунок 20 – Чтение байта и перевод в int

Рисунок 20 представляет из себя преобразования байтов в целочисленные значения (to_int), используя встроенное преобразование, а также чтение указанного количества байт (logging_bytes_reader) с файла, указанного как f.

Как указывалось ранее, при успешной проверке значения int, пункт 4.c, должна вызываться одна из соответствующих функций, которые необходимо реализовать. Часть реализаций продемонстрирована на рисунке 21.

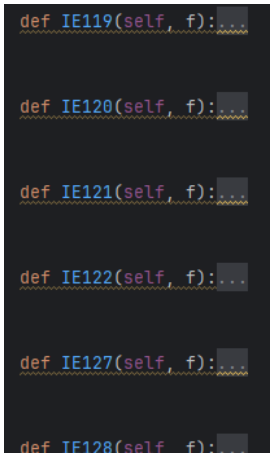
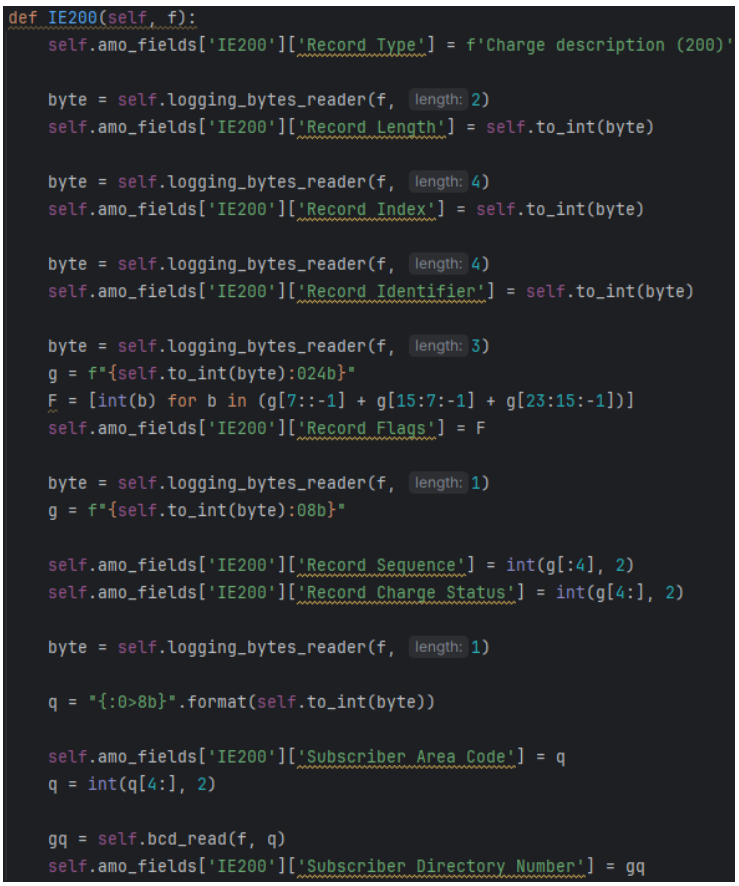


Рисунок 21 – Часть реализованных типов записей

На рисунке 21 показаны некоторые информационные элементы, которые однотипно принимают поток чтения с файла f.

Пользуясь же рисунком 22, разберем работу на примере той же записи IE200 с рисунка 9, так как все остальные работают идентичным образом, а эта запись охватывает все возможные варианты полей в записях.



```
def IE200(self, f):
    self.amo_fields['IE200']['Record Type'] = f'Charge description (200)'

    byte = self.logging_bytes_reader(f, length: 2)
    self.amo_fields['IE200']['Record Length'] = self.to_int(byte)

    byte = self.logging_bytes_reader(f, length: 4)
    self.amo_fields['IE200']['Record Index'] = self.to_int(byte)

    byte = self.logging_bytes_reader(f, length: 4)
    self.amo_fields['IE200']['Record Identifier'] = self.to_int(byte)

    byte = self.logging_bytes_reader(f, length: 3)
    g = f"{self.to_int(byte):024b}"
    F = [int(b) for b in (g[7::-1] + g[15:7:-1] + g[23:15:-1])]
    self.amo_fields['IE200']['Record Flags'] = F

    byte = self.logging_bytes_reader(f, length: 1)
    g = f"{self.to_int(byte):08b}"

    self.amo_fields['IE200']['Record Sequence'] = int(g[:4], 2)
    self.amo_fields['IE200']['Record Charge Status'] = int(g[4:], 2)

    byte = self.logging_bytes_reader(f, length: 1)

    q = "{:0>8b}".format(self.to_int(byte))

    self.amo_fields['IE200']['Subscriber Area Code'] = q
    q = int(q[4:], 2)

    gq = self.bcd_read(f, q)
    self.amo_fields['IE200']['Subscriber Directory Number'] = gq
```

