МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

Специализация

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Сеть сотовой связи»

Выполнил студент Шабуня Алексей Сергеевич

(Ф.И.О.)

Руководитель проекта асс., Нистюк Ольга Александровна

(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)

Консультант:  асс., Нистюк Ольга Александровна

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

Нормоконтролер: асс., Нистюк Ольга Александровна

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

Курсовой проект защищен с оценкой

Минск 2021

# Реферат

Пояснительная записка курсового проекта содержит 43 страницы пояснительного текста, 47 иллюстраций, 6 источников литературы, 3 приложения.

NODE.JS, JAVASCRIPT, MICROSOFT SQL SERVER, EXPRESS.JS, NODE-MSSQL, MSNODESQLV8, QUERY STORE, PERFOMANCE MONITORING

Основной целью курсового проекта ставится разработка базы данных сети оператора сотовой связи и демонстрационного веб-приложения.

В первой главе проводится аналитический обзор литературы по тематике курсового проекта, выявление требований к будущей базе данных и анализ методов решения поставленных задач.

Вторая глава посвящена процессу разработки и включает в себя обзор основных объектов и описание алгоритмов процедур.

В третьей главе приводится разбор технических приёмов, использованных при разработке и обоснование их использования.

В четвёртой главе описан процесс тестирования разработанного веб-приложения и описание функционала некоторых элементов.

В заключении приведены результаты проделанной работы.

# Содержание

[Реферат 2](#_Toc90449004)

[Введение 4](#_Toc90449006)

[1 Аналитический обзор литературы 5](#_Toc90449007)

[2 Разработка базы данных 7](#_Toc90449008)

[2.1 Создание таблиц 7](#_Toc90449009)

[2.2 Хранимые процедуры 12](#_Toc90449010)

[2.2.1 CRUD-процедуры таблиц 13](#_Toc90449011)

[2.2.2 Выборка данных по запросу 16](#_Toc90449012)

[2.2.3 Процедуры, отвечающие за регистрацию звонков 19](#_Toc90449013)

[2.2.4 Процедуры, реализующие средства мониторинг 21](#_Toc90449014)

[2.3 Функции 29](#_Toc90449015)

[2.4 Индексы 30](#_Toc90449016)

[2.5 Триггеры 30](#_Toc90449017)

[3 Обоснование технических приемов программирования 32](#_Toc90449018)

[4 Тестирование, проверка работоспособности и анализ данных 33](#_Toc90449019)

[Заключение 39](#_Toc90449020)

[Список используемой литературы 40](#_Toc90449021)

[Приложение А 41](#_Toc90449022)

[Приложение Б 42](#_Toc90449023)

[Приложение В 43](#_Toc90449024)

# Введение

Целью данного проекта было проектирование с последующей реализацией базы данных на тему «Сеть сотовой связи». Реализованная база данных должна быть пригодна для использования в разработанном демонстрационном приложении «Оператор сотовой связи».

За основу базы данных «Сеть сотовой связи» была взята модель реляционных баз данных. Реляционная база данных – это набор данных с предопределенными связями между ними, после сравнений современных моделей баз данных, данная модель была выбрана для реализации проекта, реляционная модель отвечает необходимым требованиям: простота организации связей и понятное хранение данных в таблицах. Каждая строка представляет отдельную запись или элемент данных в таблице, который содержит значения для каждого из столбцов.

В качестве СУБД был выбрана Microsoft SQL Server, простота, производительность и надёжность выделяют её среди похожих решений.

Демонстрационная программа, веб-приложение «Оператор сотовой связи» была построена на клиент-серверной архитектуре с использованием современной, хорошо зарекомендовавшей себя платформа Node.js – отвечающая за серверную часть веб-приложения.

Node.js или Node — программная платформа, основанная на движке V8, расширяет возможности языка JavaScript предоставляя набор соответствующих инструментов, позволяющих построить полноценное веб-приложение на клиент-серверной архитектуре. Функционал Node расширяется за счёт установки модулей, которые расширяют и без того богатый функционал платформы.

Для реализации необходимого функционала на стороне сервера платформа Node.js была расширена следующими модулями:

* Express.js - гибкий веб-фреймворк для приложений Node.js, предоставляющий обширный набор функций для мобильных и веб-приложений.
* Msnodesqlv8 – TDS драйвер обеспечивающий работу с Microsoft SQL Server, основан на драйвере node-mssql.
* Node-mssql – Node.js драйвер клиента для Microsoft SQL Server.
* Bcrypt – модуль, предоставляющий набор хэш-функций для надёжного хранения данных, например пользовательских паролей.
* Cookie-parse – модуль, встраиваемый в работу Express для простой и комфортной работы с cookie-файлами

За работу клиентской части отвечают страницы, написанные на HTML с использованием CSS и JavaScript, в основном отвечающий за клиент-серверное взаимодействие.

# 1 Аналитический обзор литературы

Сотовая связь достаточно молодая область, по меркам истории, первые принципы архитектуры данной системы были изложены только в начале 1970-х годов, первая-же беларуская компания-оператор появилась только в 1993 году.

Сейчас-же наличие мобильного телефона – обыденность каждого человека, современного человека, мобильные устройства продаются миллионами каждый год, что также расширяет и количество абонентов у операторов, подобная динамика приводит к тому что надо иметь не только надёжную систему хранения всех данных об абонентах, но и соответствующие инструменты для сбора статистики и манипулирования данными связанными с клиентами – что и приводит нас к использованию баз данных, которые позволяют не только структурированно хранить информацию, но и изменять её при необходимости.

Системы учёта операторов своей клиентской базы и прочей внутренней информации – достаточно закрытая корпоративная информация, но это касается нюансов её архитектуры, её-же общее построение и принципы можно выстроить исходя из задач, которые перед ней ставятся и информации в открытом доступе.

Очевидно, что база данных оператора сотовой связи должна хранить информацию о пользователе, зарегистрированном в системе, включая его имя, фамилию, отчество, т.к. номер телефона нередко используется как первичное средство идентификации личности.

За дополнительной информации о структуре обратимся к информации в открытом доступе, рассмотрим сайт популярного в Республике Беларусь оператора «МТС», данный оператор предоставляет широкий спектр услуг, но нас интересуют только услуги, связанные с мобильной связью.

Рассмотрим один из элементов главной страницы на рисунке 1.1

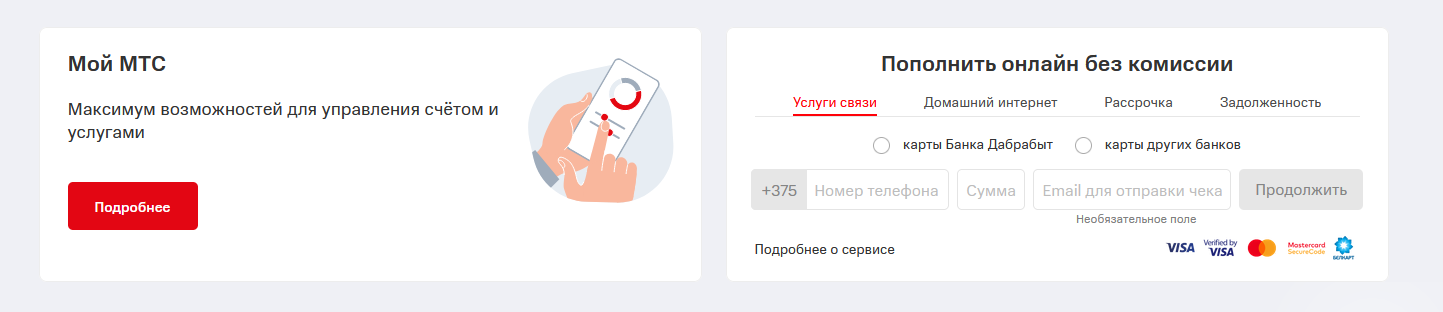


Рисунок 1.1 – Пополнение счёта оператора связи «МТС»

Помимо информации о доступе к личному кабинету мы также видим форму для пополнения баланса, пользователем из чего следует заключение, что помимо информации о пользователе мы также должны хранить информацию о его текущем балансе. Таким образом вот необходимой состав информации о пользователе:

* Имя;
* Фамилия;
* Отчество;
* Логин;
* Пароль;
* Баланс.

Рассмотрим страницу, на которой пользователь выбирает себе необходимые услуги сотовой связи:

