МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

База данных оператора сотовой связи с использованием технологии Advanced Analytics

Выполнил студент Бельский А. А.

(Ф.И.О.)

Руководитель проекта ст. п. Блинова Е. А.

(учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В .

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Минск 2021

Реферат

Пояснительная записка курсового проекта содержит 46 страниц пояснительной записки, 4 источника литературы, 7 приложений.

.NET Framework, Windows Presentation Foundation (WPF), Oracle SQL Server, ADO.NET, Dapper.

Цель курсового проекта – разработка базы данных и приложения, которые обеспечат простое и удобное взаимодействие между собой, что позволит пользователю управлять своими договорами с оператором сотовой связи по предоставляемым им услугам.

В первой главе проводится аналитический обзор схожих приложений по тематике курсового проекта.

Вторая глава посвящена процессу проектирования системы и описание технологий, использованных во время выполнения проекта.

В третьей главе описывается процесс разработки, принципы функционирования проекта.

В четвёртой главе описано тестирование, проверка работоспособности и анализ данных системы.

В заключении приведены результаты проделанной работы.

**Содержание**

[Реферат 2](#_Toc71921509)

[Введение 4](#_Toc71921510)

[1 Обзор технических решений и литературных источников 5](#_Toc71921511)

[2 Проектирование и разработка базы данных 7](#_Toc71921512)

[2.1 Спецификация функциональных требований 7](#_Toc71921513)

[2.2 Разработка модели базы данных 8](#_Toc71921514)

[2.3 Хранимые процедуры 10](#_Toc71921515)

[2.4 Индексы 11](#_Toc71921516)

[2.5 Планировщики задач 11](#_Toc71921517)

[2.6 Триггеры 11](#_Toc71921518)

[2.7 Представления 12](#_Toc71921519)

[2.8 Реализация технологии 12](#_Toc71921520)

[2.9 Экспорт и импорт данных 15](#_Toc71921521)

[3 Проектирование и разработка программного средства 17](#_Toc71921522)

[4 Тестирование и проверка работоспособности 21](#_Toc71921523)

[4.1 Тестирование производительности базы данных 21](#_Toc71921524)

[4.2 Проверка работоспособности программного средства 22](#_Toc71921525)

[Заключение 29](#_Toc71921526)

[Список литературы 30](#_Toc71921527)

[Приложение А 31](#_Toc71921528)

[Приложение Б 32](#_Toc71921529)

[Приложение В 36](#_Toc71921530)

[Приложение Г 42](#_Toc71921531)

[Приложение Д 43](#_Toc71921532)

[Приложение Е 45](#_Toc71921533)

[Приложение Ж 46](#_Toc71921534)

Введение

В данной записке приведено описание реляционной базы данных и desktop-приложения, разработанных в соответствии с заданием на курсовое проектирование по теме «База данных оператора сотовой связи с использованием технологии Advanced Analytics».

Задачей данной курсовой работы является разработка базы данных, предназначенной для ведения учёта сотрудников, клиентов, их договоров с сопутствующей информацией, а также для приложения «Менеджер услуг сотовой связи». Приложение будет обладать следующими функционалом:

* авторизация пользователей;
* разделение пользователя на сотрудника и клиента;
* просмотр, удаление и добавление договоров клиентом;
* просмотр, удаление и добавление услуг клиентом;
* просмотр номеров телефона, привязанных к договорам клиентом;
* просмотр и добавление рекомендованных тарифов и услуг клиентом;
* просмотр информации об аккаунте клиентом;
* просмотр клиентов и договоров, которые с ними были подписаны, авторизованным сотрудником.

Система управления базами данных (СУБД) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Для разработки и управления базой данных курсового проекта использовалась система управления реляционными базами данных Oracle SQL Server [1], в связи с требованием курсовой работы.

1 Обзор технических решений и литературных источников

Для составления технических требований к проекту был проведен анализ похожих программных средств.

Приложение «Мой A1»

Мой А1 – приложение для абонентов сети А1 для управления своим аккаунтом, где бы не находились в любой момент времени.

Приложение «Мой А1» позволяет получать актуальные сообщения о предложениях и акциях компании А1, информацию о своем балансе и остатках минут, SMS, MMS и интернет-трафика, подключать и отключать услуги, изменять свой тарифный план и др. (рисунок 1.1).

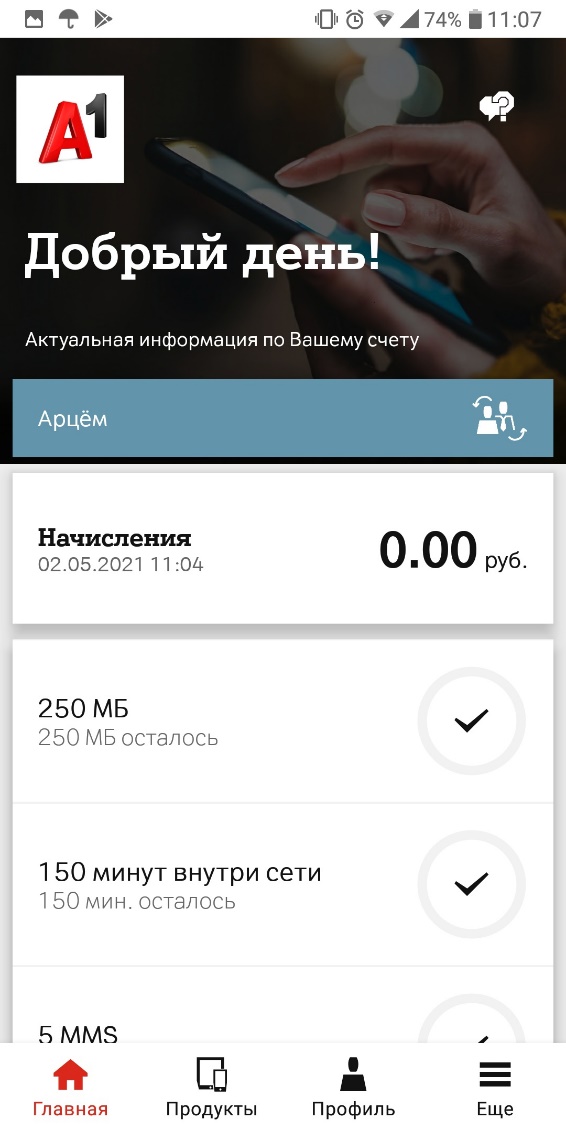


Рисунок 1.1 – Окно приложения «Мой А1»

Приложение «Мой МТС».

Приложение «Мой МТС» — это инструмент самообслуживания абонента, позволяющий получать актуальные данные о состоянии лицевого счета и остатках предоплаченных минут, SMS, MMS и интернет-трафика, подключать/отключать услуги, производить смену тарифного плана и производить другие действия в рамках своего абонентского номера (рисунок 1.2).

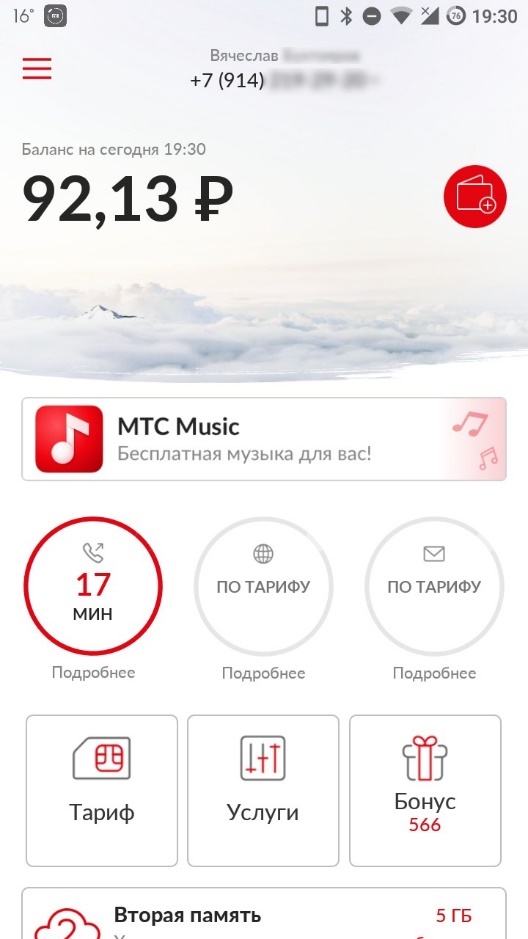


Рисунок 1.2 – Окно приложения «Мой МТС»

Все вышеперечисленные приложения обладают схожим функционалом для управления тарифным планом, услугами и контролем баланса.

2 Проектирование и разработка базы данных

При разработке курсового проекта понадобились следующие объекты:

1. Таблицы;
2. Процедуры, хранимые в пакетах;
3. Индексы;
4. Планировщики с задачами;
5. Триггеры;
6. Представления.

# 2.1 Спецификация функциональных требований

Рассмотрим возможности, которые предоставляет разработанное программное средство. Для наглядности демонстрации использовались UML -диаграммы.

UML – уникальный язык моделирования (Unified Modeling Language) – это система обозначений, которую можно применять для объектно-ориентированного анализа и проектирования. Его можно использовать для визуализации, спецификации, конструирования и документирования программных систем.

Неавторизованный пользователь ничего не сможет сделать в приложении, только авторизоваться.

Если клиент-пользователь авторизуется, то будет иметь возможности, представленные на рисунке 2.1.1.

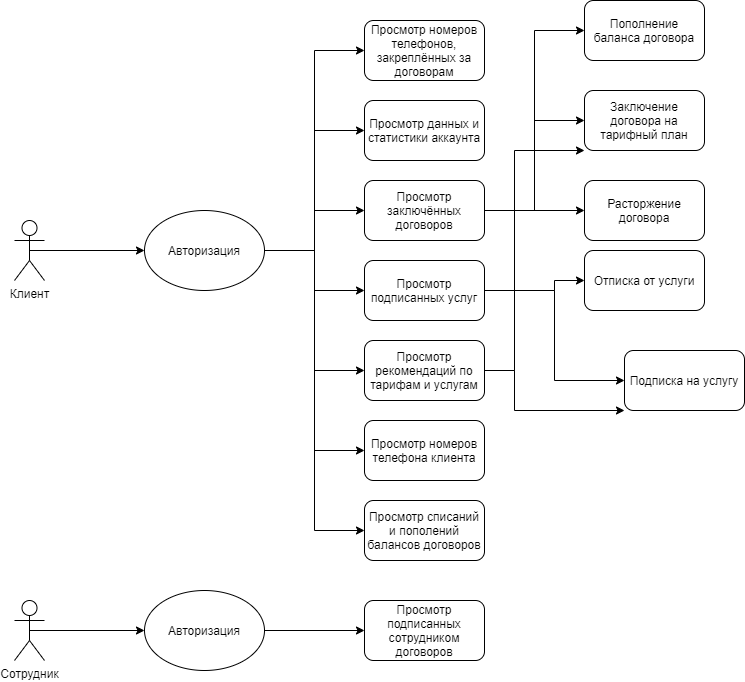


Рисунок 2.1.1 – UML диаграмма вариантов использования для авторизованного клиента

Также в приложение можно войти как сотрудник. Тогда он сможет только просмотреть подписанные с клиентами договора.

Диаграмма вариантов использования для сотрудника отображена на рисунке 2.1.2.

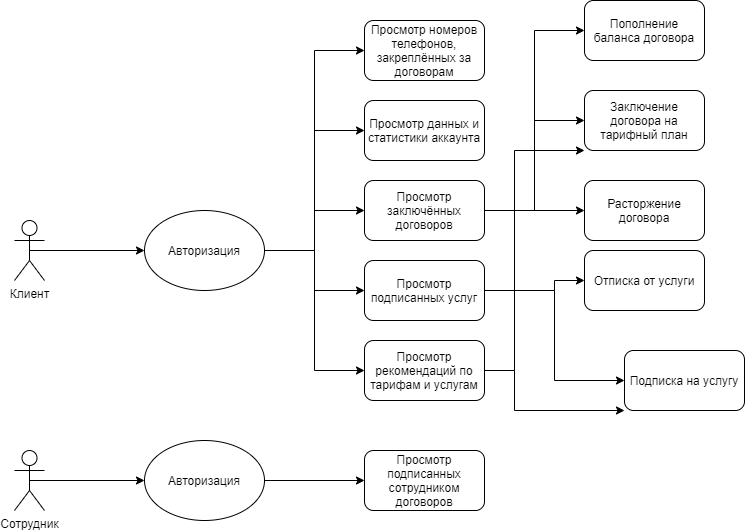


Рисунок 2.1.2 – UML диаграмма вариантов использования для сотрудника

После рассмотрения функциональных требований стоит рассмотреть модель нашей базы данных.

# 2.2 Разработка модели базы данных

Для реализации базы данных для оператора сотовой связи было разработано 13 таблиц. Диаграмма базы со структурой связей представлена в приложении А, а в приложении Б представлены скрипты создания БД.