Рисунок 22 – Программная реализация IE200

На рисунке 22 продемонстрирована реализация 1 из 31 возможных записей, которая соответствует требуемой логике с рисунка 7.

Здесь, согласовано с документацией, используя ранее разобранную функцию logging_bytes_reader поэтапно считывается количество байт, указанных для данной записи, переводится и записывается в поле ama_fields согласно номеру записи и типу поля.

Строка q = {:0>8b}.format(self.to_int(byte)) переводит байтовое представление в побитовое, согласно количеству считанных байт. То есть для 1 байта {:0>8b}, для 3 байт будет {:0>24b}.

К тому же присутствует gq = self.bcd_read(f, q), работу которого разберем пользуясь рисунком 23.

```python
def bcd_read(self, f, length):
    byte = self.logging_bytes_reader(f, (length + length % 2) // 2)
    stringr = ''
    for b in byte:
        stringr += ''.join(str((b & (1 << i)) and 1) for i in reversed(range(8)))
    g = []
    for i in range(0, (length) * 4, 4):
        if int(stringr[i:i + 4], 2) in range(10):
            g.append(str(int(stringr[i:i + 4], 2)))
        else:
            if int(stringr[i:i + 4], 2) == 11:
                g.append('*')
            elif int(stringr[i:i + 4], 2) == 12:
                g.append('#')

    return ''.join(g)
```

Рисунок 23 – Реализация bcd_read

На рисунке 23 представлен метод bcd_read, реализующий необходимую для корректной работы логику и опирающийся на рисунок 11. В данном методе вычисляется количество октет (length + length % 2) // 2, где в одной по два символа.

Затем выполняется извлечение каждого бита из байта и добавляется в строку в порядке от старшего к младшему

В for цикле length * 4 определяет общее количество битов в строке stringr, поскольку каждый октет состоит из 4 бит.

В цикле for с шагом 4 происходит итерация по строке stringr. Для каждой четверки битов stringr[i:i + 4] происходит попытка интерпретировать их как число в двоичной системе счисления int(stringr[i:i + 4], 2. Полученное число:

– если находится в диапазоне от 0 до 9, оно добавляется в список g как строковое представление числа;

– если равно 11, добавляется символ *;

– если равно 12, добавляется символ #.

Функция возвращает строку, полученную путем объединения всех элементов списка g в одну строку.

После завершения работ над компонентом парсера необходимо разработать компонент, который будет устанавливать соединение с БД на

сервере и записывать в заранее подготовленную таблицу новые полученные данные.

Для начала работы с базой данных необходимо установить соединение. Его установка представлена на рисунке 24.

```
def create_connection():
    # Параметры подключения к базе данных Oracle
    dsn_tns = cx_Oracle.makedsn( *args: 'localhost', '1521', sid='xe')
    try:
        connection = cx_Oracle.connect(user='SYSTEM', password='1400909cC', dsn=dsn_tns)
        print("Connection successful")
        return connection
    except cx_Oracle.Error as error:
        print(f"Error connecting to Oracle: {error}")
        return None
```

Рисунок 24 – Подключение к БД

На рисунке 24 метод create_connection возвращает объект соединения или None в случае ошибки.

Для начала определяем DSN (Data Source Name). Здесь:

– localhost это имя хоста, где запущена БД;

– 1521 является портом, на котором Oracle DB слушает подключения;

– sid = xe, определяющий идентификатор службы БД.

После чего пытаемся выполнить подключение, используя заданный DSN и имя пользователя, пароль. В случае успеха возвращаем подключение чтобы можно было с ним взаимодействовать.