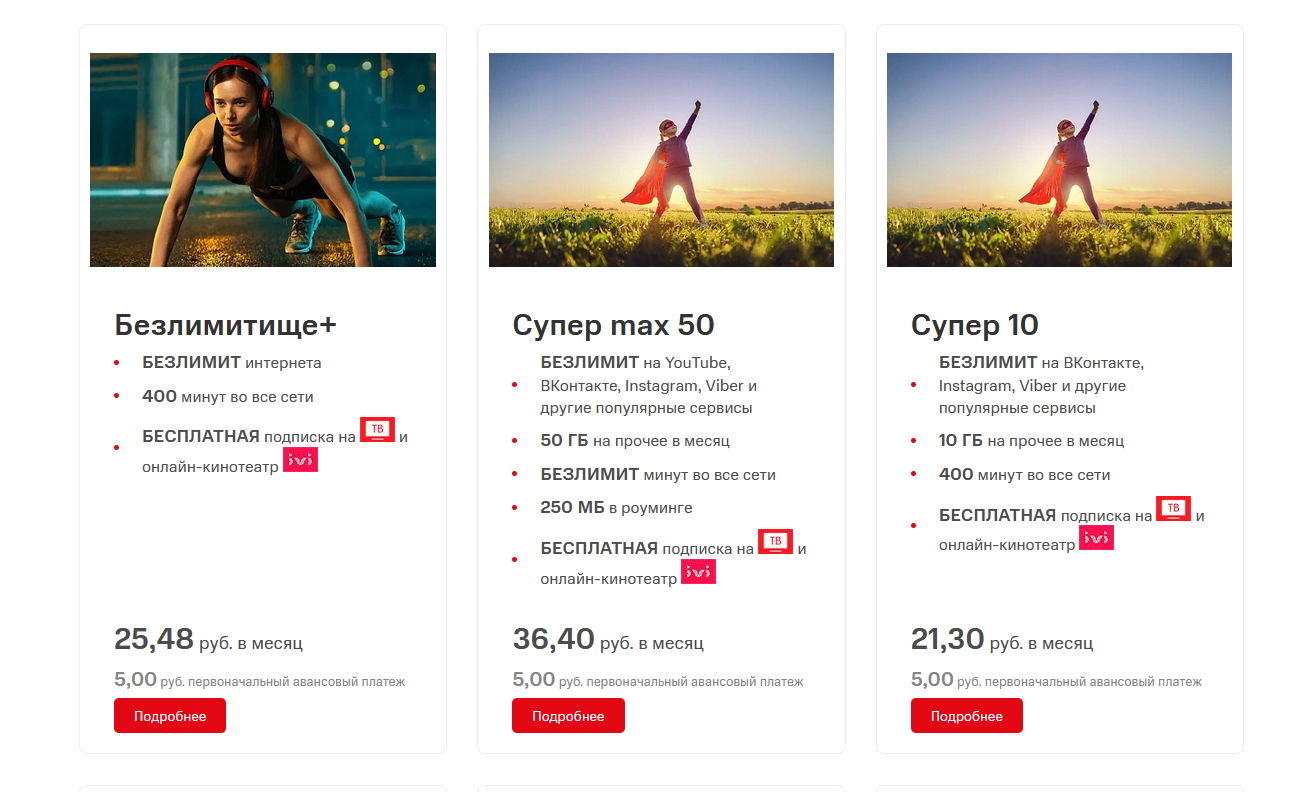


Рисунок 1.2 – Страница «Тарифа» оператора сотовой связи «МТС»

На данной странице описаны тарифы, подписавшись на которые пользователь может получать соответствующие услуги от оператора. Рассмотрев представленные тарифы на рисунке 1.2, следует очевидная структура тарифов:

* Название тарифа;
* Цена тарифа.

Имея представление о построении информации о тарифах и пользователях строим структуру связующего звена этих двух структур – информации о номере, которым владеет пользователь.

* Номер телефона;
* Указатель на пользователя;
* Указатель на тариф.

# 2 Разработка базы данных

При разработке курсового проекта ниже будут описаны следующие объекты базы данных:

* Таблицы;
* Хранимые процедуры;
* Функции;
* Индексы;
* Триггеры.

## 2.1 Создание таблиц

Для реализации работы базы данных было создано 8 таблиц, 5 основных и 3 дополнительные, 2 из которых временные, т.е. создаются только во время сессии и при определённых условиях. В данной главе введены следующие условные обозначение для некоторых специальных полей в таблице, обозначение PK – Primary Key - первичный ключ, ограничение, позволяющее однозначно идентифицировать каждую запись в таблице. FK – Foreign Key – внешний ключ, внешние ключи позволяют установить связи между таблицами.

Информация об организации основных таблиц приведена в приложении А, к основным таблицам относятся:

* USER\_TYPE;
* USERS;
* NUMBERS;
* TARIFFS;
* CALLS.

Таблица USER\_TYPE содержит в себе информации о типе пользователя – это обычный пользователь или администратор, имеющий доступ к дополнительному функционалу, такому как администрирование пользователь, просмотр всей информации в базе данных, добавлению тарифов и т.д. и содержит следующие поля:

* PK Id – уникальный идентификатор типа пользователя;
* User\_Type – тип пользователя в текстовом формате.

Скрипт для создания таблицы USER\_TYPE приведён на рисунке 2.1.

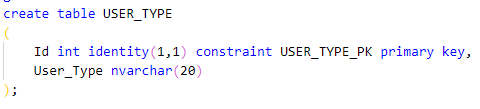


Рисунок 2.1 – Скрипт создания таблицы USER\_TYPE

Таблица USERS предоставляет информацию о пользователях, зарегистрировавшихся в системе, содержит внешний ключ к таблице USER\_TYPE и содержит следующие поля:

* PK User\_Id -уникальный идентификатор пользователя;
* User\_Name, User\_Surname, User\_MidName – имя, фамилия и отчество пользователя соответственно;
* Date\_Birth – дата рождения пользователя;
* Login – логин пользователя – уникальное имя для входа в систему;
* Password – пользовательский пароль;
* FK User\_Type – внешняя ключ к таблице USER\_TYPE – отвечает за тип зарегистрированного пользователя;
* Balance – поле, отвечающее текущий баланс пользователя;
* IsActive – статус пользователя, активен/не активен, при удалении пользователем своего аккаунта переводится в состояние IsActive: false.

Скрипт создания таблицы пользователей приведён на рисунке 2.2.

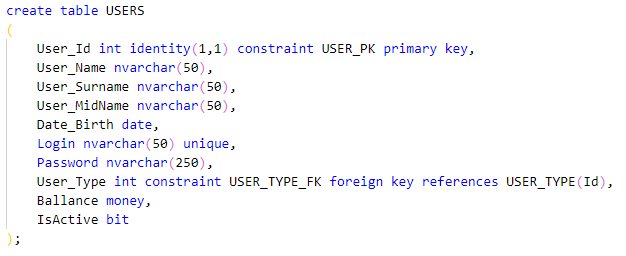


Рисунок 2.2 – Скрипт создание таблицы USERS

Таблица TARIFFS содержит информацию о тарифах, предоставляемых оператором сотовой связи, и содержит следующие поля:

* PK Tariff\_Id – уникальный идентификатор тарифа;
* Description – описание тарифа;
* Call\_Cost\_perm – стоимость звонка за одну минуту.

Скрипт создание таблицы с тарифами приведён на рисунке 2.3.

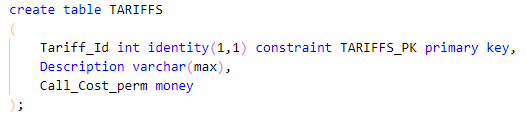


Рисунок 2.3 – Скрипт создания таблицы TARIFFS

Таблица NUMBERS отвечает за хранение информации о номерах, выданных пользователям, один пользователь может иметь несколько номеров, один номер может иметь только один тариф. Таблица NUMBERS содержит два внешних ключа, с таблицей USERS для связи номера и пользователя и с таблицей TARIFFS для связи номера и тарифа.

Поле с номером само по себе является уникальным для каждой записи, но не может быть выбрано в качестве первичного ключа, т.к. реализованная база данных предусматривает хранение информацию о звонках, которая содержит номер отправителя и получателя. Может возникнуть ситуация, когда пользователь отказался от номера, но информация о звонке должна хранится пока не будет удалена информация о пользователе, поэтому в таблицу добавлен отдельный уникальный идентификатор.

Таблица содержит следующие поля:

* PK Number\_Id – уникальный идентификатор номера;
* Number – уникальный шестизначный номер для каждой записи;
* FK User\_Id – внешний ключ к таблицу USERS с идентификатором пользователя, которому принадлежит номер;
* FK Tariff\_Id – внешний ключ к таблице TARIFFS с идентификатором тарифа, закреплённым за номером;
* Date\_Open – дата выдачи номера пользователю.

Скрипт создания таблицы, описывающей номера приведён на рисунке 2.4.

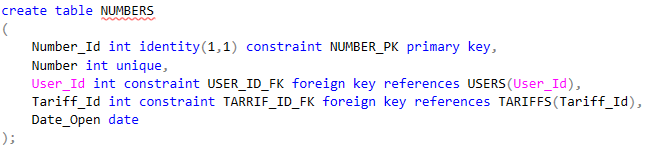


Рисунок 2.4 – Скрипт создания таблицы NUMBERS

Таблица CALLS содержит информацию о звонках, совершаемых между пользователями, пользователь не может звонить на номер, с которого звонит, но может звонить на другой номер, закреплённый за ним, как и на номера, принадлежащие другим пользователям. Таблица содержит два внешних ключа на таблицу USERS отвечающих за связь того, кто совершает звонок – отправителя (User\_Sender\_Id) и того, кому звонили – получателя (User\_Receiver\_Id), таблица CALLS содержит следующие поля:

* PK Call\_Id – уникальный идентификатор звонка;
* FK User\_Sender\_Id – внешний ключ на таблицу USERS, отвечающий за уникальный идентификатор отправителя;
* User\_Sender\_Number – номер с которого звонил пользователь;
* FK User\_Receiver\_Id – внешний ключ на таблицу USERS, отвечающий за уникальный идентификатор получателя:
* User\_Receiver\_Number – номер по которому пользователь совершил звонок;
* Call\_Time – общее время звонка в секундах;
* Call\_Cost – стоимость звонка высчитанная согласно тарифу, закреплённому за номером с которого звонил пользователь (User\_Sender\_Number).

Скрипт создания таблицы звонков, совершённых пользователями представлен на рисунке 2.5.

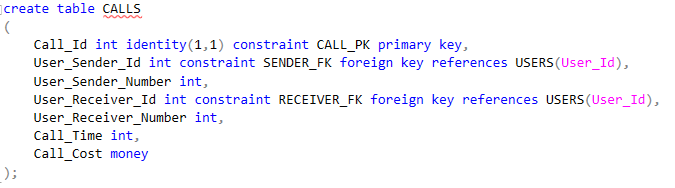


Рисунок 2.5 – Скрипт создания таблицы CALLS

Далее будут описаны дополнительные таблицы заложенные при проектировании базы данных.