Таблица Account предназначена для хранения информации о зарегистрированных пользователях, состоит из 3 столбцов:

* login – логин аккаунта, уникальный для каждого пользователя, тип nvrachar2(50), первичный ключ;
* hash\_password – хэш пароля пользователя, захэшированного алгоритмом MD5, тип nvarchar2(32);
* access\_level – уровень доступа пользователя в приложении, тип int.

Таблица Account\_event предназначена для хранения информации о событиях, связанных с аккаунтами (добавление, вход, обновление, удаление), состоит из 4 столбцов:

* id – идентификатор события, тип int, первичный ключ;
* account\_login – логин аккаунта, тип nvarchar2(50);
* event\_datetime – время события, тип timestamp;
* message – сообщение, сопровождающее событие, тип nvarchar2(200).

Таблица Post предназначена для хранения информации о должностях сотрудников, состоит из 3 столбцов:

* id – идентификатор должности, тип int, первичный ключ;
* post\_name – должность, тип nvarchar2(50);
* category – категория должности, тип nvarchar2(50).

Таблица Client предназначена для хранения информации о клиентах, состоит из 4 столбцов:

* id – идентификатор клиента, тип int, первичный ключ;
* full\_name – полное имя клиента, тип nvarchar2(50);
* passport\_number – номер паспорта клиента, тип nvarchar2(12);
* account\_login – логин аккаунта, тип nvarchar2(50), внешний ключ.

Таблица Service\_description предназначена для хранения информации об описании услуг, состоит из 3 столбцов:

* id – идентификатор описания услуги, тип int, первичный ключ;
* service\_name – имя услуги, тип nvarchar2(50);
* service\_description – описание услуги, тип nvarchar2(500).

Таблица Tariff\_plan предназначена для хранения информации об тарифных планах, состоит из 3 столбцов:

* id – идентификатор тарифного плана, тип int, первичный ключ;
* tariff\_name – имя тарифа, тип nvarchar2(50);
* tariff\_amount – стоимость тарифа, тип float(126).

Таблица Employee предназначена для хранения информации о сотрудниках, состоит из 4 столбцов:

* id – идентификатор сотрудника, тип int, первичный ключ;
* full\_name – полное имя сотрудника, тип nvarchar2(50);
* post\_id – идентификатор должности сотрудника, тип int, внешний ключ;
* account\_login – логин аккаунта, тип nvarchar2(50), внешний ключ.

Таблица Contract предназначена для хранения информации о договорах, состоит из 5 столбцов:

* id – идентификатор сотрудника, тип int, первичный ключ;
* tariff\_id – идентификатор тарифа, тип int, внешний ключ;
* client\_id – идентификатор клиента, тип int, внешний ключ;
* employee\_id – идентификатор сотрудника, тип int, внешний ключ;
* signing\_datetime – время подписания, тип timestamp.

Таблица Phone\_number предназначена для хранения информации о номерах телефонов, состоит из 3 столбцов:

* id – идентификатор номера телефона, тип int, первичный ключ;
* phone\_number – номер телефона, тип nvarchar2(14), соответствует регулярному выражению мобильного телефона;
* contract\_id – идентификатор контракта, тип int, внешний ключ.

Таблица Call предназначена для хранения информации о звонках, состоит из 5 столбцов:

* id – идентификатор звонка, тип int, первичный ключ;
* contract\_id – идентификатор контракта, тип int, внешний ключ;
* to\_phone\_number – номер телефона, к которому происходит звонок, тип nvarchar2(14), соответствует регулярному выражению мобильного телефона;
* talk\_time – продолжительность разговора, тип interval day to second;
* call\_datetime – время звонка, тип timestamp.

Таблица Payment предназначена для хранения информации о платежах, состоит из 4 столбцов:

* id – идентификатор оплаты, тип int, первичный ключ;
* contract\_id – идентификатор контракта, тип int, внешний ключ;
* payment\_amount – размер оплаты, тип float(126);
* payment\_datetime – время оплаты, тип timestamp.

Таблица Debit предназначена для хранения информации о списаниях, состоит из 5 столбцов:

* id – идентификатор оплаты, тип int, первичный ключ;
* contract\_id – идентификатор контракта, тип int, внешний ключ;
* debit\_amount – размер списания, тип float(126);
* debit\_datetime – время оплаты, тип timestamp;
* reason – время оплаты, тип nvarchar2(250).

Таблица Service предназначена для хранения информации о существующих сервисах, состоит из 6 столбцов:

* id – идентификатор оплаты, тип int, первичный ключ;
* contract\_id – идентификатор контракта, тип int, внешний ключ;
* description\_id – идентификатор описания, тип int, внешний ключ;
* service\_amount – стоимость сервиса, тип float(126);
* connection\_datetime – время подключения, тип timestamp;
* disconnection\_datetime – время отключения, тип timestamp.

# 2.3 Хранимые процедуры

Хранимая процедура – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере.

При разработке курсового проекта были созданы процедуры для следующих целей:

1. Добавление строки (для каждой таблицы);
2. Изменение строки (для каждой таблицы);
3. Удаление строки (для каждой таблицы);
4. Получение строки (для каждой таблицы, кроме таблиц аккаунтов);
5. Получение всех строк (для каждой таблицы, кроме таблиц аккаунтов);
6. Получение уровня доступа по логину и паролю аккаунта;
7. Получение клиента по логину аккаунта;
8. Получение сотрудника по логину аккаунта;
9. Получение услуг клиента;
10. Получение договоров клиента;
11. Получение договоров, которые подписал сотрудник;
12. Получение рекомендаций тарифов клиента;
13. Получение баланса контракта;
14. Получение услуг контракта;
15. Получение номеров телефонов, прикреплённых за контрактом;
16. Получение списаний, прикреплённых за контрактом;
17. Получение пополнений, прикреплённых за контрактом;
18. Получение рекомендаций услуг по контракту;
19. Определение и добавление ежемесячных списаний (для планировщика задач);
20. Получение строки номера телефона по полю номера телефона;
21. Удаление просроченных услуг (для планировщика задач).

Детально процедуры описаны в приложении В.

# 2.4 Индексы

Индекс – объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать много времени.

При разработке курсового проекта были созданы следующие индексы:

1. Индекс по идентификатору контракта и по причине для таблицы Debit (нужен для быстрого нахождения и вставки списаний);
2. Индекс по идентификатору контракта для таблицы Service (нужен для быстрого нахождения сервисов для вставки списаний);
3. Индекс для нахождения строк номеров телефонов по полю номера мобильного телефона;
4. Индекс для поиска услуги по дате отключения (для нахождения просроченных услуг).

Детально индексы описаны в приложении Г.

# 2.5 Планировщики задач

СУБД Oracle – большой и сложный механизм, требующий выполнения определенных плановых работ, таких как сбор статистики о хранимых объектах или сбор/чистка внутренней информации. Необходимость осуществлять плановый запуск работ могут испытывать и пользователи БД.

При разработке курсового проекта были созданы следующие планировщики задач:

1. Планировщик определения и добавления списаний, на основании прошедшего цикла в виде месяца;
2. Планировщик удаления просроченных услуг.

Детально планировщики задач описаны в приложении Д.

# 2.6 Триггеры

Триггер – хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением INSERT, удалением DELETE строки в заданной таблице, или изменением UPDATE данных в определённом столбце заданной таблицы реляционной базы данных.

При разработке курсового проекта были созданы следующие триггеры:

1. Добавление в таблицу событий аккаунтов после каждого добавления нового аккаунта;
2. Добавление в таблицу событий аккаунтов после изменения аккаунта;
3. Добавление в таблицу событий аккаунтов после удаления аккаунта.

Детально триггеры описаны в приложении Е.

# 2.7 Представления

Представление – виртуальная таблица, представляющая собой поименованный запрос, который будет подставлен как подзапрос при использовании представления. В отличие от обычных таблиц реляционных баз данных, представление не является самостоятельной частью набора данных, хранящегося в базе.

При разработке курсового проекта были созданы следующие представления:

1. Объединённые поля идентификатора контракта, идентификатора клиента и идентификатора описания сервиса из таблиц Contract и Service (для установки ассоциации в Data Miner);
2. Объединённые поля идентификатора клиента и идентификатора тарифа из таблиц Contract и Tariff\_plan (для установки ассоциации в Data Miner).

Детально представления описаны в приложении Ж.

# 2.8 Реализация технологии

В процессе разработки проекта была использована технология Advanced Analytics, а точнее её компонента Oracle Data Mining.

Advanced Analytics - дополнительная надстройка к Oracle Database, способная превратить базу данных в полномасштабную аналитическую платформу за счет двух основных компонентов: Oracle Data Mining и Oracle R Enterprise. Опция Oracle Advanced Analytics дает возможность создавать аналитические приложения, а также приложения для стратегического планирования в режиме реального времени. Данный функционал наиболее актуален для разработки аналитики в таких сферах, как динамика прибытия или убытия клиентов, выдвижение рекомендации и предотвращение мошенничества.

Компонента Oracle Data Mining была использована для установки ассоциаций между клиентами, их наборами тарифов, и тарифами, которые скорее всего им также будут нужны, а также ассоциаций между договорами, их услугами, и теми услугами, которые скорее всего подойдут под набор. То есть компонента Oracle Data Mining используется для подбора рекомендаций по тарифным планам и услугам.

Для этого, как было описано выше, были созданы представления (рисунок 2.8.1).

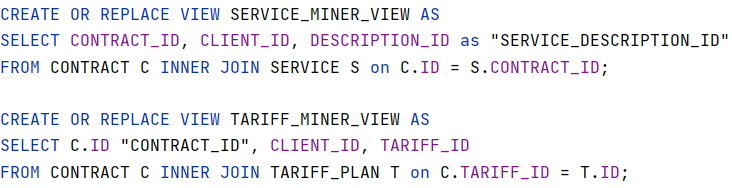


Рисунок 2.8.1 – Представления для установления ассоциаций

После в «Рабочем процессе» (Workflow) добавляем созданные представления, а также компоненты «Ассоциация» (Association), по одной на представление и попарно объединяем (рисунок 2.8.2).

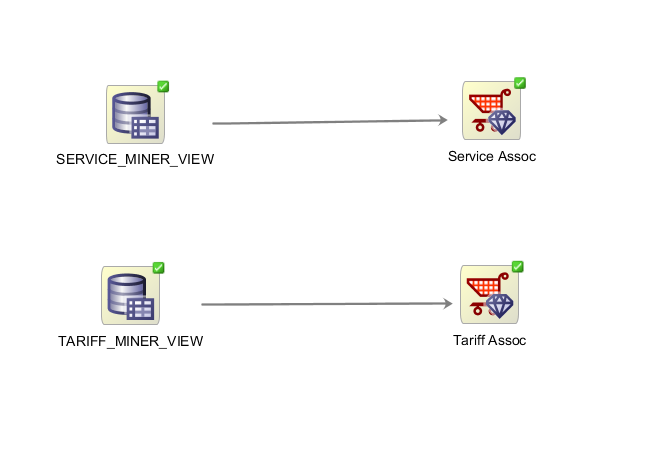


Рисунок 2.8.2 – Связь представлений и компонент ассоциации

Далее в редактировании ассоциации устанавливаем в идентификаторы транзакций (Transaction IDs) элементы представления, с которыми связаны элементы, по которым будут установлены правила ассоциации, затем в идентификаторы предметов (Item ID) сами связанные элементы (рисунок 2.8.3).

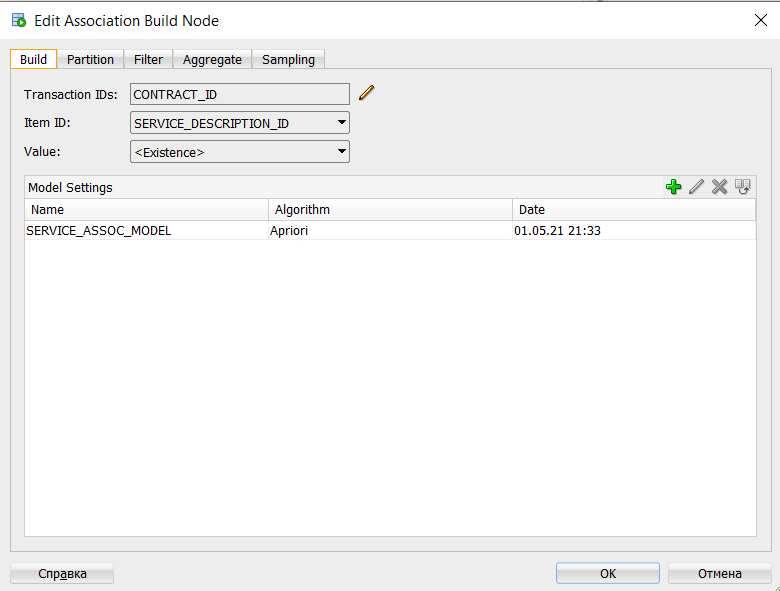


Рисунок 2.8.3 – Окно редактирования ассоциации

После в расширенных настройках (Advanced Settings) устанавливаем параметры максимального размера правила (Maximum rule length), минимальное доверие (Minimum confidence), минимальную поддержку (Minimum support), минимальное размер поддержки (Minimum support count) и минимальную обратное доверие (Minimum reverse confidence) (рисунок 2.8.4).