Перед записью в БД необходимо преобразовать новые данные согласно их маскам, представленным на рисунках 16, 17, применяя ama_fields_to_db_map на ama_fields. Реализация представлена рисунком 25.

```
def extract_data(amo_fields, amo_fields_to_db_map):
    extracted_data = []

    for amo in amo_fields:
        for record in amo:
            extracted_record = {}
            for ie_key, fields in record.items():
                if ie_key not in extracted_record:
                    extracted_record[ie_key] = {}
                for field_name, field_value in fields.items():
                    db_field_name = amo_fields_to_db_map[ie_key].get(field_name)
                    if db_field_name:
                        extracted_record[ie_key][db_field_name] = field_value
            extracted_data.append(extracted_record)

    return extracted_data
```

Рисунок 25 – Конвертирование полей

На рисунке 25 функцией extract_data инициализируется список для хранения преобразованных данных.

Сначала происходит прохождение по каждой записи в ama_fields, вложенное прохождение по ключам информационных элементов и их полям.

Затем преобразование имен полей ama в имена полей БД, используя карту соответствия ama_fields_to_db_map.

После полного прохождения по всем ama, всем записям и полям, возвращаем extracted_data и уже эти данные записываем в БД.

В целях записи данных создается курсор, который является объектом, позволяющим выполнять SQL запросы и управлять набором результатов, полученных в результате выполнения. Данный процесс изложен рисунком 26.

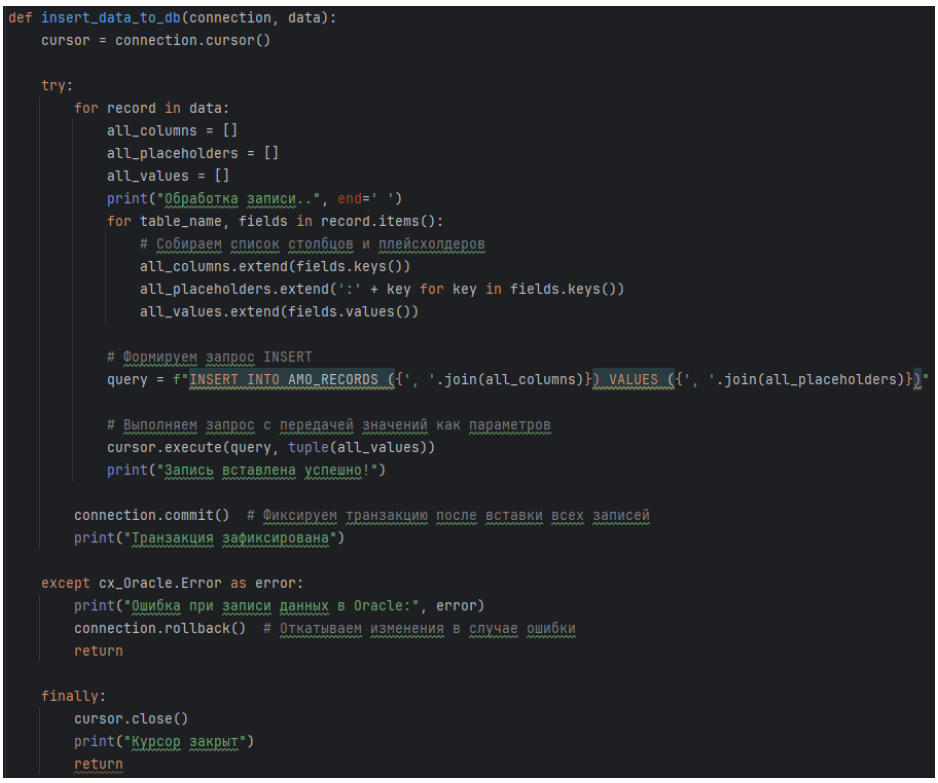


Рисунок 26 – Формирование INSERT SQL запроса

На рисунке 26 подключается курсор для взаимодействия с БД. Затем необходимо пройти по всем полученным сконвертированным записям из extract_data, которые поступают в функции в data. После чего составляется all_columns, all_placeholders, all_values:

– all_columns список имен столбцов собирающихся по ключам fields.keys();

– all_placeholders список заполнителей для будущих значений собирающихся по количеству значений в fields.keys();

– all_values список значений, которые будут вставлены в заполнители. Собирается через расширение all_values.extend() на fields.values().