Постоянна таблица CUDLogAllTime – хранит в себе информацию о некоторых операциях DML, а конкретнее об операциях INSERT, UPDATE, DELETE и относится к средству мониторинга базы данных. Данная таблица задействована в триггерах, описанных ниже в главе 2.5, после срабатывания триггера операция записывается в таблицу и хранит операцию, время в которое она была совершена, а также значение до операции для операций UPDATE и DELETE и последнее (новое) для операций UPDATE и INSERT. CUDLogAllTime содержит следующие столбцы:

* PK OpId – уникальный идентификатор операции в базе данных;
* OperationKey – ключ для поиска операции I, U, D для операций INSERT, UPDATE и DELETE соответственно;
* Operation – полной название операции;
* TableName – название таблицы над которой производилась операция;
* BeforeValue – запись, хранящаяся в базе данных до операции, в строковом представлении, null для операции INSERT;
* AfterValue – запись, получившаяся в результате операции над таблице, в строковом представлении, null для операции DELETE;
* Date – дата и время, в которое была произведена операция

Скрипт создания таблицы с логами операций CREATE, UPDATE, DELETE приведён на рисунке 2.6.

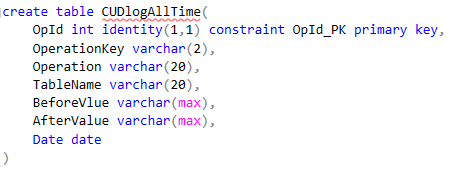


Рисунок 2.6 – Скрипт создания таблицы CUDlogAllTime

Временная таблица ##CUDlogSession содержит те-же поля и хранит ту-же информацию что и таблица CUDlogAllTime, только для операций, совершённых в текущую сессию. Взаимодействует с теми-же триггерами что и CUDlogAllTime, основное отличие в их создании, таблица CUDlogAllTime хранится в базе данных постоянно, таблица ##CUDlogSession создаётся при первой-же DML операции совершённой в текущую сессию. Таблица содержит следующие поля:

* PK OpId – уникальный идентификатор операции в базе данных;
* OperationKey – ключ для поиска операции I, U, D для операций INSERT, UPDATE и DELETE соответственно;
* Operation – полной название операции;
* TableName – название таблицы над которой производилась операция;
* BeforeValue – запись, хранящаяся в базе данных до операции, в строковом представлении, null для операции INSERT;
* AfterValue – запись, получившаяся в результате операции над таблице, в строковом представлении, null для операции DELETE;
* Date – дата и время, в которое была произведена операция

Скрипт создания таблицы логов за текущую сессию приведён на рисунке 2.7.

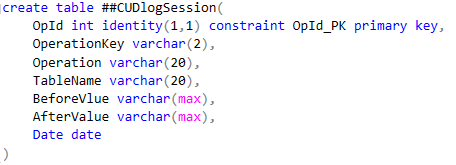


Рисунок 2.7 – Скрипт создания таблицы ##CUDlogSession

Временная таблица ##FreeNumbers создана для решения проблемы с образованием пробелов в последовательности номеров в таблице NUMBERS. При выдаче пользователю номера ему выдаются номер с приращением на 1 с последним зарегистрированным номер в таблице NUMBERS, т.е. последний номер + 1.

При использовании базы данных может возникнуть ситуация, при которой пользователь решил отказаться от номера из середины списка номеров, таким образом нарушается последовательность в таблице. Для решения этой проблемы при старте сессии в таблицу ##FreeNumbers записываются пропущенные номера в упорядоченной последовательности между первым (100000 по умолчанию) и последним зарегистрированным номерами из таблицы NUMBERS, таким образом, если в таблице ##FreeNumbers имеются записи новым пользователям сперва будут выдавать номера из этой таблицы, а после того как она опустеют уже будут выдавать следующие в последовательности номера.

Таблица содержит одно поле с номером, скрипт создания приведён на рисунке 2.8. Для начала он проверяет существует ли она, на случай если таблица уже создана, если нет, она создаётся.

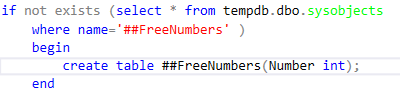


Рисунок 2.8 – Скрипт создания таблицы ##FreeNumbers

## 2.2 **Хранимые процедуры**

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

При разработке курсового проекта была создано 41 процедура. Некоторые из них имеют схожий функционал, например процедуры, реализующие CRUD-функции (Create, Read, Update, Delete), но разные параметры.

Все процедуры можно разделить на 5 категории для выполнения следующих задач:

1. Процедуры, реализующие CRUD-функции для каждой таблицы;
2. Процедуры, реализующие выборку данных по запросу;
3. Процедуры, отвечающие за работу с таблицей CALLS и регистрацией звонков, совершённых пользователями;
4. Процедуры, реализующие средство мониторинга базы данных
5. Процедуры выполняющие операции, которые нельзя отнести ни в одну из категорий вынесены в специальные процедуры.

Далее мы более подробно остановимся на каждой из категорий.

### 2.2.1 CRUD-процедуры таблиц

Все CRUD функции для каждой операции для всех таблиц построены по одному шаблону, за исключением, для операций CREATE они принимают параметры для каждого поля таблица или те, которые должны быть обновлены для UPDATE. Уникальный идентификатор в качестве параметра передаётся для операций DELETE и процедур Find, осуществляющих READ-операции для поиска записи по идентификатор. READ-операцией также являются процедуры All для вывода всех записей из таблицы.

Чтобы избежать повторений и из-за шаблонности CRUD-процедур все примеры будут рассмотрены на примере процедур для одной таблицы – таблица USERS.

Процедуры, реализующие функцию CREATE, в нашем случае INSERT именуются по следующему шаблону:

<Имя объекта, который хранит таблица> Add

Для таблицы USERS – это UserAdd. Все CREATE-процедуры принимают в качестве параметров все поля, описывающие таблицу, т.к. валидация производится в приложении, то все параметры сразу добавляются в таблицу посредством операции INSERT, пример скрипта представлен на рисунке 2.9.

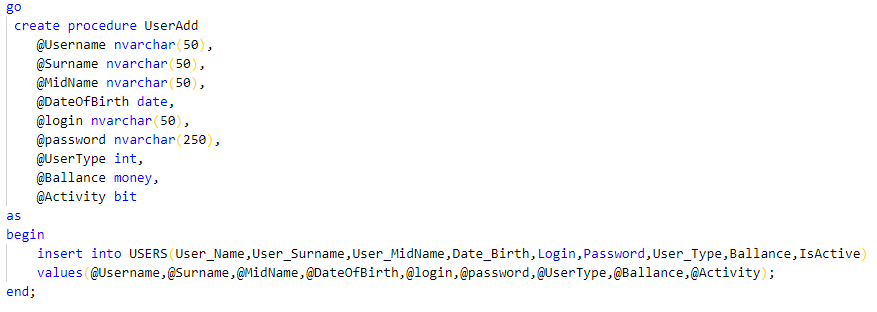


Рисунок 2.9 – СREATE-процедура на примере процедуры UserAdd

Следующей в последовательности CRUD-процедур – READ-процедуры, их всех можно разделить на процедуры, осуществляющие вывод одной записи и всех.

Процедуры для вывода или поиска одной записи – Find-операции именуются по следующем шаблону:

Find< Имя объекта, который хранит таблица >

Для таблицы USERS – это FindUser. Все Find-операции принимают один параметр – уникальный идентификатор записи в таблице, т.к. все таблицы содержат Primary Key создание индекса для данной операции не требуются, т.к. индекс создаётся автоматически. После того как уникальный индекс передан процедура ищет в среди записей таблицы ту, уникальному идентификатору которой соответствует переданный параметр, пример скрипта представлен на рисунке 2.10.

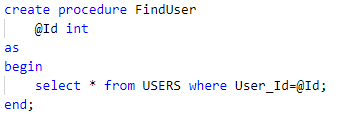


Рисунок 2.10 – Пример READ-операции типа Find

Процедуры для вывод всех записей таблицы – All-операции именуются по следующему шаблону:

All< Имя объекта, который хранит таблица во множественном числе>

Для таблицы USERS – это AllUsers. Все All-операции принимают два параметра – номер первой строки, которую следует вывести и последней, т.к. некоторые таблицы должны содержать по 100 000 строк, выводить все 100 000 на экран было-бы избыточно, поэтому на стороне базы данных производится выборка из n строк в диапазоне переданных параметров. Пример скрипта представлен на рисунке 2.11.

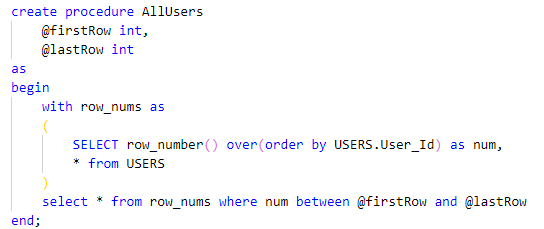


Рисунок 2.11 – Пример READ-операции типа All

Далее будет рассмотрен пример UPDATE операции, они именуются по следующему шаблону:

<Имя объекта, который хранит таблица> Update

И могут принимать все параметры, которые описывают таблицу, т.е. UPDATE-операции – процедуры, в которых все параметры кроме уникального идентификатора, по которому будет осуществляться поиск записи для обновления, не обязательны. Каждый параметр соответствует полю таблицы, после того как параметры, которые требуются обновить и уникальные идентификатор переданы, все не переданные параметры приравниваются к NULL, во избежание обновления полей, которые в этом не нуждаются. Параметры для обновления и уникальный идентификатор переданы, остальные приравнены к NULL, процедура последовательно опрашивает параметры, если он передан, то поле, за которым закреплён параметр обновляется на переданное значение, если параметр NULL, то поле этой записи не изменяется. Пример скрипта представлен на рисунке 2.12.