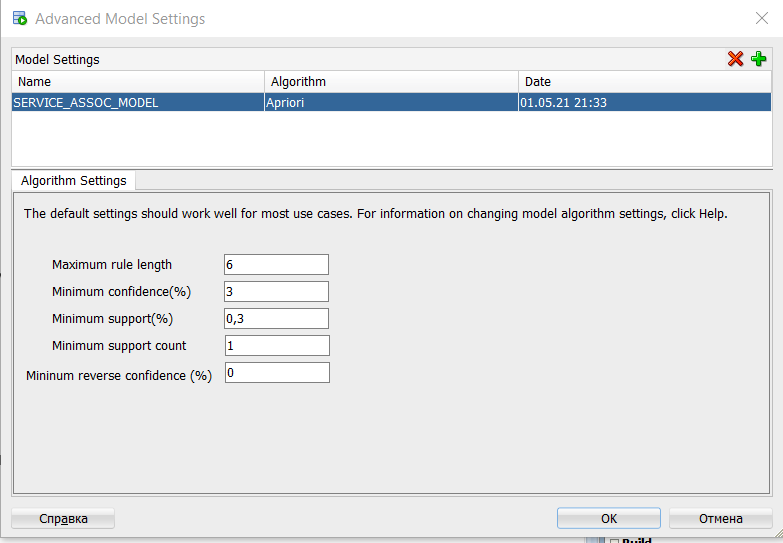


Рисунок 2.8.4 – Окно расширенных настроек ассоциации

Далее запускаем анализ данных (Run) (рисунок 2.8.5).

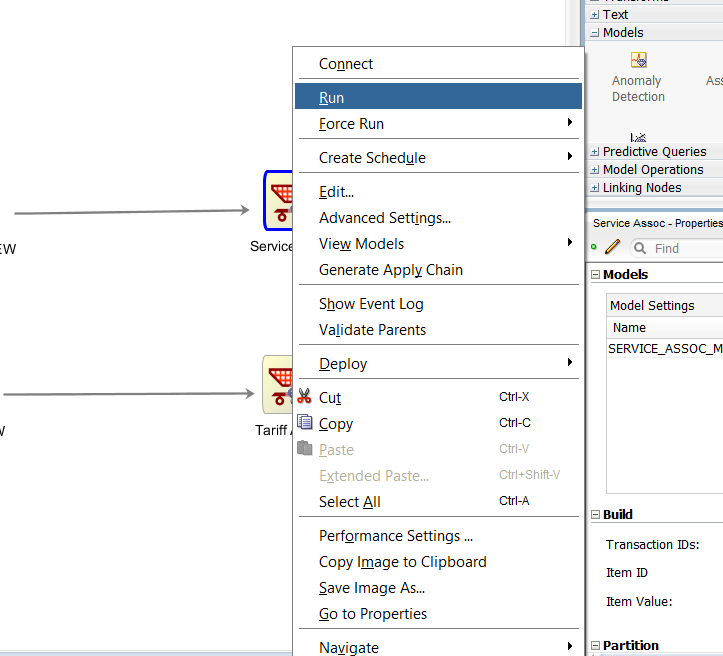


Рисунок 2.8.5 – Контекстное меню с запуском анализа

В результате мы получаем список правил и их характеристик. Для одного набора элементов может быть составлено сразу множество правил, потому лучше отбирать те, у которых показатель Lift выше всего, пример списка правил на рисунке 2.8.6.

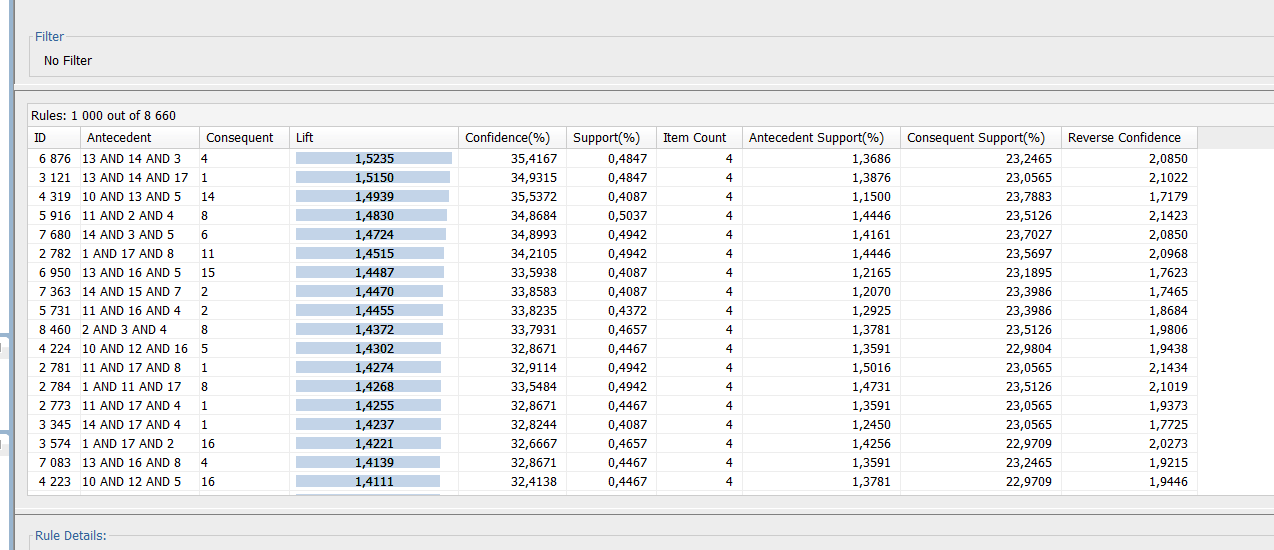


Рисунок 2.8.6 – Список правил ассоциации

В наборе таблиц и представлений появились таблицы и представления со специфическими именами (рисунок 2.8.7).

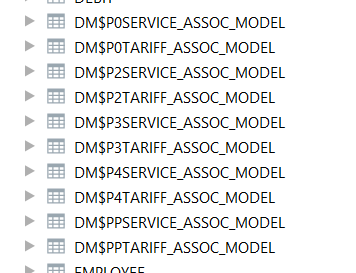


Рисунок 2.8.7 – Сгенерированные таблицы

Именно из них можно извлекать данные для составления рекомендаций.

# 2.9 Экспорт и импорт данных

Для экспорта и импорта данных будет использоваться формат XML. Для этого нужно создать директорию, где будут храниться наши данные, пример на рисунке 2.9.1.



Рисунок 2.9.1 – Код создания директории

В списке объектов директорий появится объект (рисунок 2.9.2).

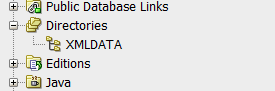


Рисунок 2.9.2 – Объект директории

Используя следующий шаблон, можно экспортировать данные из таблицы в формат XML (рисунок 2.9.3).

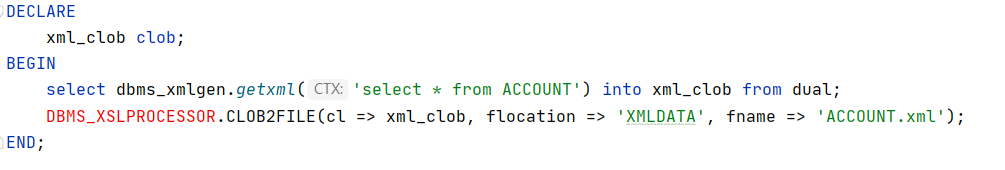


Рисунок 2.9.3 – Шаблон кода для экспортирования данных

Для импортирования данных подойдёт такой шаблон (рисунок 2.9.4). Этот шаблон состоит из двух частей. В первой части происходит сама вставка данных. Во второй части происходит перемещение последовательности, генерирующей новый числовой идентификатор, на шаг самого большого существующего в таблице идентификатора.

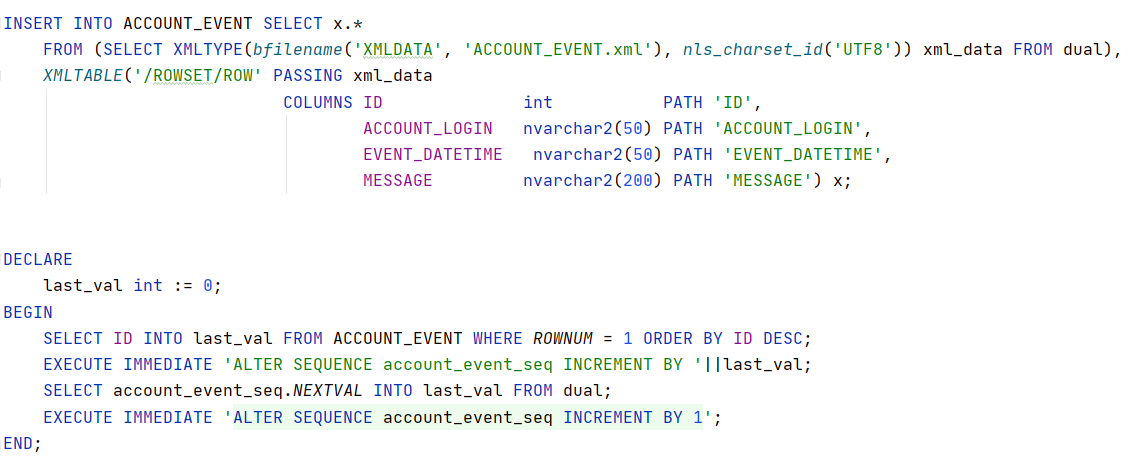


Рисунок 2.9.4 – Шаблон кода для импортирования данных

Отметим, что вставка данных должна проводится в заново созданных таблицах, желательно с обнулёнными последовательностями.

3 Проектирование и разработка программного средства

В процессе выполнения курсового проекта было разработано desktop-приложение, предназначенное для взаимодействия пользователя с базой данных. Приложение было разработано на языке программирования C# с использованием таких технологий как Windows Presentation Foundation (WPF) [3], ADO.NET [4] и ORM Dapper [2]. Структура разработанного программного средства представлена ниже (рисунок 3.1).

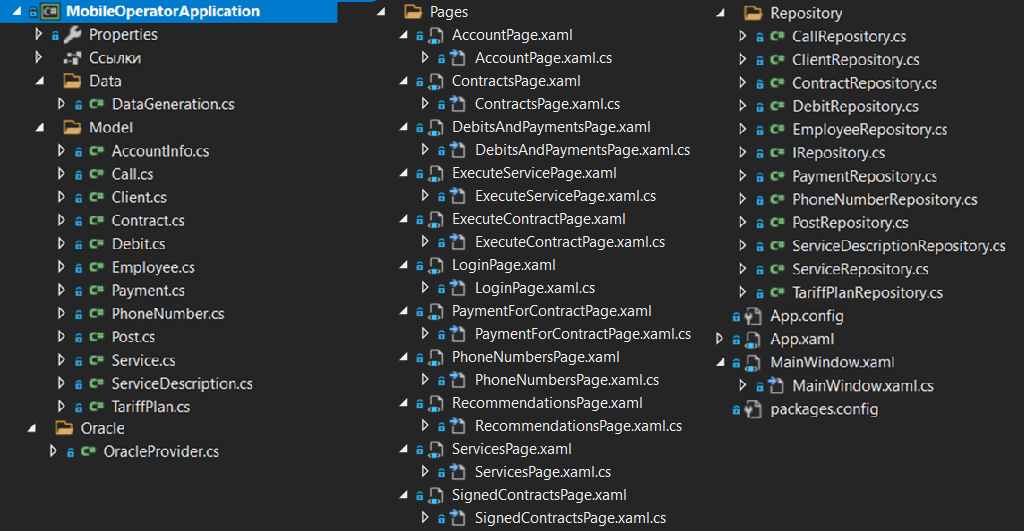


Рисунок 3.1 – Структура проекта

Проект включает в себя следующее окно:

* MainWindow – главное окно c пространством кнопок переключения;

Проект включает в себя следующие страницы:

* AccountPage – страница с информацией об аккаунте клиента;
* ContractsPage – страница со списком договоров клиента;
* DebitsAndPaymentsPage – страница со списком вычетов и начислений на счета договоров.
* ExecuteServicePage – страница подписания услуги для клиента;
* ExecuteContractPage – страница подписания договора для клиента;
* LoginPage – страница входа в аккаунт клиента или сотрудника;
* PaymentForContractPage – страница пополнения баланса контракта;
* PhoneNumbersPage – страница со списком номеров телефона клиента;
* RecommendationsPage – страница рекомендаций для клиента;
* ServicesPage – страница со списком услуг клиента;
* SignedContractsPage – страница подписанных сотрудником договоров.