После сбора всех полей из записи, подготавливается Insert запрос query, имеющий следующую логику: вставить в таблицу по колонкам значения all_values. Производится попытка записи, после чего фиксируется транзакция и закрывается курсор.

Когда все компоненты готовы необходимо их объединить в одно целое. Для этого создается main файл, представленный на рисунке 27, который будет инициализировать подключение к БД и создание объекта парсера.

```python
import os
import shutil
import time

sleep = 50
dir_newAma = 'C:\\Users\FS\Desktop\BY3\BA3\\ama_new'
dir_checkedAma = 'C:\\Users\FS\Desktop\BY3\BA3\\ama_checked'


if __name__ == '__main__':
    import DB as DB
    from ParserAmo import create_AmoParser

    AmoParser = create_AmoParser(dir_newAma)

    db = DB.create_connection()

    while True:
        data = AmoParser.Parse()
        if data:
            extracted_data = DB.extract_data(data, AmoParser.amo_fields_to_db_map)
            DB.insert_data_to_db(db, extracted_data)

            for file in os.listdir(dir_newAma):
                source_file = os.path.join(dir_newAma, file)
                target_file = os.path.join(dir_checkedAma, file)

                if os.path.isfile(source_file):
                    shutil.move(source_file, target_file)
            print(f'Обработанные файлы .amo были перемещены в директорию {dir_checkedAma}!')

        else:
            print(f'Новых файлов в директории {dir_newAma} не обнаружено')
        print("Ожидание времени...")
        time.sleep(sleep)

    db.close()
```

Рисунок 27 – main.py

Здесь на рисунке 27 собирается воедино DB.py, отвечающий за работу с БД и ParserAma.py, отвечающий за сам синтаксический анализ.

– Объявляется AmaParser, передавая директорию с файлами ama;

– Создается объект db, пытающийся подключиться к БД;

– После чего в бесконечном цикле парсим данные из файлов в директории и если вернулся не None объект, то делаем extract_data(), передавая полученные до этого данные с функции Parse() и словарь конвертации этих данных, доставая из AmaParser;

– Попытка записать extracted_data в БД;

– Перемещение всех обработанных файлов из папки amaNew в amaChecked, чтобы не записать их повторно и сохранить;

– Имитируется ожидание 30 минут после которых повторяется пункты с 3 по 6.

По итогу всех действий, согласно рассмотренной ранее архитектуры приложения, получается четыре компонента: работы с БД, парсера, инициализации настроек и главный объединяющий.

Через PyInstaller происходит компиляция главного компонента с добавлением необходимых модулей для python_oracledb. По итогу формируется .exe файл под названием AmaParser.

После чего, согласно диаграмме компонентов, представленной на рисунке 12, для запуска, корректного функционирования и автоматического восстановления работы приложения была реализована связка вспомогательных скриптов, написанных на языках BAT (Batch) [7] и VBS (Visual Basic Script) [20].

BAT-файл (ParserStart.bat) — командный скрипт, предназначенный для запуска Python-приложения, ведения логирования и обеспечения его непрерывной работы. Представлен на рисунке 28.

VBS-файл (hidden_start.vbs), представленный на рисунке 29, служит для скрытого запуска BAT-файла без отображения командной строки на экране пользователя, обеспечивая защиту от случайного завершения, и, вместе с тем, защищает от повторного запуска.

```
@echo off
chcp 65001

break off

set "EXE_FILE=AmaParser.exe"
set "LOG_FILE=app.log"
set "DELAY_SECONDS=10"

set "WORKING_DIR=%~dp0"
cd /d "%WORKING_DIR%"

:LOOP
powershell -Command "Add-Content -Path '%LOG_FILE%' -Value '[%DATE% %TIME%] Запуск %EXE_FILE%'"

start "ExeAppWindow" cmd /c ""%EXE_FILE%" >> "%LOG_FILE%" 2>&1"

"%EXE_FILE%" >> "%LOG_FILE%" 2>&1

powershell -Command "Add-Content -Path '%LOG_FILE%' -Value '[%DATE% %TIME%] Приложение завершено. Перезапуск через %DELAY_SECONDS% сек...'"

timeout /t %DELAY_SECONDS% /nobreak >nul
goto LOOP
```

Рисунок 28 – BAT скрипт

BAT скрипт с рисунка 28 включает в себя следующие ключевые этапы:

– Установка кодировки UTF-8 (chcp 65001) для корректного отображения русскоязычного текста в логах;

– Определение переменных: имя исполняемого файла (AmaParser.exe), лог-файла (app.log), интервала ожидания между перезапусками в случае аварийного завершения;

– Основной цикл (LOOP): запуск основного .exe файла, логирование начала и завершения работы, ожидание при аварийном завершении работы и повторный запуск. В случае аварийного завершения основной .exe файл будет автоматически перезапущен через заданное количество секунд (DELAY_SECONDS), обеспечивая отказоустойчивость модуля.

```
Set WshShell = CreateObject("WScript.Shell")

Set objWMIService = GetObject("winmgmts:\\.\root\cimv2")
Set colProcesses = objWMIService.ExecQuery("SELECT * FROM Win32_Process WHERE Name = 'wscript.exe'")

For Each objProcess In colProcesses
    If InStr(objProcess.CommandLine, "script.vbs") > 0 Then
        WScript.Echo "Этот скрипт уже запущен."
        WScript.Quit
    End If
Next

batFile = WScript.CreateObject("Scripting.FileSystemObject").GetFile(WScript.ScriptFullName).ParentFolder.Path & "\ParserStart.bat"

WshShell.Run "cmd.exe /c """ & batFile & """", 0, False
```

Рисунок 29 – VBS скрипт

VBS скрипта с рисунка 29 включает в себя следующее:

– Создание оболочки командной строки: Сначала создаётся объект оболочки WScript.Shell, позволяющий запускать процессы в Windows;

– Подключение к службе WMI (Windows Management Instrumentation), которая используется для получения информации о запущенных процессах;

– Поиск уже запущенного экземпляра script.vbs через запрос всех процессов wscript.exe (исполнитель VBS-скриптов), после чего

производится проверка командной строки каждого процесса на наличие script.vbs;

– Определение пути до BAT-файла (ParserStart.bat), который должен находиться в той же папке, что и script.vbs;

– Скрытый запуск BAT-файла через cmd.exe, при этом окно командной строки не отображается (0 означает скрытый режим).

По итогу, для автоматического запуска при входе в систему, используется Планировщик задач Windows (Task Scheduler). Через него создаётся простая задача, которая обеспечивает запуск VBS-скрипта сразу после входа пользователя в систему. Это позволяет обеспечить непрерывную и автономную работу модуля без необходимости ручного запуска.

Настройка задачи в планировщике включает следующее:

– открывается планировщик задач, создаётся новая задача, например, StartAmaParser;

– выбирается триггер устанавливающий запуск при входе пользователя в систему, чтобы скрипт начинал работу автоматически;

– действием выбирается запуск программы, где в качестве исполняемого файла указывается путь до wscript.exe, встроенного интерпретатора для VBS, а в качестве аргумента сам написанный VBS скрипт.

Таким образом, данная система запуска представляет собой устойчивый механизм автоматизации, обеспечивающий:

– автоматический старт приложения при входе пользователя в систему (при добавлении script.vbs в автозагрузку Windows);

– защиту от множественного запуска;

– устойчивость к сбоям приложения за счет самоперезапуска;

– логирование работы и ошибок для последующего анализа.

Такой подход особенно актуален в условиях, когда программное обеспечение должно функционировать непрерывно на оборудовании без

постоянного участия пользователя, что соответствует выдвинутым требованиям.

3.3 Тестирование приложения

После завершения разработки модуля обработки и анализа данных системы телефонии необходимо провести его тестирование. Целью тестирования является проверка корректности работы всех компонентов приложения, его устойчивости к сбоям, а также выявление возможных ошибок и нестабильностей.

Тестирование [1] проводилось по следующим направлениям:

а) Функциональное тестирование, рисунок 30. Проверялась корректность выполнения основных функций модуля:

- автоматическое считывание и обработка .ama файлов;
- корректное преобразование бинарных данных в структуру, соответствующую полям БД;
- успешная запись данных в базу;
- перемещение обработанных файлов в папку архива;
- корректная работа при наличии повреждённых или отсутствующих файлов.

б) Тестирование устойчивости к сбоям, рисунок 31 и 32. Проводилось, моделируя следующие сценарии:

- аварийное завершение AmaParser.exe;
- отключение подключения к базе данных в момент записи;
- удаление конфигурационного файла.