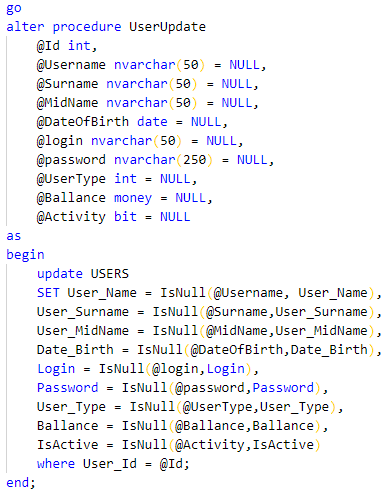


Рисунок 2.12 – Пример UPDATE-операции

Процедуры, отвечающие за удаление также шаблонны. Они именуются следующим образом:

<Имя объекта, который хранит таблица> Delete

Для таблицы USERS – это UserDelete. Все DELETE-процедуры принимают один параметр – уникальный идентификатор записи таблицы, после чего осуществляют поиск по ней, найдя запись для удаления – удаляют её. Пример скрипта приведён на рисунке 2.13.

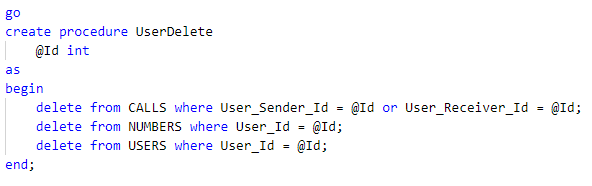


Рисунок 2.13 – Пример DELET-операции

### 2.2.2 Выборка данных по запросу

Следующей группой процедур являются процедуры, осуществляющие выборку данных по сформированному SELECT-запросу. В данную категорию попадают процедуры, результатом которых является результирующий набор, полученный из SELECT-запроса или результат, сформированный на его основе, также процедуры, занесённые в данную категорию, не выполняет функционал какой-либо другой группы.

В ходе проектирования базы данных одним из требований к ней была возможность пользователя получать доступ к своим данным и к результатам своих действий в базе данных, почти все функции данной группы направленны на работу с пользователем.

Первой задачей было осуществить пользователю, собственно, сам доступ, а также ограничить его доступ к тем данным, просматривать которые и осуществлять какие-либо операции он не может из-за отсутствия соответствующих привилегий. Самым простым и очевидным способом осуществления доступа является механизм авторизации, он и был выбран в качестве решения данной проблемы.

Уникальным идентификатором пользователя является не только Primary Key User\_Id таблицы USERS, но и поле Login с ограничением unique (уникальное значение на всю таблицу), которое задаёт сам пользователь и по которому он может себя идентифицировать в базе данных. Помимо логина пользователь задаёт пароль для своей учётной записи.

Таким образом механизм авторизации работает на совпадении введённых пользователем логина и пароля с соответствующими полями в базе данных. Программной реализацией механизма авторизации на стороне базы данных является процедура FindUserByLogin.

Процедура FindUserByLogin принимает один параметр – логин, введённый пользователем при попытке авторизации. Получив параметр логин, процедура осуществляет поиск записи в таблице USERS, значение Login которой соответствует переданному параметру, таким образом функция реализует первую часть механизма авторизации – идентификация пользователя по логину. Для обеспечения надёжности доступа к пользовательским данным никто кроме пользователя, установившего пароль, не может его знать или посмотреть, это осуществляется за счёт хэширования пароля на серверной стороне Node при помощи хэш-функций из модуля Bcrypt.

Результатом процедуры является передача записи о пользователе из таблицы с совпавшим логином на Node-сервер, запись включает не только пароль, совпадение которого и предоставляет доступ к функционалу веб-приложения, но и поле FK User\_Type, значение которого может разблокировать функции администратора, если пользователь обладает соответствующими правами. Скрипт процедуры приведён на рисунке 2.13.

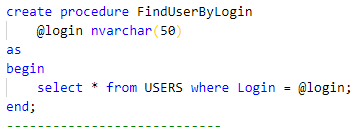


Рисунок 2.13 – Скрипт процедуры FindUserByLogin

В базе данных реализован механизм регистрации звонков, совершённых пользователем, более подробно его реализацию мы рассмотрим в данной главе ниже при описании процедур, осуществляющих работу со звонками. Для звонка пользователю необходимо иметь свой номер и знать номер, по которому он хочет позвонить. Если пользователь задался целью совершить звонок, то предполагается что он знает номер, по которому хочет позвонить, но разумно допущение, что обладая несколькими или многими номерами пользователь может не помнить их все, эту проблему решает процедура FindUserNumbers.

Помимо предоставления всех имеющихся у пользователя номеров для страницы, на которой осуществляется звонок, данная процедура также задействована для предоставления более подробной информации о номерах для страницы профиля, на которой помимо самого номера пользователь может также узнать, какой тариф закреплён за номером и когда номер был ему выдан.

Процедура FindUserNumbers принимает 3 параметра – уникальный идентификатор пользователя для выборки номеров конкретного пользователя, а также необязательные параметры первой и последней строки выборки, назначение и применение которых было описано в данной главе выше, когда мы рассматривали All-типы READ-операций. Также в данной процедуре предусмотрено ветвление, которое в случае, если заданы параметры для выборки строк, возвращает n строк, а при их отсутствии для страницы, с которой осуществляется звонок, выводятся все номера пользователя. Скрипт процедуры приведён на рисунке 2.14.

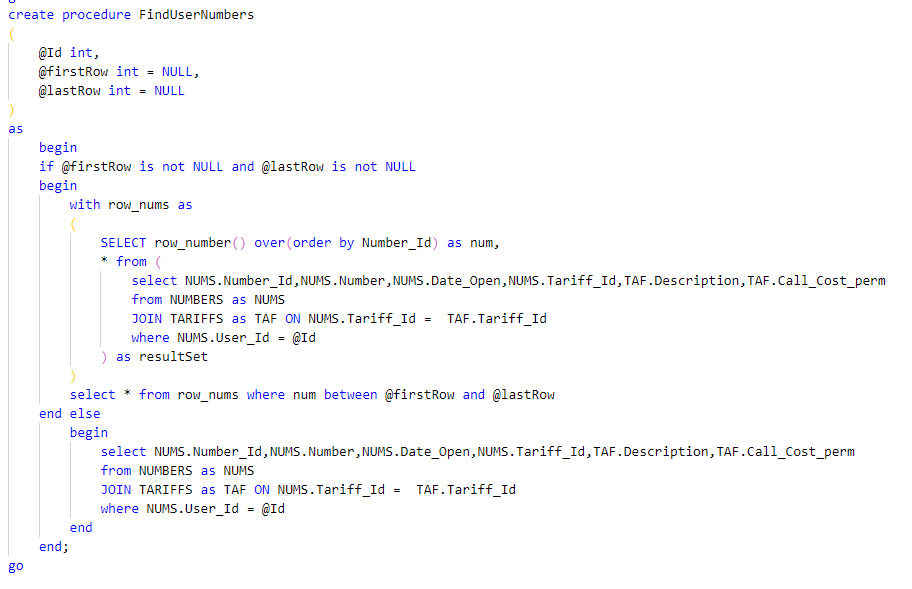


Рисунок 2.13 – Скрипт функции FindUserNumbers

Помимо информации о зарегистрированных на пользователя номерах на странице профиля также пользователь может ознакомится со совей историей звонков. За реализацию данной возможности отвечает процедура FindUserCalls.

Процедура для отображения истории звонков работает по тому-же принципу что и All-типы READ-операций, и процедура FindUserNumbers, она принимает 3 параметра – уникальный идентификатор пользователя, первую и последнюю строку для выборки количества выводимых номеров. Получив параметр уникального идентификатора, процедура осуществляет поиск по таблицу CALLS на совпадение переданного параметра с полем отправителя – звонившего пользователя (User\_Sender\_Id), результирующим набором запроса является список из n звонков, в заданном диапазоне, которые совершил пользователь. Скрипт процедуры приведён на рисунке 2.14.

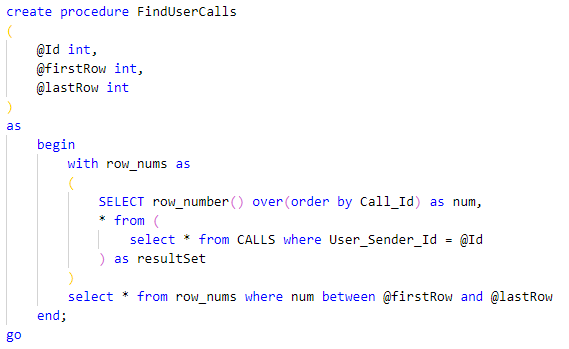


Рисунок 2.14 – Скрипт процедуры FindUserCalls

### 2.2.3 Процедуры, отвечающие за регистрацию звонков

Выше в данной главе мы уже вскользь затрагивали некоторые процедуры, которые принимают участие в регистрации звонков, в данной-же части мы рассмотрим процедуры, которые непосредственно отвечают за механизм пользовательских звонков.

Функционал механизма пользовательских звонков при проектировании был заложен в серверную часть веб-приложения, но на этапе разработки, когда стала необходимость в написании дополнительных процедур, чтобы избежать большого количества последовательных запросов к базе данных было принято возложить реализацию задачи на базу данных

Выше в данной главе при рассмотрении процедуры FindUserNumbers уже было обозначено, что минимально необходимая информация для совершения звонка – это номер отправителя (звонившего) и получателя (того, кому звонят), получение данной информации уже было описано выше, далее будут рассмотрены процедуры, параметрами которых являются данные переменные и которые непосредственно реализуют механизм регистрации звонков, он представлен двумя процедурами CallStart и CallEnd, для начала и завершения звонка соответственно.

Для начала рассмотрим процедуру, которая отвечает за начала звонка – CallStart, на неё возложена, на мой взгляд, более ответственная задача – зарегистрировать звонок в системе, сама операция не очень сложная, но т.к. для оператора сотовой связи важно иметь информацию о совершённых пользователям звонках, если звонок прервётся запись всё равно останется – что является одним из требований для спроектированной базы данных, поэтому данная процедуры и выделена как более важная в механизме регистрации звонков.