Для взаимодействия через приложения с данными из БД была применена технология ADO.NET и использована ORM Dapper [2]. ADO.NET предоставляет собой технологию работы с данными, которая основана на платформе .NET Framework. Эта технология представляет нам набор классов, через которые мы можем отправлять запросы к базам данных, устанавливать подключения, получать ответ от базы данных и производить ряд других операций. Dapper представляет собой библиотеку с методами расширения для ADO.NET.

Для установки соединения с нашей базой данных указывается строка подключения в момент создания OracleConnection в OracleProvider как на рисунке 3.2.

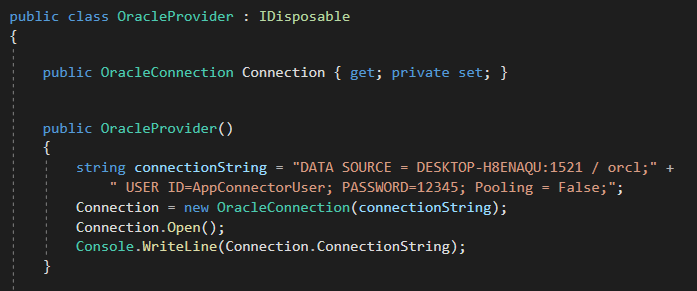


Рисунок 3.2 – Строка подключения

Элементы папки Model предназначены для хранения строк из базы данных. Пример на рисунке 3.3.



Рисунок 3.3 – Пример модели таблицы

Repository содержит в себе классы для управления таблицами, реализующие интерфейс IRepository (рисунок 3.4).

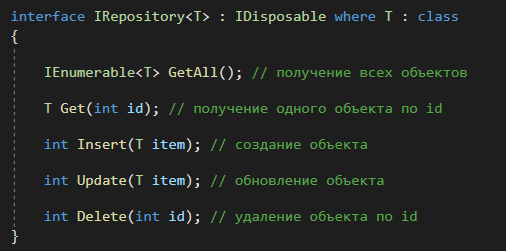


Рисунок 3.4 – Интерфейс IRepository

Пример класса для управления таблицей, реализующего интерфейс IRepository на рисунке 3.5.

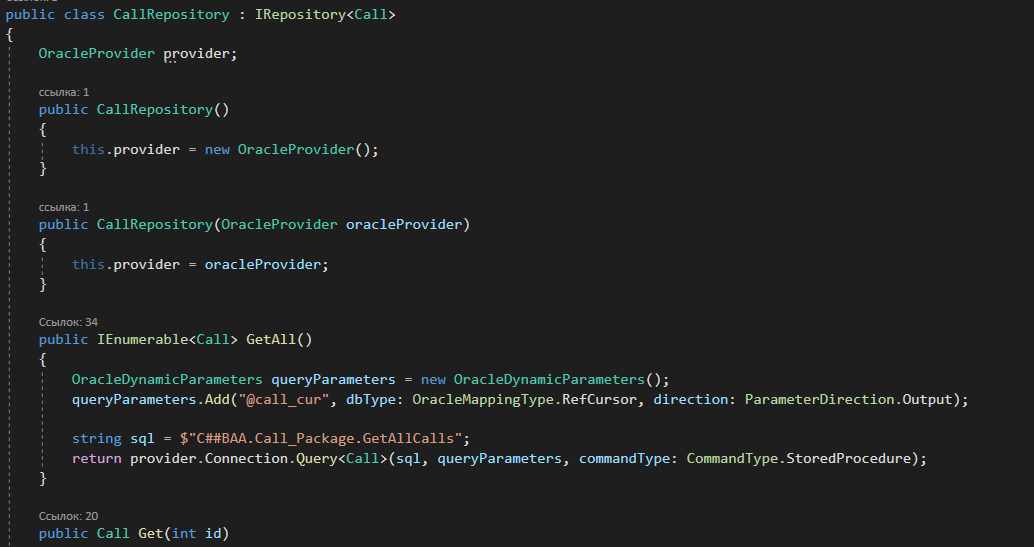


Рисунок 3.5 – Пример менеджера таблицы

Класс DataGeneration предназначен для генерации всей базы данных (Рисунок 3.6).

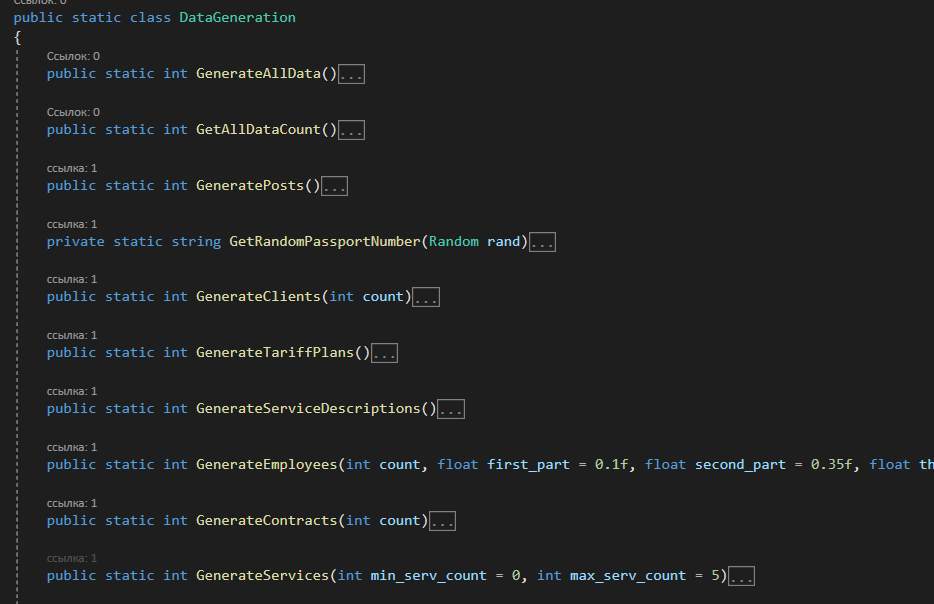


Рисунок 3.6 – Пример менеджера таблицы

Благодаря этому классу и планировщику задач с процедурой по выявлению и добавлению, не совершённых списаний, было сгенерировано более 750 000 строк данных.

4 Тестирование и проверка работоспособности

# 4.1 Тестирование производительности базы данных

Для проверки производительности базы данных необходимо заполнить ее большим количеством различных данных и узнать время выполнения одного запроса. Для этого с помощью процедуры для планировщика задач, который генерирует списания на основании подошедшего срока списания, было создано более 600000 строк (рисунок 4.1.1).

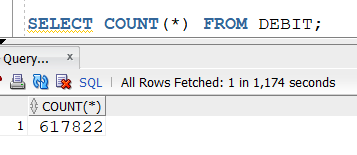


Рисунок 4.1.1 – Количество записей в таблице

Построим план запроса по всем элементам таблицы и посмотрим время его выполнения (рисунок 4.1.2).

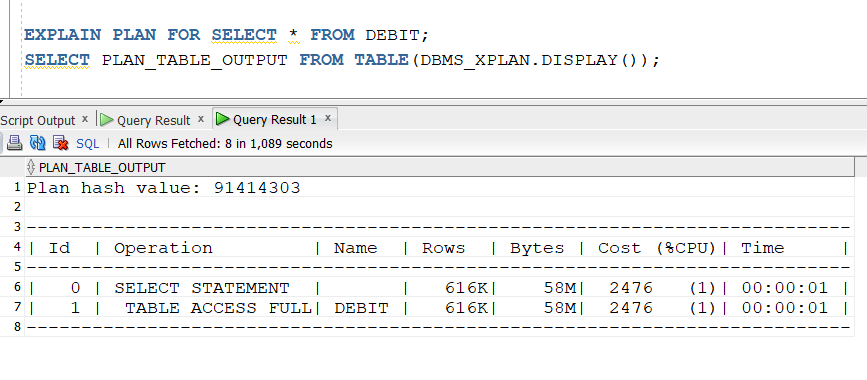


Рисунок 4.1.2 – План запроса

Как можно увидеть из рисунка запрос на более чем 600000 строк выполнился очень быстро, примерно за 1 секунду.

# 4.2 Проверка работоспособности программного средства

Разработанное программное средство полностью соответствует созданному для него техническому заданию.

Для проверки корректности работы программного средства, продемонстрируем авторизацию пользователя (рисунок 4.2.1).

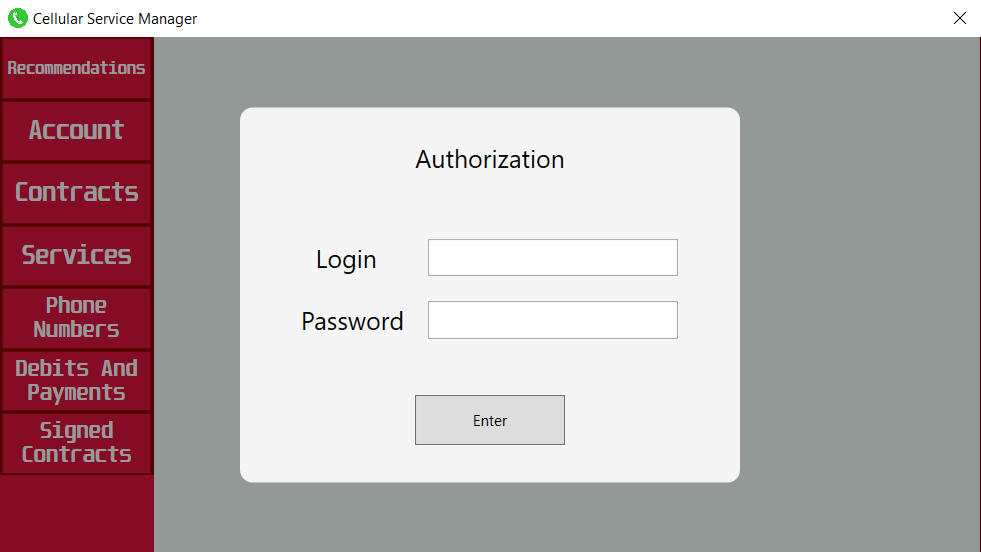


Рисунок 4.2.1 – Окно авторизации

После ввода данных авторизации и нажатия на кнопку «Enter» производится вызов процедуры в базе данных, которая хэширует введённых пароль, и при совпадении логина и хэшированного пароля в базе данных, возвращает уровень доступа у данного аккаунта. В случае его отсутствия, возвращает пустой логин и уровень доступа -999.

Функция вызова из приложения изображён на рисунке 4.2.2.

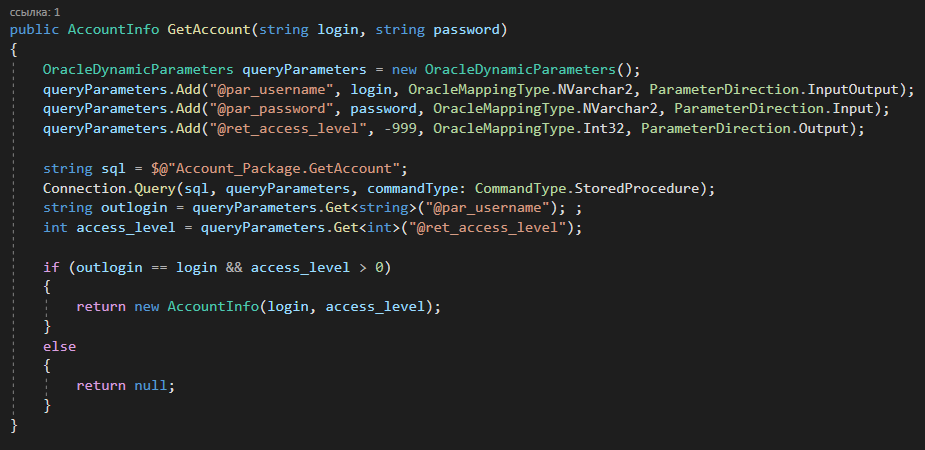


Рисунок 4.2.2 – Функция вызова процедуры из приложения

Процедура авторизации в базе данных изображена на рисунке 4.2.3.



Рисунок 4.2.3 – Процедура авторизации в базе данных

Содержимое таблицы Account изображено на рисунке 4.2.4.

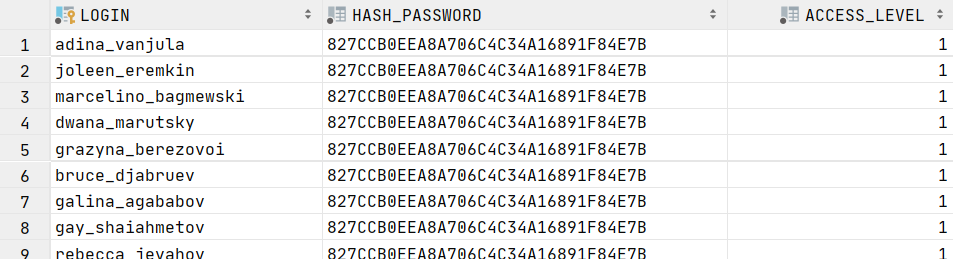


Рисунок 4.2.4 – Содержимое таблицы Account

Если был выполнен вход в аккаунт сотрудника, то в приложении возможно будет только просмотреть список всех контрактов, которые подписал данный сотрудник (рисунок 4.2.5).