в) Нагрузочное тестирование. Было проведено при одновременной обработке большого объёма данных. В директорию было помещено 45 .ama файлов.

г) Логирование, рисунок 33. Проверялась полнота и читаемость логов:

– фиксируются события запуска, завершения, ошибки и сообщения от компонентов;

– указывается точное время возникновения событий.

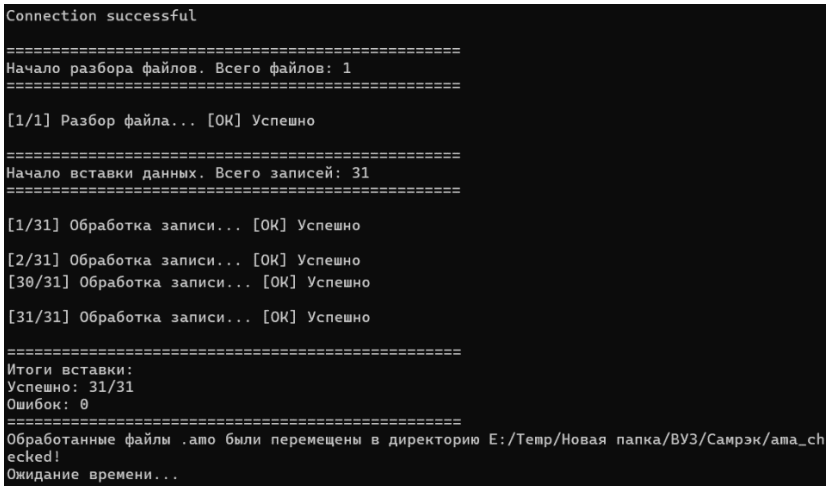


Рисунок 30 – Функциональное тестирование

На рисунке 30 представлен результат функционального тестирования, где все заявленные функции выполняются корректно. Ошибки при обработке повреждённых файлов обрабатываются исключениями, и информация о сбое фиксируется в лог-файле.

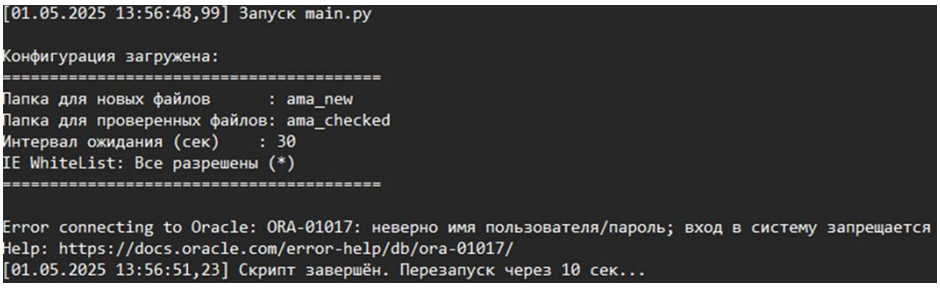


Рисунок 31 – Тестирование устойчивости при отключении подключения

На рисунке 31 BAT-скрипт корректно перезапускает приложение при отключении подключения к базе данных.

```
13.04.2025 16:55:54,94] Запуск main.py
онфиг файл не найден по адресу: config.ini
13.04.2025 16:55:55,65] Скрипт завершён. Перезапуск через 10 сек...
```

Рисунок 32 – Тестирование устойчивости при удалении конфигурационного файла

На рисунке 32 BAT-скрипт корректно перезапускает приложение при удалении конфигурационного файла.

Ошибки фиксируются в app.log, данные не теряются. В случае отключения БД приложение делает повторную попытку соединения при следующем запуске. При отсутствии конфигурационного файла модуль останавливает работу и выводит соответствующее сообщение в лог.

При нагрузочном тестировании модуль успешно обработал все файлы без пропусков и зависаний. Время обработки одного файла с записью в базу данных в среднем составило 0.02–0.035 секунд в зависимости от количества записей внутри файла. Обработка всех файлов заняла примерно 1.44 секунды.