Процедура принимает 3 параметра – номер отправителя, номер получателя и время продолжительности на момент вызова процедуры. После получения всех параметров первой итерацией алгоритма является поиск пользователей, которым принадлежат номера. Напоминаем, из-за допущения, что номер, по которому производился звонок, может быть позже удалён из базы данных, регистрация звонков происходит для пользователей по их уникальным идентификаторам, а не по номерам. Для поиска пользователей, которым принадлежат номера используется функция FindUser\_IdByNumber, алгоритм которой будет рассмотрен ниже, в разделе, описывающим функции базы данных, на данном этапе нам важно знать, что она возвращает идентификатор пользователя, которому принадлежит номер. Получив результаты выполнения функции для отправителя и получателя, в случае если оба идентификатора получены, процедура заносит в таблицу CALLS, строку о данном звонке, передавая в качестве стоимости звонка 0, т.к. звонок ещё не завершён, расчёт стоимости будет произведён после его завершения, процедура возвращает уникальный идентификатор записи звонка, по которому тот будет завершаться, т.е. будет отмечена продолжительность звонка и произведён расчёт стоимости. В случае, если отправитель или получатель не был найден – это свидетельствует о том, что переданного номера не существует и возвратом процедуры будет сигнал об ошибке с кодом -1. Скрипт процедуры приведён на рисунке 2.15.

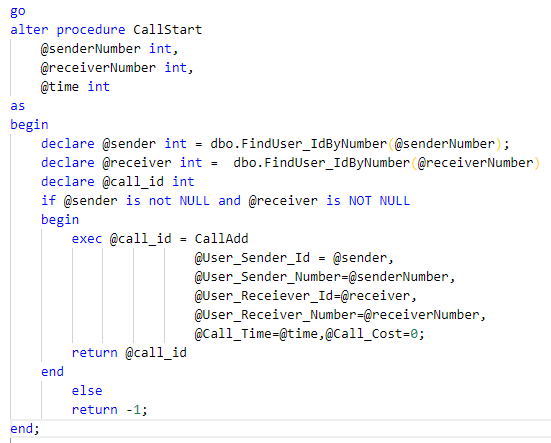


Рисунок 2.15 – Скрипт процедуры CallStart

Следующей и завершающей итерацией в механизме регистрации звонков является процедура CallEnd, которая отвечает за модификацию записи в таблице CALLS, добавляя время звонка и рассчитанную стоимость.

Процедура принимает 3 параметра – уникальный идентификатор звонка для поиска записи, которую мы будем модифицировать, номер отправителя и общее время звонка. Вторым параметрам мы передаём номер получателя, он уже есть в таблице, но зачем делать дополнительный запрос к таблице CALLS, если номер уже хранится на сервере веб-приложения и его можно передать для ускорения выполнения процедуры. По номеру отправителя мы извлекаем запись о стоимость звонка за минуту из тарифа, закреплённого за номером, это значение нам понадобится для расчёт общей стоимости звонка. Стоимость звонка рассчитывается по следующей формуле: (стоимость звонка) / (общее время звонка в секундах /60), т.к. стоимость звонка за минуту хранится в SQL-типе money, для расчёта стоимости и изменения баланса пользователя все участвующие в вычислениях переменные приводятся к эквивалентному money типу decimal (19,4). После расчёта стоимости производится UPDATE-операция в таблице CALLS, изменяющая информацию о времени стоимости звонка и аналогичная операция, изменяющая текущий баланс пользователя. Скрипт процедуры приведён на рисунке 2.16.

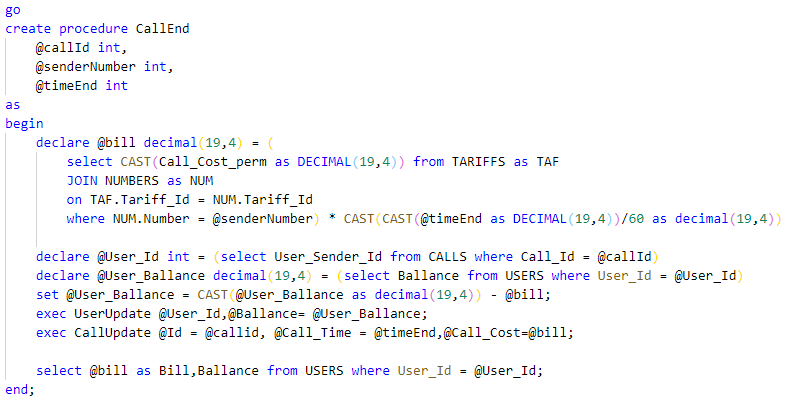


Рисунок 2.16 – Скрипт процедуры CallEnd

### 2.2.4 Процедуры, реализующие средства мониторинг

В данном разделе второй главы мы более подробно остановимся на процедурах, отвечающих за дополнительную технологию, реализованную в данной базе данных – средства мониторинга состояния базы данных.

Для понимания в каком состоянии прибывает база данных одним из знаний, которым необходимо обладать – это количество объектов в базе данных, для этого была разработана соответствующая процедура DBObjCount, скрипт которой приведён на рисунке 2.17.

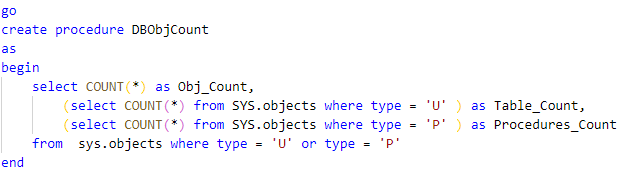


Рисунок 2.17 – Скрипт процедуры

Описанная процедура не принимает параметров и её результирующим набором является строка, содержащая общее количество объектов, созданных разработчиком в базе данных, значение которого складывается из суммы хранящихся таблиц – записей с типом U, отвечающем за USER\_TABLE и суммы хранимых процедур – суммы записей с типом P, отвечающем за SQL\_STORED\_PROCEDURE.

Все остальные процедуры данной главы можно поделить на две группы, первая – процедуры реализующие сбор статистической информации о количестве вызванных процедур, последних вызовах в базе данных и информации о наиболее длительных запросов, выполняющихся в базе данных для рекомендаций по оптимизации разработанных процедур, с расширением базы данных, некоторые процедуры могут нуждаться в оптимизации. И вторая – процедура, отвечающие за логирование CUD-операций в базе данных, они позволяют отслеживать изменения, происходящие с записями в базе данных.

Для процедур первой группы был использован встроенный в Ms SQL Server инструмент хранилище запросов.

Хранилище запросов SQL Server предоставляет подробные сведения о выборе и производительности плана запроса. Оно упрощает устранение неполадок с производительностью, помогая быстро находить разницу в производительности, вызванную изменением плана запроса. Хранилище запросов автоматически собирает журнал запросов, планов и статистики выполнения, сохраняя эти данные для просмотра. Данные разделяются по временным диапазонам, благодаря чему вы можете просматривать закономерности использования и узнавать об изменениях плана запроса на сервере. Хранилище запросов можно настроить с помощью инструкции [ALTER DATABASE SET](https://docs.microsoft.com/ru-ru/sql/t-sql/statements/alter-database-transact-sql-set-options?view=sql-server-ver15).

Основной функционал процедур, работающих с хранилищем запросов строится на работе со следующими представлениями хранилища запросов:

* [sys.database\_query\_store\_options (Transact-SQL)](https://docs.microsoft.com/ru-ru/sql/relational-databases/system-catalog-views/sys-database-query-store-options-transact-sql?view=sql-server-ver15)
* [sys.query\_context\_settings (Transact-SQL)](https://docs.microsoft.com/ru-ru/sql/relational-databases/system-catalog-views/sys-query-context-settings-transact-sql?view=sql-server-ver15)
* [sys.query\_store\_plan (Transact-SQL)](https://docs.microsoft.com/ru-ru/sql/relational-databases/system-catalog-views/sys-query-store-plan-transact-sql?view=sql-server-ver15)
* [sys.query\_store\_query (Transact-SQL)](https://docs.microsoft.com/ru-ru/sql/relational-databases/system-catalog-views/sys-query-store-query-transact-sql?view=sql-server-ver15)
* [sys.query\_store\_query\_text (Transact-SQL)](https://docs.microsoft.com/ru-ru/sql/relational-databases/system-catalog-views/sys-query-store-query-text-transact-sql?view=sql-server-ver15)
* [sys.query\_store\_wait\_stats (Transact-SQL)](https://docs.microsoft.com/ru-ru/sql/relational-databases/system-catalog-views/sys-query-store-wait-stats-transact-sql?view=sql-server-ver15)
* [sys.query\_store\_runtime\_stats (Transact-SQL)](https://docs.microsoft.com/ru-ru/sql/relational-databases/system-catalog-views/sys-query-store-runtime-stats-transact-sql?view=sql-server-ver15)
* [sys.query\_store\_runtime\_stats\_interval (Transact-SQL)](https://docs.microsoft.com/ru-ru/sql/relational-databases/system-catalog-views/sys-query-store-runtime-stats-interval-transact-sql?view=sql-server-ver15)
* [sys.query\_store\_query\_hints (Transact-SQL)](https://docs.microsoft.com/ru-ru/sql/relational-databases/system-catalog-views/sys-query-store-query-hints-transact-sql?view=sql-server-ver15)

Обращаясь к ним используя возможности соединений SQL мы получаем информацию о идентификаторе запроса, его тексте, среднем времени выполнения, количестве операций ввода вывода, плане выполнения. Используя соединение с представлением sys.object мы можем связать текст запросы с объектом базы данных.