Рисунок 4.2.5 – Список подписанных сотрудником контрактов

В случае, если вход был выполнен с аккаунта клиента, то интерфейс приложения будет иметь следующий вид (рисунок 4.2.6).

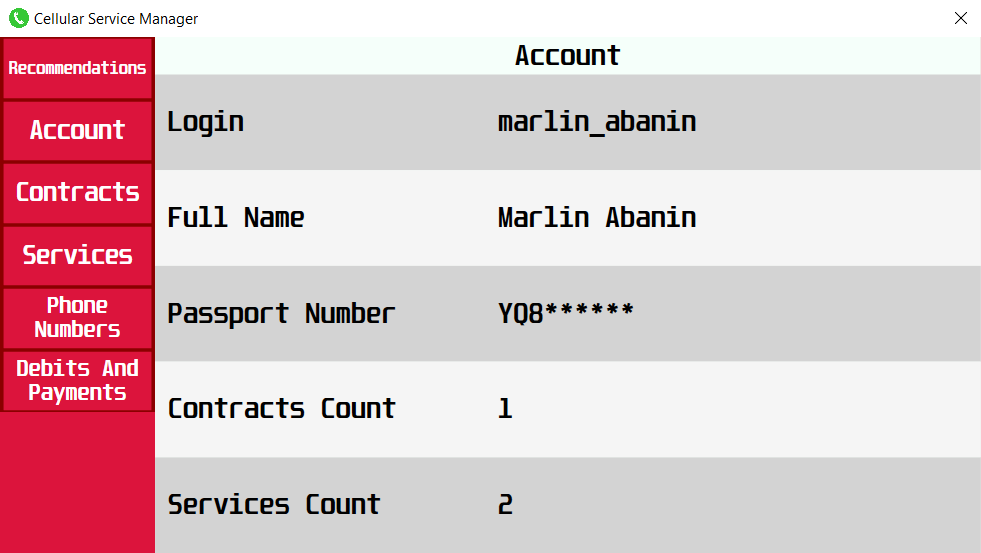


Рисунок 4.2.6 – Интерфейс приложения для клиента

Клиенту доступны страницы «Рекомендации» (Recommendations), «Аккаунт» (Account), «Договоры» (Contracts), «Услуги» (Services), «Номера телефонов» (Phone Numbers) и «Списания и оплаты» (Debits And Payments). Первым же делом, при входе в аккаунт клиента, открывается страница «Аккаунт», где пользователь может просмотреть некоторую информацию об аккаунте и его статистику (рисунок 4.6). На странице рекомендаций можно просмотреть списки рекомендаций по тарифам и услугам, для каждого контракта в отдельности.

Рекомендации для аккаунта marlin\_abanin изображены на рисунке 4.2.7.

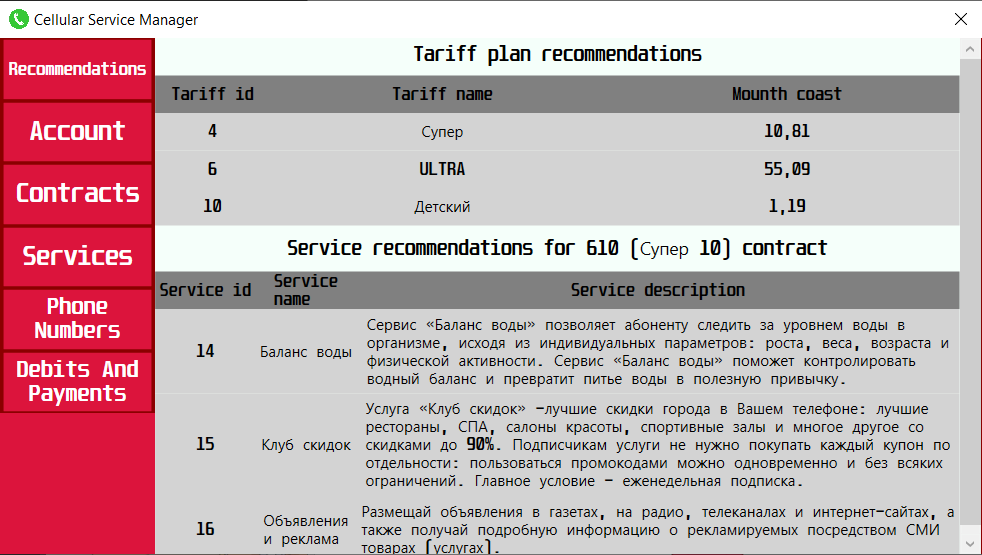


Рисунок 4.2.7 – Рекомендации для аккаунта marlin\_abanin

Рекомендации для аккаунта hector\_jagovenko изображены на рисунке 4.2.8.

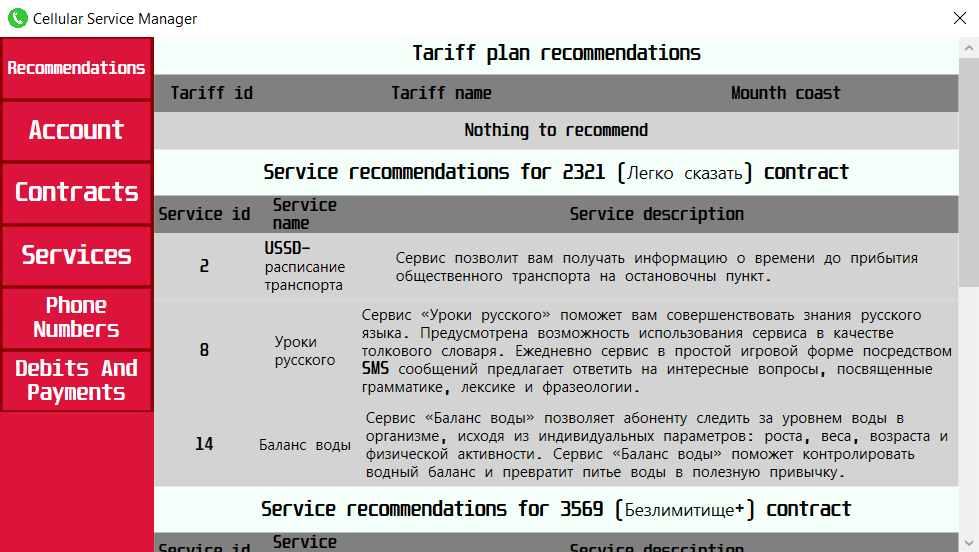


Рисунок 4.2.8 – Рекомендации для аккаунта hector\_jagovenko

На странице договоров можно посмотреть список договоров (рисунок 4.2.9).

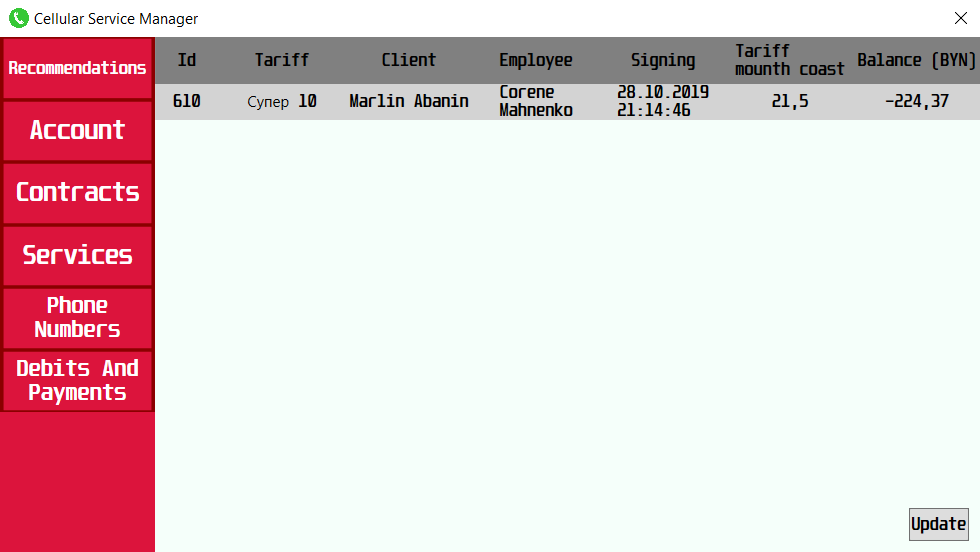


Рисунок 4.2.9 – Страница договоров

Правой кнопкой мыши по договору, через контекстное меню можно выбрать «Пополнение» (Payment) и «Расторжение» (Terminate) (рисунок 4.2.10).

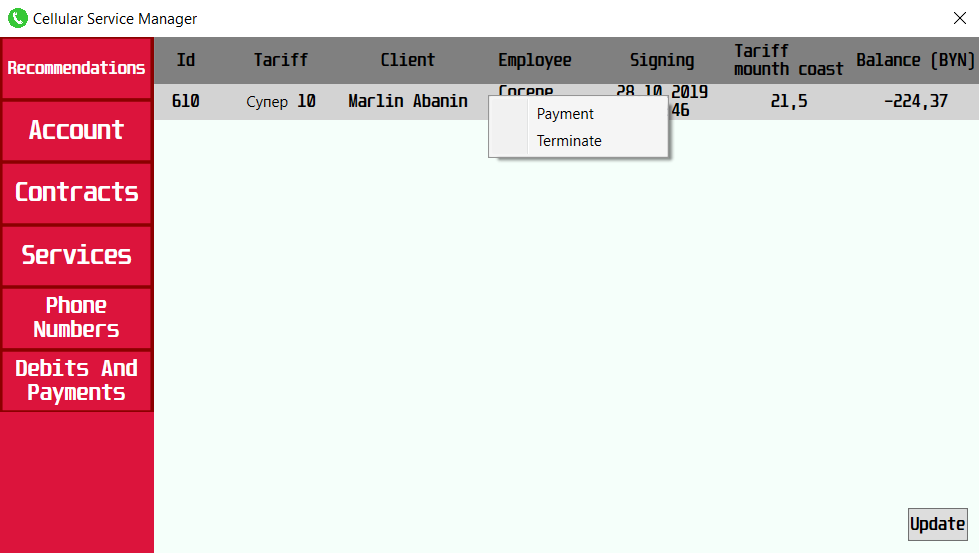


Рисунок 4.2.10 – Контекстное меню

Первая опция откроет соответствующую форму (рисунок 4.2.11).

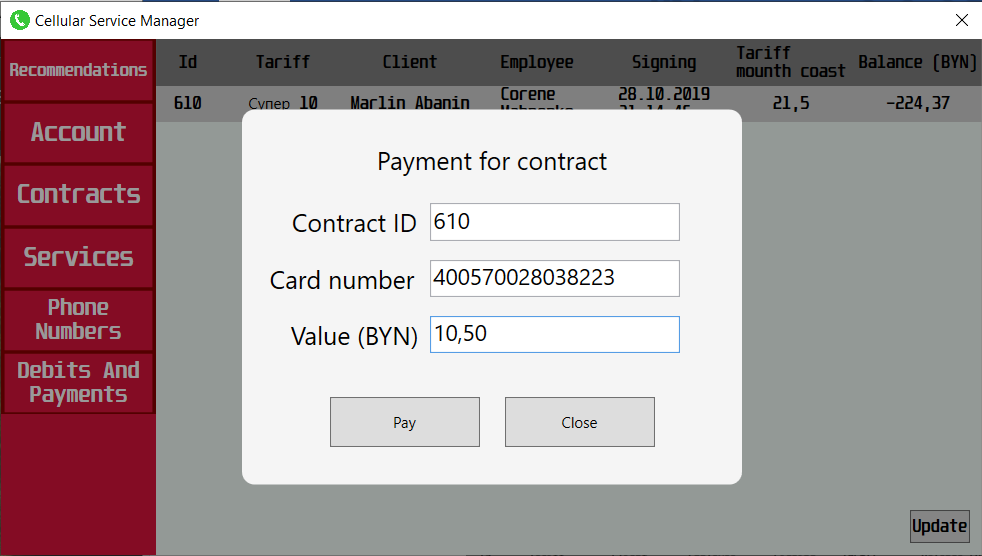


Рисунок 4.2.11 – Форма для пополнения баланса

Правой кнопкой мыши по контейнеру договоров через контекстное меню, выбрав «Подписать контракт» (Execute contract) откроется форма подписания нового контракта, с возможностью выбора тарифа и предпочитаемого номера телефона (рисунок 4.2.12).