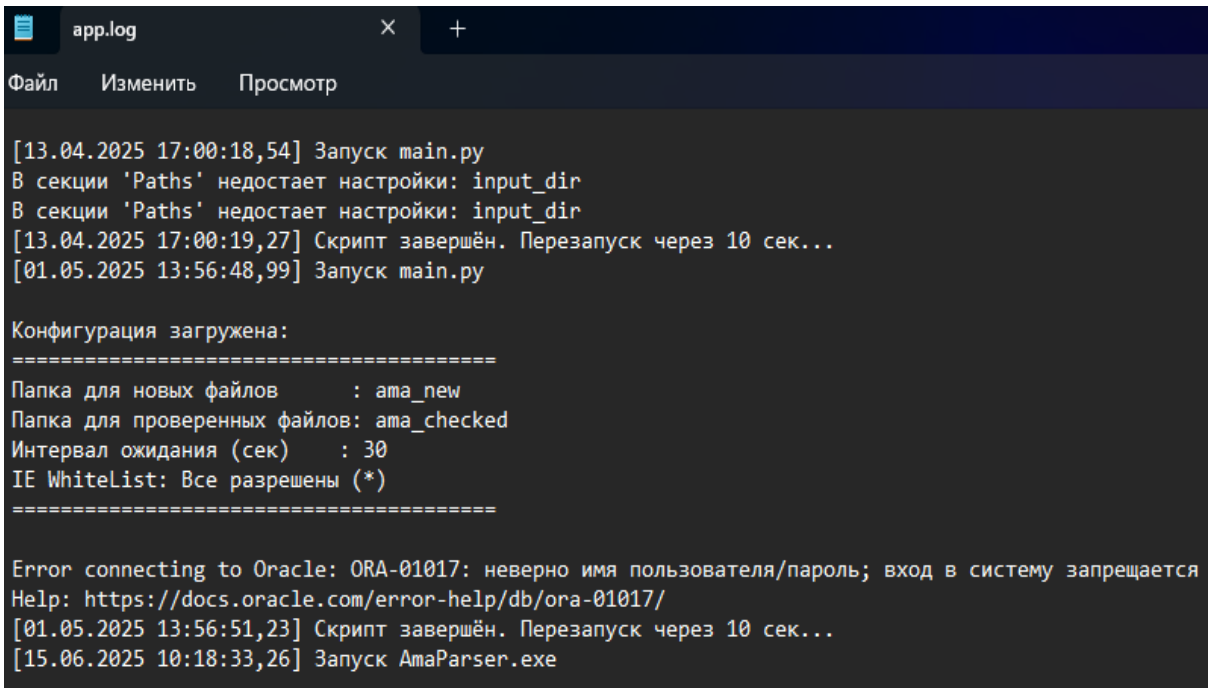


Рисунок 33 – Тестирование логирования

Логирование, представленное на рисунке 33, работает корректно, позволяет точно восстановить хронологию событий и выявить причины возможных сбоев.

Результаты тестирования подтвердили работоспособность модуля в условиях, приближенных к реальной эксплуатации. Все компоненты приложения соответствуют заявленным требованиям. Система демонстрирует стабильную и предсказуемую работу, устойчива к сбоям.

В третьем разделе была создана полноценная система, включающая в себя обработку бинарных .ama файлов, преобразование данных по предоставленной технической документации и запись информации в БД Oracle. Разработка была выполнена на языке Python с использованием специализированных библиотек. Обеспечено логирование, самоперезапуск, автозагрузка и отказоустойчивость. Проведено тестирование модулей, подтвердившее корректность работы и соответствие заявленным требованиям. Разработка завершилась созданием полноценного автономного модуля, готового к внедрению.

Заключение

В рамках выпускной квалификационной работы был разработан модуль обработки и анализа системы телефонии для ООО «СамРЭК-Эксплуатация», который внес вклад в перестройку внутреннего бизнес-процесса компании, касающегося работы с файлами типа ama. Была успешна решена актуальная задача по автоматизации обработки данных телефонии.

Модуль удовлетворяет установленным требованиям, устраняет недостатки текущего процесса, такие как ручной труд, человеческий фактор, поправляя низкую эффективность и риски накопления данных.

Работа началась с анализа существующей проблемы, изучения применяемого оборудования и формулирования требований к разрабатываемому программному решению.

Взаимодействие с сотрудниками телекоммуникационного отдела и внимательное изучение технической документации позволили определить функциональные рамки ПО.

Результатом работы является модуль обработки и анализа данных системы телефонии, включающий компоненты с рисунка 13.