Первая, из трёх, процедур оперирующих представлениями хранилища запросов – процедура LastExecs, она выводит первые 20 процедур, которые последними были вызваны в базе данных, путём множественных соединений мы получаем значение last\_execution\_tim из представление sys.query\_store\_runtime\_stats, которое отвечает за время последнего вызова. Соединяя идентификатор запроса из представления sys.query\_store\_query с идентификатором запроса из представления sys.objects мы получаем название процедуры, данная операция будет производится и в двух далее описанных процедурах, поэтому там её подробное описание будет опущено. Процедура LastExecs – очевидно после своего вызова является последней вызванной в базе данных, поэтому из результирующего набора она исключается. Скрипт процедуры приведён на рисунке 2.18.

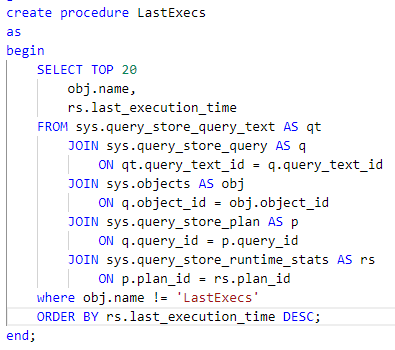


Рисунок 2.18 – Скрипт процедуры LastExecs

Следующей процедурой средства мониторинга базы данных является процедура ProcExecsCount, результирующим набором которой является список из пары наименование вызванного объекта и количества его вызовов. За данный параметр отвечает поле total\_execution\_count представления sys.query\_store\_runtime\_stats. Используя секцию ORDER BY ранжирование производится от наиболее часто вызываемого объекта к наименее частому. Скрипт процедуры приведён на рисунке 2.19.

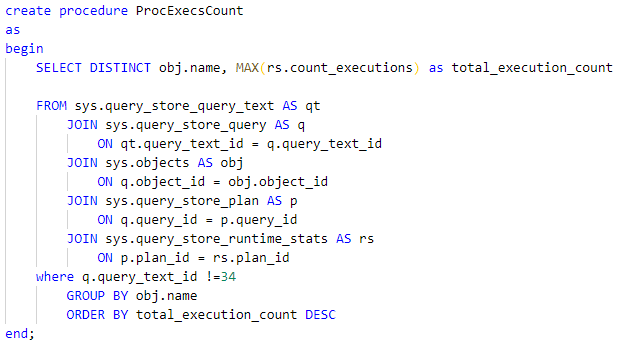


Рисунок 2.19 – Скрипт процедуры ProcExecsCount

Последняя процедура реализованной технологии из первой группы – процедура LongestAVGexecTime, которая высчитывает среднее значение выполнения операции в базе данных – данная процедура может сигнализировать о необходимости оптимизировать тот или иной запрос. Процедура суммирует среднее значение всех операций процедуры используя поле avg\_duration представления sys.query\_store\_runtime\_stats и выводит 10 запросов, среднее выполнение которых является наивысшим. Скрипт процедуры приведён на рисунке 2.20.

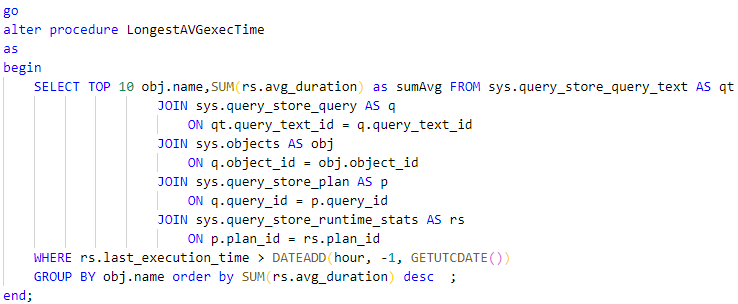


Рисунок 2.20 – Скрипт процедуры LongestAVGexecTime

В описании таблиц, спроектированных в базе данных было упомянуто, что логирование CUD операций происходит на основе DML-триггеров, в данном разделе мы рассмотрим вторую группу процедур средства мониторинга базы данных – процедуры, отвечающие за логирование.

Основной процедурой механизма логирования является процедура CUDlog, именно она вызывается при срабатывании триггеров и заносит информацию о изменениях таблицы логов. Процедура принимает 5 параметров, идентичных полям в таблицах ##CUDlogSession и CUDlogAllTime – ключ операции, название DML-операции, имя таблицы для которой сработал триггер, предыдущее значение записи и обновлённое. После вызова процедура проверяет создана ли таблицы ##CUDlogSession, в которой хранятся логи текущей сессии и если это была первая DML-операция и таблица не создана, то CUDlog создаёт её, после чего происходит INSERT данных об изменениях в таблицы ##CUDlogSession и CUDlogAllTime. Скрипт процедуры приведён на рисунке 2.21.

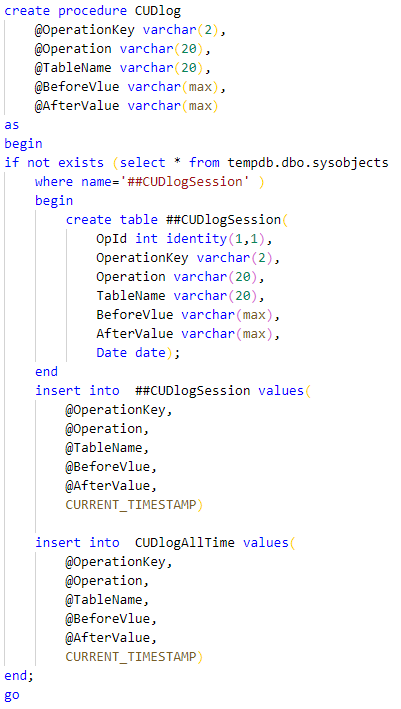


Рисунок 2.21 – Скрипт процедуры CUDlog

Данные из таблиц логов можно посмотреть на странице мониторинга, обладая правами администратора, для этого разработаны две процедуры – LogInfo и LogInfoSession, выводящий информацию о логах за весь период и за сессию, соответственно. Выборка данных в этих процедурах из таблиц логов происходит согласно требованиям, сконфигурированным администратором, по определённой дате, за промежуток времени, определённой CUD-операции или над какой-то таблице. Описанный функционал реализован за счёт операции except (исключения) над множествами всех операций или выборке по дате, операций над определённой таблицей и конкретной CUD-операции. Все параметры являются необязательными и в случае, если ни один из них не передан процедура возвращает все операции из таблицы. Алгоритмы в процедурах LogInfo и LogInfoSession идентичны только производятся для разных таблиц – CUDlogAllTime и ##CUDlogSession соответственно, поэтому в приложении Б представлен только скрипт процедуры LogInfo, т.к. скрипт LogInfoSession ему идентичен.

Для сбора статистики по проведённым CUD-операциям был разработан пользовательский тип LogStats – включающий следующие поля:

* InsertCount – количество операций INSERT за всё время;
* UpdateCount – количество операций UPDATE за всё время;
* DeleteCount – количество операций DELETE за всё время;
* InsertCountSession – количество операций INSERT за сессию;
* UpdateCountSession – количество операций UPDATE за сессию;
* DeleteCountSession – количество операций DELETE за сессию;
* OperationCount – количество операций за всё время;
* OperationCountSession – количество операций за сессию.

С разработанным пользовательским типом взаимодействует процедура LogInfoCUDCount, которая и задаёт поля типа соответствующими значениями, проводя подсчёт тех или иных операций из таблиц ##CUDlogSession и CUDlogAllTime. Процедура, в качестве результата, возвращает заполненные тип LogStats. Скрипт приведён на рисунке 2.22

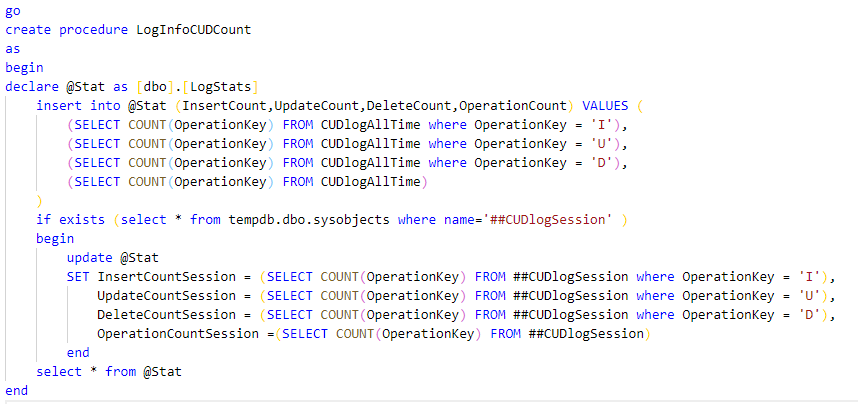


Рисунок 2.22 – Скрипт процедуры LogInfoCUDCount

Последняя, пятая группа процедур – специальные процедуры – процедуры, выполняющие функционал, который нельзя отнести ни к одной из описанных выше групп.

Выше в тексте была описана одна из проблем, которая может возникнуть по мере эксплуатирования базы данных – нарушение последовательности номеров в таблице NUMBERS, для решения этой проблемы на этапе проектирования были заложены две процедуры – SetFreeNums и GetNumber, первая отвечает за поиск нарушений последовательности, вторая за выдачу номера пользователю.

Начнём с определения пробелов в последовательности номеров, процедура SetFreeNums взаимодействует с таблицей ##FreeNumbers, описанной в разделе создания таблиц, туда она заносит найденные свободные номера.

При инициализации сессии именно процедура SetFreeNums запускается первой, скрипт приведён на рисунке 2.23.