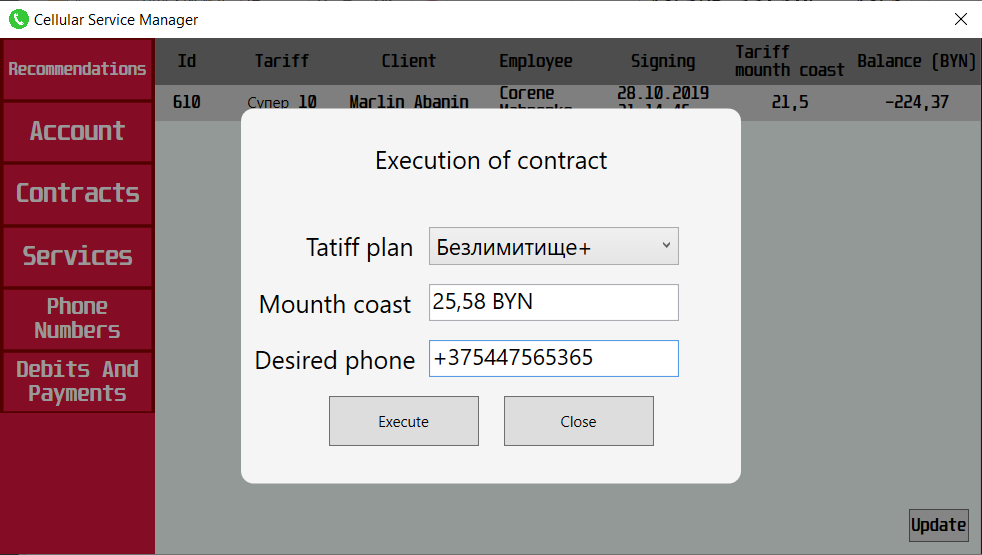


Рисунок 4.2.12 – Форма подписания контракта

На странице услуг можно посмотреть список услуг (рисунок 4.2.13).

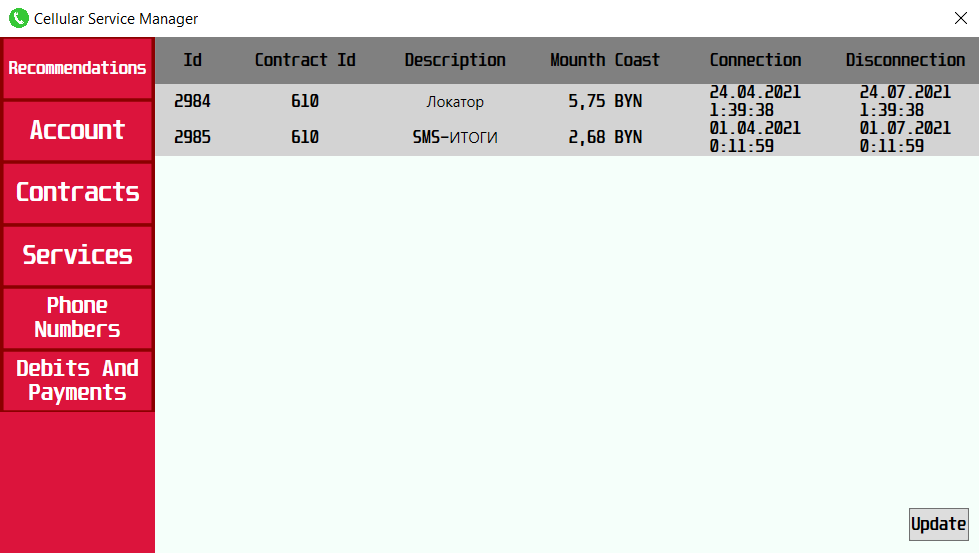


Рисунок 4.2.13 – Страница услуг

Правой кнопкой мыши по контейнеру договоров через контекстное меню, выбрав «Подписать услугу» (Execute service) откроется форма подписания нового контракта, с возможностью выбора тарифа и предпочитаемого номера телефона (рисунок 4.2.14).

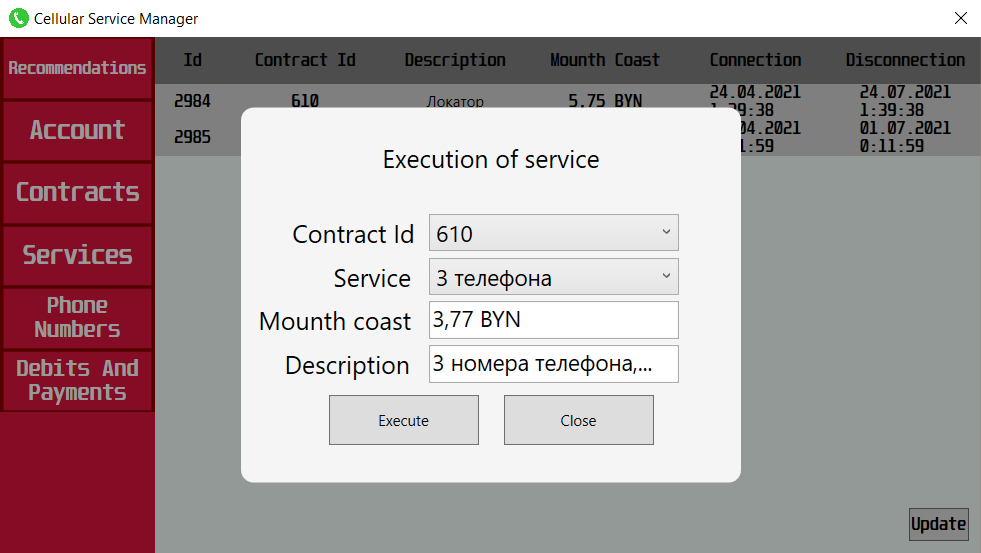


Рисунок 4.2.14 – Форма подписания услуги

На странице номеров телефона можно посмотреть все номера телефонов, закреплённых за договорами (рисунок 4.2.15).

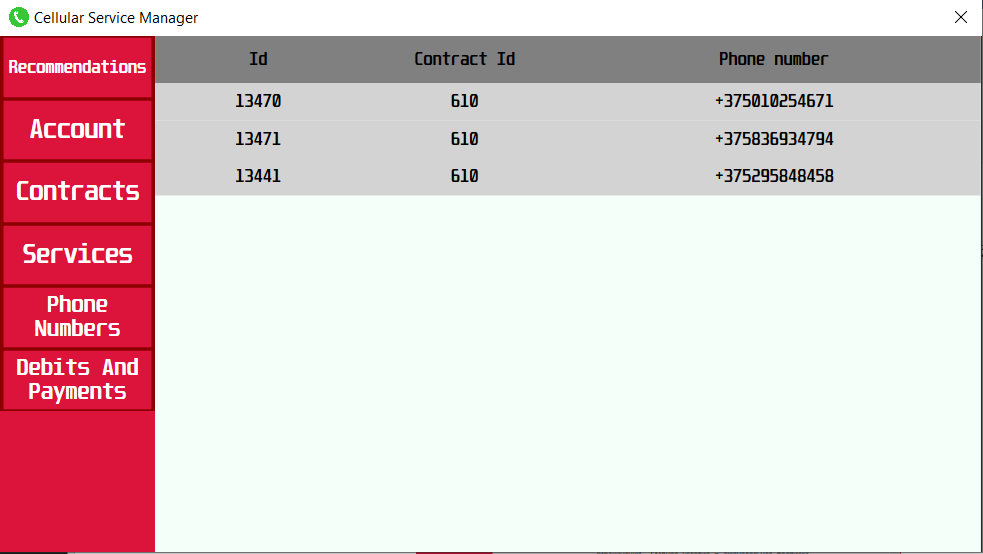


Рисунок 4.2.15 – Страница номеров телефона

На странице вычетов и пополнений можно посмотреть все изменения балансов контрактов (рисунок 4.2.16).

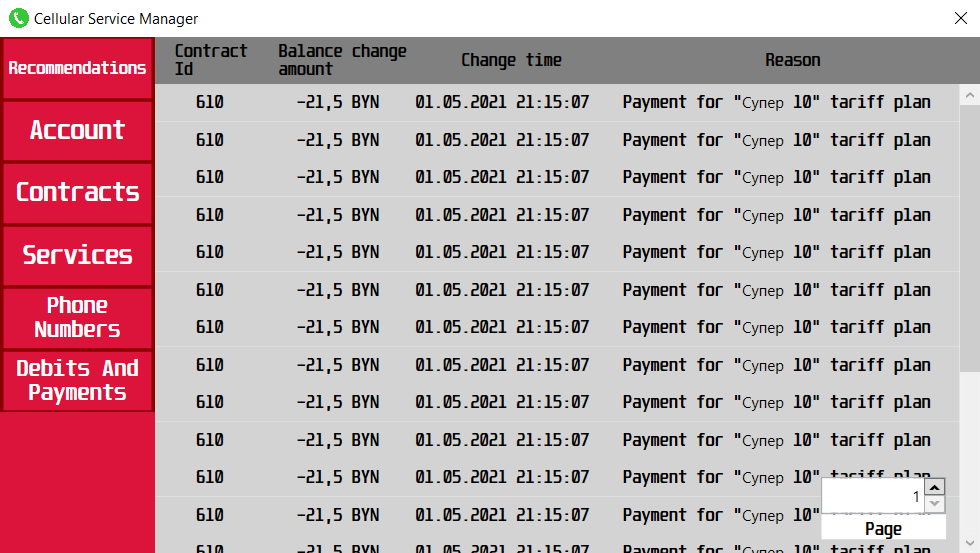


Рисунок 4.2.16 – Страница изменения балансов контрактов

Переключение между частями списка по 20 элементов происходит через элемент управления в правом нижнем углу, так как записей изменения баланса может быть крайне много, на столько, что интерфейс не сможет такое количество элементов потянуть.

Заключение

В процессе решения поставленной задачи была достигнута поставленная цель по созданию базы данных оператора сотовой связи, которая в купе с приложением формирует полноценное desktop-приложение. Основной целью курсового проекта стало проектирование базы данных для дальнейшей интеграции с приложением, которое помогло облегчить взаимодействие с базой данных посредством программного интерфейса. При разработке выполнены следующие пункты:

* авторизация пользователей;
* разделение пользователя на сотрудника и клиента;
* просмотр, удаление и добавление договоров клиентом;
* просмотр, удаление и добавление услуг клиентом;
* просмотр номеров телефона, привязанных к договорам клиентом;
* просмотр клиентов и договоров, которые с ними были подписаны, авторизованным сотрудником;
* просмотр и добавление рекомендованных тарифов и услуг клиентом;
* просмотр информации об аккаунте клиентом.

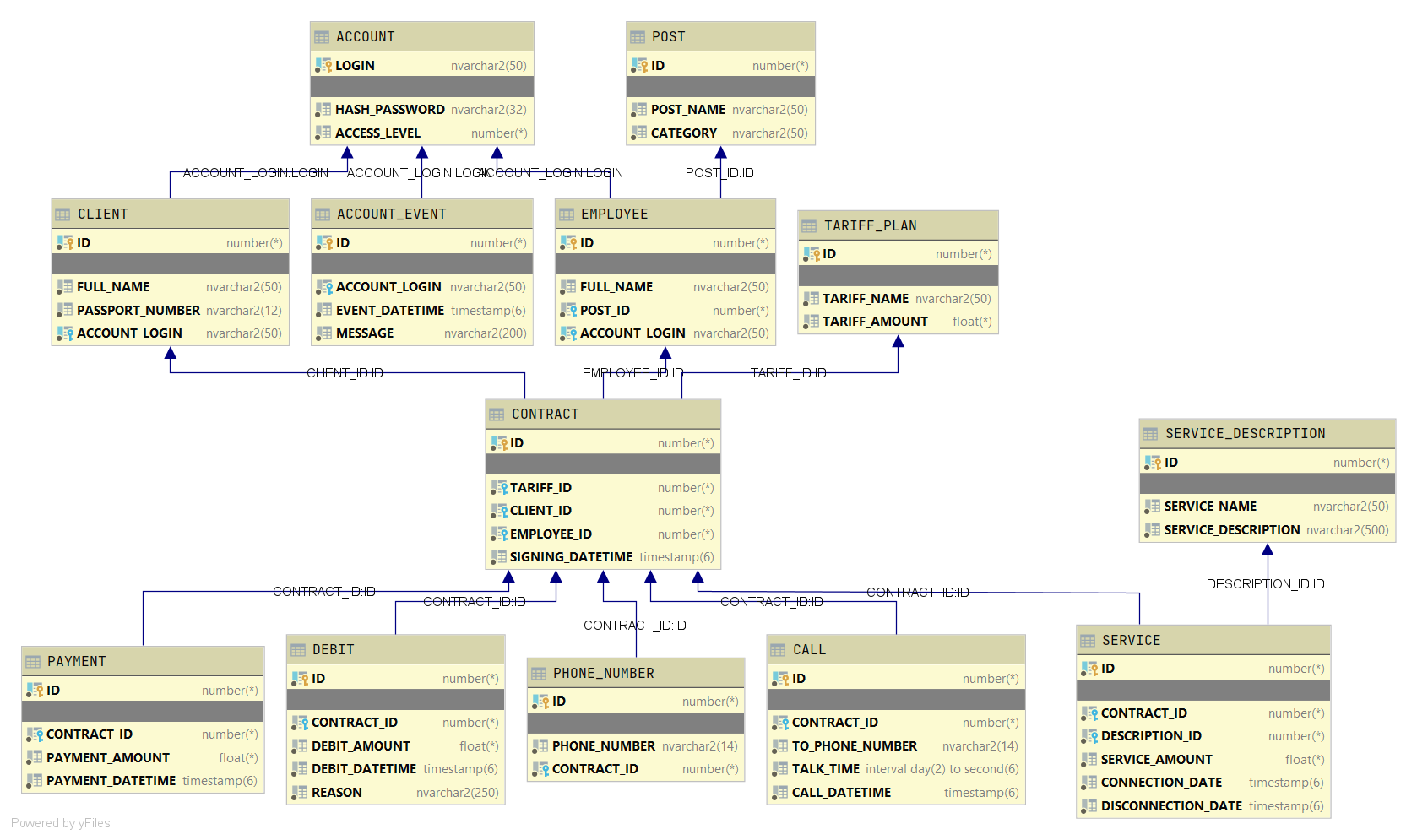
Для последнего пункта была использована технология Advanced Analytics, точнее её компонента Oracle Data Mining, где с помощью компонента «Ассоциация» был составлен список всех возможных правил с разным уровнем ценности.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает правильно, а требования технического задания выполнены.