Выпускная квалификационная работа для ООО «СамРЭК-Эсплуатация» ознакомила с телекоммуникационными сетями и их работой, подтянула в работе с документацией, дала опыт работы. Кроме того, ознакомила с организационной структурой и внутренними процессами организации.

Полученные знания и навыки в области проектирования архитектуры и работа с БД Oracle стали ценным практическим опытом. Реализация проекта продемонстрировала важность предварительной аналитики.

Разработанное решение внесло вклад для дальнейшего развития информационной системы предприятия и имеет потенциал совершенствоваться в разных направлениях. Вроде интеграции с другими системами или применение технологий машинного обучения для прогнозирования нагрузки на телефонию.

Список используемой литературы и используемых источников

1. Виды тестирования по целям и задачам [электронный ресурс] URL: https://vvrozhkova.github.io/vidy-testirovaniya-po-czelyam-i-zadacham/ (дата обращения 08.06.2025)
2. DFD (Data Flow Diagram) Диаграммы - зачем они нужны и какие бывают [электронный ресурс] URL: https://habr.com/ru/articles/668684/ (дата обращения 02.06.2025)
3. Как думать на SQL? [электронный ресурс] URL: https://habr.com/ru/articles/305926/ (дата обращения 28.02.2025)
4. Логическая модель базы данных: что это и как её создать [электронный ресурс] URL: https://sky.pro/wiki/sql/logicheskaya-model-bazy-dannyh-chto-eto-i-kak-eyo-sozdat/ (дата обращения 28.02.2025)
5. Физическая модель базы данных [электронный ресурс] URL: https://neerc.ifmo.ru/wiki/index.php?title=Физическая_модель_базы_данных (дата обращения 28.02.2025)
6. Что такое нотация BPMN 2.0 и как она помогает смоделировать бизнес-процесс [электронный ресурс] URL: https://practicum.yandex.ru/blog/notaciya-bpmn-dlya-biznes-processov/ (дата обращения 22.05.2025)
7. An A-Z Index of Windows CMD commands [электронный ресурс] URL: https://ss64.com/nt/ (дата обращения 07.06.2025)
8. BCD or Binary Coded Decimal [электронный ресурс] URL: https://www.geeksforgeeks.org/dsa/bcd-or-binary-coded-decimal/ (дата обращения 01.03.2025)
9. Database Designer’s Guide [электронный ресурс] URL: https://circle.visual-paradigm.com/docs/database-design-engineering/database-designers-guide/ (дата обращения 28.02.2025)

10. Deployment Diagram Tutorial [электронный ресурс] URL: https://online.visual-paradigm.com/ru/diagrams/tutorials/deployment-diagram-tutorial/ (дата обращения 05.06.2025)

11. Introduction of ER Model [электронный ресурс] URL: https://www.geeksforgeeks.org/dbms/introduction-of-er-model/ (дата обращения 28.02.2025)

12. Oracle PL/SQL Developer. Documentation [электронный ресурс] URL: https://docs.oracle.com/cd/F52398_01/plsql-developer/pl-sql-developers-guide.pdf (дата обращения 18.02.2025)

13. os – Miscellaneous operating system interface [электронный ресурс] URL: https://docs.python.org/3/library/os.html (дата обращения 03.03.2025)

14. PyInstaller. Manual [электронный ресурс] URL: https://pyinstaller.org/en/stable/index.html (дата обращения 23.02.2025)

15. Python. Documentation [электронный ресурс] URL: https://docs.python.org/3/index.html (дата обращения 24.02.2025)

16. Python-oracledb. Documentation [электронный ресурс] URL: https://python-oracledb.readthedocs.io/en/latest/ (дата обращения 27.02.2025)

17. SQLAlchemy 2.0 Documentation [электронный ресурс] URL: https://docs.sqlalchemy.org/en/20/ (дата обращения 27.02.2025)

18. Sequence Diagrams - Unified Modeling Language (UML) [электронный ресурс] URL: https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/ (дата обращения 20.02.2025)

19. Use Case Diagram - Unified Modeling Language (UML) [электронный ресурс] URL: https://www.geeksforgeeks.org/use-case-diagram/ (дата обращения 20.02.2025)

20. VBScript Language Reference [электронный ресурс] URL: https://documentation.help/MS-Office-VBScript/VBSTOC.htm (дата обращения 07.06.2025)