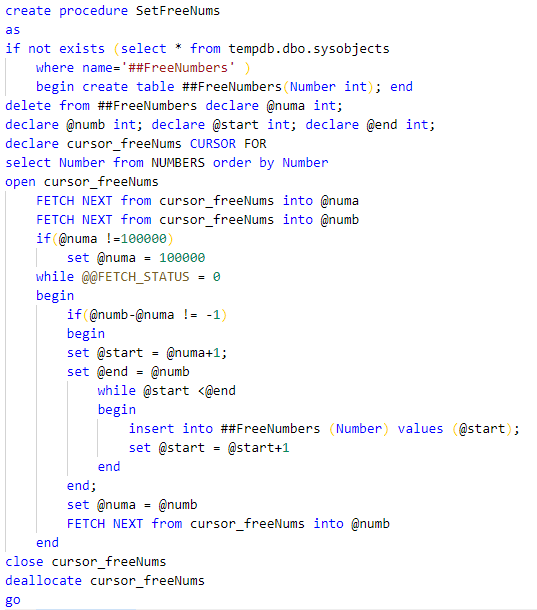


Рисунок 2.23 – Скрипт процедуры SetFreeNums

Использую статический курсор процедура проходит по всем записям таблицы NUMBERS, проверяя удовлетворяет ли упорядоченная последовательность номеров таблицы условию: значение n+1 записи (следующей) = n значению (предыдущей) +1 или проще говоря последовательности с приращением на единицу. В случае если пара значений следующее-предыдущее не удовлетворяет описанному выше условию вычисляется промежуток между ними – этот промежуток и будет равен количеству пропущенных номеров, например: для n номера 100032 и n+1 номера 100040 количество пропущенных номеров будет равно 7 или последовательности от100033 до 100039, вычисленная последовательность недостающих номеров записывается в таблицу ##FreeNumbers, наличие записей в которой говорит о том что вначале номера пользователям будут выдаваться оттуда, а уже после последовательность будет дополнятся следующими по порядку значениями, за механизм выдачи номеров отвечает процедура GetNumber.

GetNumber, скрипт представлен на рисунке 2.24, выдаёт номера пользователям по следующему принципу, если таблица ##FreeNumbers существует и в ней есть хотя-бы одна запись, номера пользователями будут выдаваться из таблицы ##FreeNumbers, пока та не опустеет, после того как она опустеет процедура выдает пользователю номер равный последнему номеру в упорядоченной последовательности номеров +1.

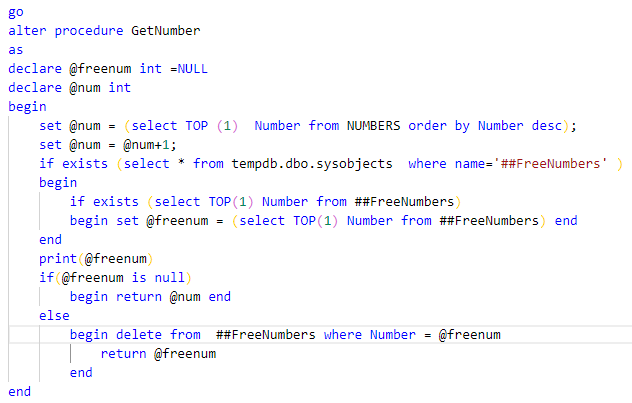


Рисунок 2.25 – Скрипт процедуры GetNumber

В проекте необходимо было продемонстрировать работу базы данных с хранением более чем 100 000 строк, в приложении В представлен скрипт заполнения таблицы NUMBERS ста тысячами строк.

## 2.3 Функции

В базе данных представлена одна функция, которая была затронута при рассмотрении процедур осуществляющих регистрацию звонков.

Функция FindUser\_IdByNumber, скрипт приведён на рисунке 2.26, принимает один параметр – номер пользователя, производит поиск по таблицу NUMBERS на соответствие номера записи в таблице переданному параметру, возвращает скалярное, целочисленное значение соответствующие уникальному идентификатору пользователя, за котором закреплён совпавший номер.

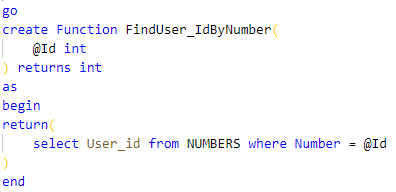


Рисунок 2.26 – Скрипт функции FindUser\_IdByNumber

## 2.4 **Индексы**

Для поддержания быстрого поиска записей в базе данных было разработано 3 некластеризованных индекса для таблиц, поиск в которых осуществляется не по уникальным идентификаторам записи.

Таблица NUMBERS хранит в себе 100 000 строк и в ней, помимо уникального идентификатора, осуществляется по номеру, для неё был создан индекс Numbers\_Nums по полю Number.

Для ускорения поиска в таблице CALLS был создан дополнительный индекс Call\_Index по полю Call\_Id, т.к. звонки постоянно совершаются, следовательно таблица расширяется, что приводит к необходимости ускорения поиска.

Таблица CUDlogAllTime содержит логи за всё время эксплуатации базы данных и по ней производится выборка по нескольким полям, для неё был создан индекс Log\_Index по полям OperationKey, TableName, Date.

## 2.5 **Триггеры**

В базе данных для 4 основных таблиц USERS, NUMBERS, TARIFFS, CALLS были созданы DML-триггеры для осуществления механизма логирования.

На примере триггера TariffTrigger таблицы TARIFFS мы рассмотрим их принцип действия, алгоритм триггеров отличается только в формировании полей BeforeValue и AfterValue, для каждой таблицы – свой шаблон, в остальном их реализация полностью идентична.

Скрипт приведён на рисунке 2.27

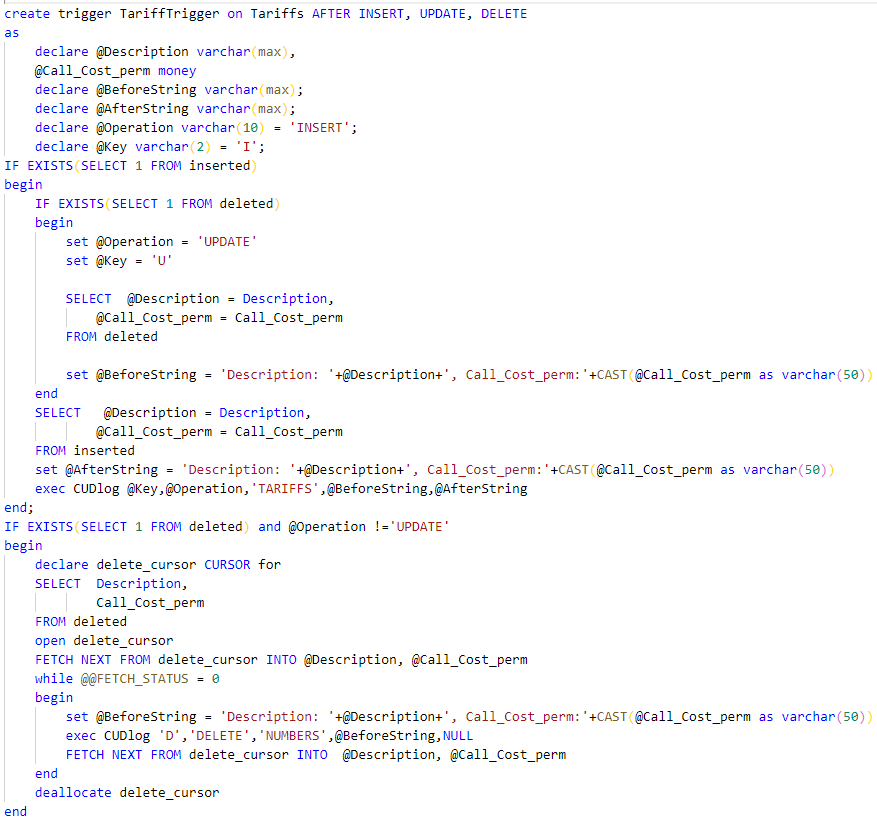


Рисунок 2.27 – Скрипт триггера TariffTrigger

Триггеры представляют специальный тип хранимой процедуры, которая вызывается автоматически при выполнении определенного действия над таблицей, для каждой из 4 основных таблиц был создан AFTER-триггер, срабатывающий после операций DML. Триггеры работают с двумя списками – inserted и deleted, для операции INSERT запись, вызвавшая срабатывание триггера находится в списке inserted, для DELETE – в deleted, для UPDATE в обоих, в inserted – новое значение записи, в deleted – старое. После срабатывания создают переменные описывающие строки таблицы, для которой он сработал и переменные описывающие строки таблицы ##CUDlogSession и CUDlogAllTime. Из таблиц inserted и deleted, в зависимости от операции, вызвавшей срабатывание триггера формируются значения, описывающие запись таблицы, на их основе формируется значение BeforeValue и AfterValue. Наличие или отсутствие данных в таблицах inserted и deleted характеризует операцию, вызвавшую срабатывание триггера. После того как все значения заданы, они передаются в процедуру CUDlog, описанную выше.

# 3 Обоснование технических приемов программирования

При разработке проекта в качестве демонстрационного приложения было разработано веб-приложение на клиент-серверной архитектуре, в качестве серверной основы была выбрана платформа Node.js и модифицирована пакетами, предоставляющими необходимый инструментарий для разработки. Общая схема взаимодействия выглядит следующим образом: клиент посылает HTTP запрос на сервер, тот обрабатывается и при необходимости делается запрос в базу данных, после чего ответ обрабатывается на Node-сервере, он посылает клиенту ответ в формате JSON или текстовом. Схема взаимодействия представлена на рисунке 3.1.

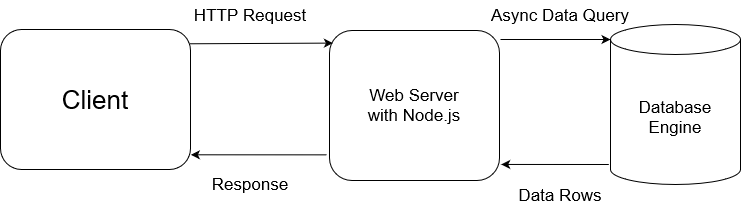


Рисунок 3.1 – Схема взаимодействия с Microsoft SQL Server

Для взаимодействия с базой Node необходим драйвер, в проект используется TDS драйвер msnodesqlv8, который позволяет упростить работу с СУБД. Для подключения на стороне Node-сервера сконфигурирована строка подключения, представленная на рисунке 3.2.