Список литературы

1. Сайт Oracle – Электронный ресурс. – Ссылка: [Oracle Database Documentation - Oracle Database](https://docs.oracle.com/en/database/oracle/oracle-database/)
2. Сайт dapper-tutorial.net – Электронный ресурс. – Ссылка: [Dapper Dapper Tutorial | Dapper Tutorial and Documentation](https://dapper-tutorial.net/)
3. Сайт Metanit WPF – Электронный ресурс. – Ссылка: [WPF и C# | Полное руководство (metanit.com)](https://metanit.com/sharp/wpf/)
4. Сайт Metanit ADO.NET – Электронный ресурс. – Ссылка: [Введение в ADO.NET (metanit.com)](https://metanit.com/sharp/adonet/1.1.php)

Приложение А



Приложение Б

CREATE TABLE Account (  
login nvarchar2(50) PRIMARY KEY,  
hash\_password nvarchar2(32) NOT NULL,  
access\_level int NOT NULL  
);  
  
CREATE SEQUENCE account\_event\_seq INCREMENT BY 1;  
  
CREATE TABLE Account\_event (  
id int DEFAULT account\_event\_seq.*NEXTVAL* PRIMARY KEY,  
account\_login nvarchar2(50) NOT NULL,  
event\_datetime timestamp NOT NULL,  
message nvarchar2(200) NOT NULL,  
FOREIGN KEY (account\_login) REFERENCES Account(login)  
);  
  
CREATE SEQUENCE post\_seq INCREMENT BY 1;  
  
CREATE TABLE Post (  
id int DEFAULT post\_seq.*NEXTVAL* PRIMARY KEY,  
post\_name nvarchar2(50) NOT NULL,  
category nvarchar2(50) NOT NULL  
);  
  
CREATE SEQUENCE client\_seq INCREMENT BY 1;  
  
CREATE TABLE Client(  
id int DEFAULT client\_seq.*NEXTVAL* PRIMARY KEY,  
full\_name nvarchar2(50) NOT NULL,  
passport\_number nvarchar2(12) NOT NULL,  
account\_login nvarchar2(50) NOT NULL UNIQUE,  
FOREIGN KEY (account\_login) REFERENCES Account(login)  
);

CREATE SEQUENCE service\_description\_seq INCREMENT BY 1;  
  
CREATE TABLE Service\_description(  
id int DEFAULT service\_description\_seq.*NEXTVAL* PRIMARY KEY,  
service\_name nvarchar2(50) NOT NULL,  
service\_description nvarchar2(500) NOT NULL  
);  
  
CREATE SEQUENCE tariff\_plan\_seq INCREMENT BY 1;  
  
CREATE TABLE Tariff\_plan(  
id int DEFAULT tariff\_plan\_seq.*NEXTVAL* PRIMARY KEY,  
tariff\_name nvarchar2(50) NOT NULL,  
tariff\_amount float(126) NOT NULL  
);CREATE SEQUENCE employee\_seq INCREMENT BY 1;  
  
CREATE TABLE Employee(  
id int DEFAULT employee\_seq.*NEXTVAL* PRIMARY KEY,  
full\_name nvarchar2(50) NOT NULL,  
post\_id int NOT NULL,  
account\_login nvarchar2(50) NOT NULL UNIQUE,  
FOREIGN KEY (post\_id) REFERENCES Post(id),  
FOREIGN KEY (account\_login) REFERENCES Account(login)  
);CREATE SEQUENCE contract\_seq INCREMENT BY 1;  
  
CREATE TABLE Contract (  
id int DEFAULT contract\_seq.*NEXTVAL* PRIMARY KEY,  
tariff\_id int NOT NULL,  
client\_id int NOT NULL,  
employee\_id int NOT NULL,  
signing\_datetime timestamp NOT NULL,  
FOREIGN KEY (tariff\_id) REFERENCES Tariff\_plan(id),  
FOREIGN KEY (client\_id) REFERENCES Client(id),  
FOREIGN KEY (employee\_id) REFERENCES Employee(id)  
);  
  
CREATE SEQUENCE phone\_number\_seq INCREMENT BY 1;

CREATE TABLE Phone\_number (  
id int DEFAULT phone\_number\_seq.*NEXTVAL* PRIMARY KEY,  
phone\_number nvarchar2(14) NOT NULL,  
contract\_id int NOT NULL,  
CONSTRAINT phone\_regex\_pn CHECK (*REGEXP\_LIKE* (phone\_number, '^((\+\d{3})|(\d{2}))\d{2}\d{3}\d{4}$')),  
FOREIGN KEY (contract\_id) REFERENCES Contract(id) ON DELETE CASCADE  
);  
  
CREATE SEQUENCE call\_seq INCREMENT BY 1;  
  
CREATE TABLE Call (  
id int DEFAULT call\_seq.*NEXTVAL* PRIMARY KEY,  
contract\_id int NOT NULL,  
to\_phone\_number nvarchar2(14) NOT NULL,  
talk\_time interval day to second NOT NULL,  
call\_datetime timestamp NOT NULL,  
CONSTRAINT phone\_regex\_c CHECK (*REGEXP\_LIKE* (to\_phone\_number, '^((\+\d{3})|(\d{2}))\d{2}\d{3}\d{4}$')),  
FOREIGN KEY (contract\_id) REFERENCES Contract(id) ON DELETE CASCADE  
);  
  
CREATE SEQUENCE payment\_seq INCREMENT BY 1;  
  
CREATE TABLE Payment(  
id int DEFAULT payment\_seq.*NEXTVAL* PRIMARY KEY,  
contract\_id int NOT NULL,  
payment\_amount float(126) NOT NULL,  
payment\_datetime timestamp NOT NULL,  
FOREIGN KEY (contract\_id) REFERENCES Contract(id) ON DELETE CASCADE  
);  
  
CREATE SEQUENCE debit\_seq INCREMENT BY 1;  
  
CREATE TABLE Debit(  
id int DEFAULT debit\_seq.*NEXTVAL* PRIMARY KEY,  
contract\_id int NOT NULL,  
debit\_amount float(126) NOT NULL,  
debit\_datetime timestamp NOT NULL,  
reason nvarchar2(250) NOT NULL,  
FOREIGN KEY (contract\_id) REFERENCES Contract(id) ON DELETE CASCADE  
);CREATE SEQUENCE service\_seq INCREMENT BY 1;  
  
CREATE TABLE Service (  
id int DEFAULT service\_seq.*NEXTVAL* PRIMARY KEY,  
contract\_id int NOT NULL,  
description\_id int NOT NULL,  
service\_amount float(126) NOT NULL,  
connection\_date timestamp NOT NULL,  
disconnection\_date timestamp NOT NULL,  
FOREIGN KEY (contract\_id) REFERENCES Contract(id) ON DELETE CASCADE,  
FOREIGN KEY (description\_id) REFERENCES Service\_description(id)  
);

Приложение В

CREATE OR REPLACE PACKAGE Account\_Package IS  
 *-- Создание аккаунта* PROCEDURE *CreateAccount*(par\_username in nvarchar2, par\_password in nvarchar2, par\_access\_level in int, created out int);  
 *-- Получение уровня доступа аккаунта, если введённые данные верны* PROCEDURE *GetAccount*(par\_username in out nvarchar2, par\_password in nvarchar2, ret\_access\_level out int);  
 *-- Поменять пароль аккаунта* PROCEDURE *ChangeAccountPassword*(par\_username in out nvarchar2, par\_old\_password in nvarchar2, par\_new\_password in nvarchar2, changed out int);  
 *-- Вставка события аккаунта* PROCEDURE *InsertAccountEvent*(par\_username in nvarchar2, par\_event\_datetime in nvarchar2, par\_message in nvarchar2, inserted out int);  
END Account\_Package;

CREATE OR REPLACE PACKAGE Call\_Package IS  
 *-- Вставка звонка* PROCEDURE *InsertCall*(par\_contract\_id in int, par\_to\_phone\_number in nvarchar2, par\_talk\_time in nvarchar2, par\_call\_datetime in nvarchar2, inserted out int);  
 *-- Обновление звонка* PROCEDURE *UpdateCall*(par\_id in int, par\_contract\_id in int, par\_to\_phone\_number in nvarchar2, par\_talk\_time in nvarchar2, par\_call\_datetime in nvarchar2, updated out int);  
 *-- Удаление звонка* PROCEDURE *DeleteCall*(par\_id in int, deleted out int);  
 *-- Получение звонка по его идентификатору* PROCEDURE *GetCallById*(par\_id in int, call\_cur out sys\_refcursor);  
 *-- Получение всех звонков в базе данных* PROCEDURE *GetAllCalls*(call\_cur out sys\_refcursor);  
END Call\_Package;

CREATE OR REPLACE PACKAGE Client\_Package IS  
 *-- Вставка клиента* PROCEDURE *InsertClient*(par\_full\_name in nvarchar2, par\_passport\_number in nvarchar2, par\_account\_login in nvarchar2, inserted out int);  
 *-- Обновление клиента* PROCEDURE *UpdateClient*(par\_id in int, par\_full\_name in nvarchar2, par\_passport\_number in nvarchar2, par\_account\_login in nvarchar2, updated out int);  
 *-- Удаление клиента* PROCEDURE *DeleteClient*(par\_id in int, deleted out int);  
 *-- Получение клиента по его идентификатору* PROCEDURE *GetClientById*(par\_id in int, client\_cur out sys\_refcursor);  
 *-- Получение клиента по логину его аккаунта* PROCEDURE *GetClientByLogin*(par\_login in nvarchar2, client\_cur out sys\_refcursor);  
 *-- Получене всех пользователей базы данных* PROCEDURE *GetAllClients*(client\_cur out sys\_refcursor);  
 *-- Получение всех услуг клиента по его идентификатору* PROCEDURE *GetAllServicesByClientId*(par\_id in int, service\_cur out sys\_refcursor);  
 *-- Получение всех договоров по его идентификатору* PROCEDURE *GetAllContractsByClientId*(par\_id in int, contract\_cur out sys\_refcursor);  
 *-- Получение рекомендаций тарифов по идентификатору клиента* PROCEDURE *GetTariffRecommendationsByClientId*(par\_id in int, par\_recommendations\_count in int, tariff\_plan\_cur out sys\_refcursor);  
END Client\_Package;

CREATE OR REPLACE PACKAGE Contract\_Package IS  
 *-- Вставка договора* PROCEDURE *InsertContract*(par\_tariff\_id in int, par\_client\_id in int, par\_employee\_id in int, par\_signing\_datetime in nvarchar2, inserted out int);  
 *-- Обновление договора* PROCEDURE *UpdateContract*(par\_id in int, par\_tariff\_id in int, par\_client\_id in int, par\_employee\_id in int, par\_signing\_datetime in nvarchar2, updated out int);  
 *-- Удаление договора* PROCEDURE *DeleteContract*(par\_id in int, deleted out int);  
 *-- Получение договора по его идентификатору* PROCEDURE *GetContractById*(par\_id in int, contract\_cur out sys\_refcursor);  
 *-- Получение всех договоров в базе данных* PROCEDURE *GetAllContracts*(contract\_cur out sys\_refcursor);  
 *-- Получение баланса договора* PROCEDURE *GetContractBalance*(par\_id in int, balance out float);  
 *-- Получение всех сервисов договора по его идентификатору* PROCEDURE *GetAllServicesByContractId*(par\_id in int, service\_cur out sys\_refcursor);  
 *-- Получение всех номеров телефона договора по его идентификатору* PROCEDURE *GetAllPhoneNumbersByContractId*(par\_id in int, phone\_number\_cur out sys\_refcursor);  
 *-- Получение всех оплат договора по его идентификатору* PROCEDURE *GetAllPaymentsByContractId*(par\_id in int, payment\_cur out sys\_refcursor);  
 *-- Получение всех списаний договора по его идентификатору* PROCEDURE *GetAllDebitsByContractId*(par\_id in int, debit\_cur out sys\_refcursor);  
 *-- Получение рекомендаций сервисов по идентификатору договора* PROCEDURE *GetServiceRecommendationsByContract*(par\_id in int, par\_recommendations\_count in int, service\_descriptions\_cur out sys\_refcursor);  
END Contract\_Package;

CREATE OR REPLACE PACKAGE Debit\_Package IS  
 *-- Вставка списания* PROCEDURE *InsertDebit*(par\_contract\_id in int, par\_debit\_amount in float, par\_debit\_datetime in nvarchar2, par\_reason in nvarchar2, inserted out int);  
 *-- Обноваление списания* PROCEDURE *UpdateDebit*(par\_id in int, par\_contract\_id in int, par\_debit\_amount in float, par\_debit\_datetime in nvarchar2, par\_reason in nvarchar2, updated out int);  
 *-- Удаление списания* PROCEDURE *DeleteDebit*(par\_id in int, deleted out int);  
 *-- Получение списания по его идентификатору* PROCEDURE *GetDebitById*(par\_id in int, debit\_cur out sys\_refcursor);  
 *-- Получение всех списаний базы данных* PROCEDURE *GetAllDebits*(debit\_cur out sys\_refcursor);  
 *-- Определение и вставка списаний, которые должны произойти по причине наступления срока оплаты* PROCEDURE *DetectAndInsertTermDebits*;  
END Debit\_Package;

CREATE OR REPLACE PACKAGE Employee\_Package IS  
 *-- Вставка сотрудника* PROCEDURE *InsertEmployee*(par\_full\_name in nvarchar2, par\_post\_id in int, par\_account\_login in nvarchar2, inserted out int);  
 *-- Обновление сотрудника* PROCEDURE *UpdateEmployee*(par\_id in int, par\_full\_name in nvarchar2, par\_post\_id in int, par\_account\_login in nvarchar2, updated out int);  
 *-- Удаление сотрудника* PROCEDURE *DeleteEmployee*(par\_id in int, deleted out int);  
 *-- Получение сотрудника по его идентификатору* PROCEDURE *GetEmployeeById*(par\_id in int, employee\_cur out sys\_refcursor);  
 *-- Получение сотрудника по логину его аккаунта* PROCEDURE *GetEmployeeByLogin*(par\_login in nvarchar2, employee\_cur out sys\_refcursor);  
 *-- Получение всех сотрудников базы данных* PROCEDURE *GetAllEmployees*(employee\_cur out sys\_refcursor);  
 *-- Получение договоров, подписанных сотрудником, используя его идентификатор* PROCEDURE *GetAllContractsByEmployeeId*(par\_id in int, contract\_cur out sys\_refcursor);  
END Employee\_Package;

CREATE OR REPLACE PACKAGE Payment\_Package IS  
 *-- Вставка оплаты* PROCEDURE *InsertPayment*(par\_contract\_id in int, par\_payment\_amount in float, par\_payment\_datetime in nvarchar2, inserted out int);  
 *-- Обновление оплаты* PROCEDURE *UpdatePayment*(par\_id in int, par\_contract\_id in int, par\_payment\_amount in float, par\_payment\_datetime in nvarchar2, updated out int);  
 *-- Удаление оплаты* PROCEDURE *DeletePayment*(par\_id in int, deleted out int);  
 *-- Получение оплаты по его идентификатору* PROCEDURE *GetPaymentById*(par\_id in int, payment\_cur out sys\_refcursor);  
 *-- Получение всех оплат базы данных* PROCEDURE *GetAllPayments*(payment\_cur out sys\_refcursor);  
END Payment\_Package;

CREATE OR REPLACE PACKAGE PhoneNumber\_Package IS  
 *-- Вставка номера телефона* PROCEDURE *InsertPhoneNumber*(par\_phone\_number in nvarchar2, par\_contract\_id in int, inserted out int);  
 *-- Обновление номера телефона* PROCEDURE *UpdatePhoneNumber*(par\_id in int, par\_phone\_number in nvarchar2, par\_contract\_id in int, updated out int);  
 *-- Удаление номера телефона* PROCEDURE *DeletePhoneNumber*(par\_id in int, deleted out int);  
 *-- Получение номера телефона по его идентификатору* PROCEDURE *GetPhoneNumberById*(par\_id in int, phone\_number\_cur out sys\_refcursor);  
 *-- Получение номера телефона по полю номера телефона* PROCEDURE *GetPhoneNumberByNumber*(par\_number in nvarchar2, phone\_number\_cur out sys\_refcursor);  
 *-- Получение всех номеров телефона базы данных* PROCEDURE *GetAllPhoneNumbers*(phone\_number\_cur out sys\_refcursor);  
END PhoneNumber\_Package;

CREATE OR REPLACE PACKAGE Post\_Package IS  
 *-- Вставка должности* PROCEDURE *InsertPost*(par\_post\_name in nvarchar2, par\_category in nvarchar2, inserted out int);  
 *-- Обновление должности* PROCEDURE *UpdatePost*(par\_id in int, par\_post\_name in nvarchar2, par\_category in nvarchar2, updated out int);  
 *-- Удаление должности* PROCEDURE *DeletePost*(par\_id in int, deleted out int);  
 *-- Получение должности по его идентификатору* PROCEDURE *GetPostById*(par\_id in int, post\_cur out sys\_refcursor);  
 *-- Получение всех должностей базы данных* PROCEDURE *GetAllPosts*(post\_cur out sys\_refcursor);  
END Post\_Package;

CREATE OR REPLACE PACKAGE ServiceDescription\_Package IS  
 *-- Встака описания услуги* PROCEDURE *InsertServiceDescription*(par\_service\_name in nvarchar2, par\_service\_description in nvarchar2, inserted out int);  
 *-- Обновление описания услуги* PROCEDURE *UpdateServiceDescription*(par\_id in int, par\_service\_name in nvarchar2, par\_service\_description in nvarchar2, updated out int);  
 *-- Удаление описания услуги* PROCEDURE *DeleteServiceDescription*(par\_id in int, deleted out int);  
 *-- Получение описания услуги по его идентификатору* PROCEDURE *GetServiceDescriptionById*(par\_id in int, service\_desc\_cur out sys\_refcursor);  
 *-- Получение всех описаний услуг базы данных* PROCEDURE *GetAllServiceDescriptions*(service\_desc\_cur out sys\_refcursor);  
END ServiceDescription\_Package;

CREATE OR REPLACE PACKAGE Service\_Package IS  
 *-- Вставка услуги* PROCEDURE *InsertService*(par\_contract\_id in int, par\_description\_id in int, par\_service\_amount in float, par\_connection\_date in nvarchar2, par\_disconnection\_date in nvarchar2, inserted out int);  
 *-- Обновление услуги* PROCEDURE *UpdateService*(par\_id in int, par\_contract\_id in int, par\_description\_id in int, par\_service\_amount in float, par\_connection\_date in nvarchar2, par\_disconnection\_date in nvarchar2, updated out int);  
 *-- Удаление услуги* PROCEDURE *DeleteService*(par\_id in int, deleted out int);  
 *-- Получение услуги по его идентификатору* PROCEDURE *GetServiceById*(par\_id in int, service\_cur out sys\_refcursor);  
 *-- Получение всех услуг базы данных* PROCEDURE *GetAllServices*(service\_cur out sys\_refcursor);  
 *-- Очистка базы данных от простроченных услуг* PROCEDURE *ClearExpiredServices*;  
END Service\_Package;

CREATE OR REPLACE PACKAGE TariffPlan\_Package IS  
 *-- Вставка тарифного плана* PROCEDURE *InsertTariffPlan*(par\_tariff\_name in nvarchar2, par\_tariff\_amount in float, inserted out int);  
 *-- Обновление тарифоного плана* PROCEDURE *UpdateTariffPlan*(par\_id in int, par\_tariff\_name in nvarchar2, par\_tariff\_amount in float, updated out int);  
 *-- Удаление тарифного плана* PROCEDURE *DeleteTariffPlan*(par\_id in int, deleted out int);  
 *-- Получение тарифного плана по его идентификатору* PROCEDURE *GetTariffPlanById*(par\_id in int, tariff\_plan\_cur out sys\_refcursor);  
 *-- Получение всех тарифных планов базы данных* PROCEDURE *GetAllTariffPlans*(tariff\_plan\_cur out sys\_refcursor);  
END TariffPlan\_Package;

Приложение Г

CREATE INDEX REASON\_DEBIT\_INDEX ON DEBIT (CONTRACT\_ID, REASON);  
  
CREATE INDEX CONTRACT\_SERVICE\_INDEX ON SERVICE (CONTRACT\_ID);

CREATE INDEX PN\_PHONE\_NUMBER\_INDEX ON PHONE\_NUMBER (PHONE\_NUMBER);

CREATE INDEX DISCONNECTION\_SERVICE\_INDEX ON SERVICE (DISCONNECTION\_DATE);

Приложение Д

BEGIN  
 dbms\_scheduler.*create\_schedule* (  
 schedule\_name => 'sch\_debits\_detection',  
 start\_date => *TO\_DATE*('20-04-21 11:15','DD-MM-YY HH24:MI'),  
 repeat\_interval => 'FREQ=MINUTELY; INTERVAL=1',  
 comments => 'sch\_debits\_detection MINUTELY at \*:00'  
 );  
END;  
  
BEGIN  
 dbms\_scheduler.*create\_program* (  
 program\_name => 'program\_debits\_detection',  
 program\_type => 'STORED\_PROCEDURE',  
 program\_action => 'Debit\_Package.DetectAndInsertTermDebits',  
 number\_of\_arguments => 0,  
 enabled => true,  
 comments => 'program\_debits\_detection doing DetectAndInsertTermDebits'  
 );  
END;  
  
BEGIN  
 dbms\_scheduler.*create\_job* (  
 job\_name => 'job\_debits\_detection',  
 program\_name => 'program\_debits\_detection',  
 schedule\_name => 'sch\_debits\_detection',  
 enabled => true  
 );  
END;

BEGIN  
 dbms\_scheduler.*create\_schedule* (  
 schedule\_name => 'sch\_services\_clear\_expired',  
 start\_date => *TO\_DATE*('20-04-21 11:20','DD-MM-YY HH24:MI'),  
 repeat\_interval => 'FREQ=HOURLY; INTERVAL=1',  
 comments => 'sch\_payments\_clear\_expired HOURLY at \*:00:00'  
 );  
END;  
  
BEGIN  
 dbms\_scheduler.*create\_program* (  
 program\_name => 'program\_services\_clear\_expired',  
 program\_type => 'STORED\_PROCEDURE',  
 program\_action => 'Service\_Package.ClearExpiredServices',  
 number\_of\_arguments => 0,  
 enabled => true,  
 comments => 'program\_payments\_clear\_expired doing ClearExpiredServices'  
 );  
END;  
  
BEGIN  
 dbms\_scheduler.*create\_job* (  
 job\_name => 'job\_services\_clear\_expired',  
 program\_name => 'program\_services\_clear\_expired',  
 schedule\_name => 'sch\_services\_clear\_expired',  
 enabled => true  
 );  
END;

Приложение Е

CREATE OR REPLACE TRIGGER Account\_AfterTrigger AFTER INSERT OR UPDATE OR DELETE ON ACCOUNT FOR EACH ROW  
DECLARE  
 new\_par\_event\_datetime nvarchar2(50);  
 new\_par\_message nvarchar2(200);  
 login\_par nvarchar2(50);  
 event\_inserted int;  
BEGIN  
 IF INSERTING THEN  
 login\_par := :new.LOGIN;  
 new\_par\_message := 'Created new account by login "'||login\_par||'"';  
 ELSIF UPDATING THEN  
 login\_par := :old.LOGIN;  
 new\_par\_message := 'Updated account by login "'||login\_par||'"';  
 ELSIF DELETING THEN  
 login\_par := :old.LOGIN;  
 new\_par\_message := 'Account by login "'||login\_par||'" was deleted';  
 END IF;  
 new\_par\_event\_datetime := *TO\_CHAR*(*SYSDATE*, 'YYYY-MM-DD HH24:MI:SS');  
 Account\_Package.*InsertAccountEvent*(login\_par, new\_par\_event\_datetime, new\_par\_message, event\_inserted);  
END;

Приложение Ж

CREATE OR REPLACE VIEW SERVICE\_MINER\_VIEW AS  
SELECT CONTRACT\_ID, CLIENT\_ID, DESCRIPTION\_ID as "SERVICE\_DESCRIPTION\_ID"  
FROM CONTRACT C INNER JOIN SERVICE S on C.ID = S.CONTRACT\_ID;  
  
CREATE OR REPLACE VIEW TARIFF\_MINER\_VIEW AS  
SELECT C.ID "CONTRACT\_ID", CLIENT\_ID, TARIFF\_ID  
FROM CONTRACT C INNER JOIN TARIFF\_PLAN T on C.TARIFF\_ID = T.ID;