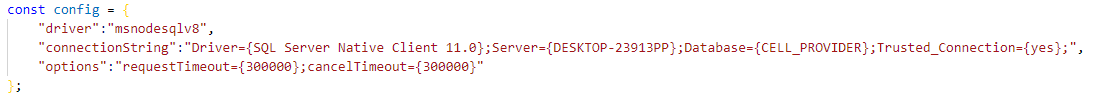
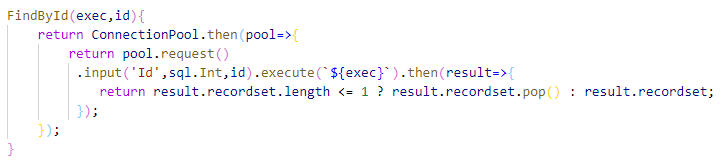


Рисунок 3.2 – Строка подключения к MS SQL Server

После этапа подключения Node-сервер ожидает запрос, который необходимо перенаправить в базу данных, если такой поступил, то он помещается в пул запросов, откуда он направляется в базу данных, пример на рисунке 3.3, получив результат, сервер обрабатывает данные и оборачивает их в HTTP ответ, который и будет отправлен клиентской стороне веб-приложения.

  
Рисунок 3.3 – Пример запроса Node-сервера к SQL-серверу

# 4 **Тестирование, проверка работоспособности и анализ данных**

Данная глава включает в себе тестирование взаимодействий демонстрационного веб-приложения с базой данных, функций доступных пользователю и администратору.

Функционал большинства приложений, не доступен, пока пользователь не авторизуется, демонстрационное приложение не стало исключением, пользователю не доступен функционал, заложенный в приложение, пока не будет пройден этап авторизации, который осуществляется на странице логина, продемонстрированной на рисунке 4.1.

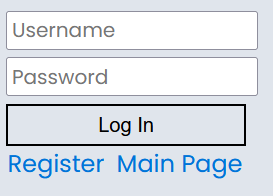


Рисунок 4.1 – Страница авторизации

Пользователь вводит логин и пароль и в случае успешной авторизации происходит перенаправление на главную страницу, на которой будут разблокированы ссылки на страницы доступные авторизованному пользователю и разблокирована возможность получить номер.

В случае, если пользователь не прошёл авторизацию, например неверно ввёл логин или пароль он будет заново перенаправлен на страницу авторизации, на поля ввода логина и пароля наложен следующие ограничения: минимальная длинна – 1, максимальная – 50.

После успешно пройденной авторизации пользователь попадает на главную страницу, которая содержит ссылки для перехода на другие страницы и информацию о тарифах, предоставляемых оператором сотовой связи – рисунок 4.2.

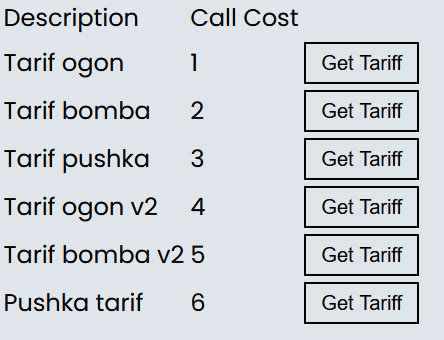


Рисунок 4.2 – Окно выбора тарифа

Одной из возможностей, доступных пользователю является получение номера с выбранным тарифом, для этого ему необходимо нажать на кнопку Get Tariff напротив понравившегося тарифа, система выдаст ему сообщение с полученным номером – рисунок 4.3, 4.4, номер также можно просмотреть в личном кабинете.

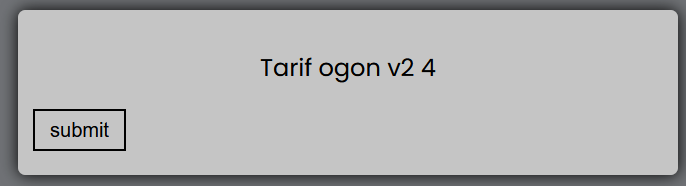


Рисунок 4.3 – Окно подтверждения подписки на тариф

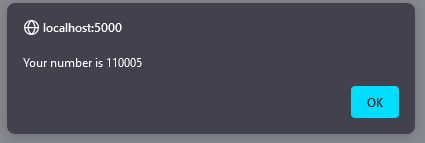


Рисунок 4.4 – Сообщение с номером, выданным пользователю

На страницу профиля – рисунок 4.5, пользователь может просматривать информацию, которую указывал при регистрации, просмотреть свои номера, историю звонков, а также пополнить баланс.

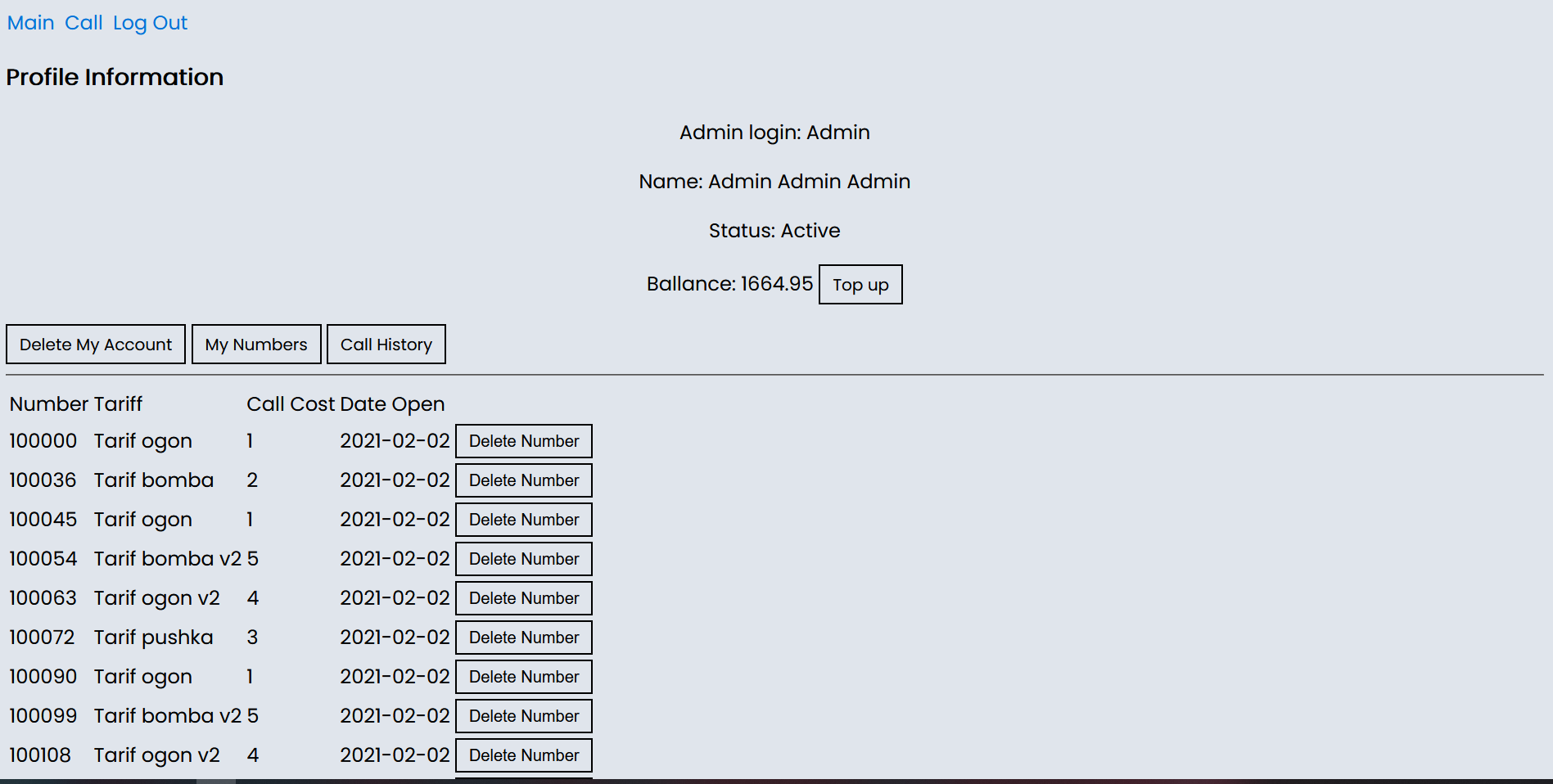


Рисунок 4.5 – Страница профиля с информацией о номерах пользователя

Остановимся подробнее на интерфейсе страницы профиля. Помимо переходов на главную страницу, страницу с которой можно совершить звонок и ссылку для выхода из личного кабинета, пользователь может производить манипуляции со своим аккаунтом – удалить свой аккаунт и пополнить баланс, а также просмотреть информацию о номерах и историю звонков. Пользователь не имеет возможности влиять на информацию о своих звонках, но может удалить номер, после чего тот пропадёт из списка номеров пользователь.

Пополнение баланса происходит по нажатию кнопки Top Up – рисунок 4.6, пользователь может ввести целые числа в диапазоне от 1 до 100 000.

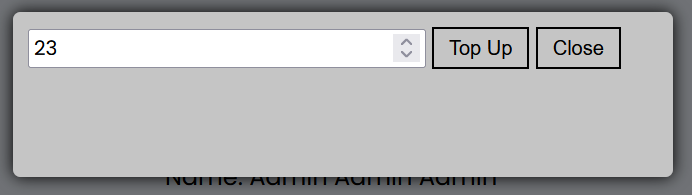


Рисунок 4.6 – Окно пополнения баланса

Если пользователь введёт некорректные данные, например буквы или число, не попадающее в диапазон от 1 до 100 000, то ему будет показано сообщение об ошибке с соответствующей рекомендацией, что отражена на рисунке 4.7.

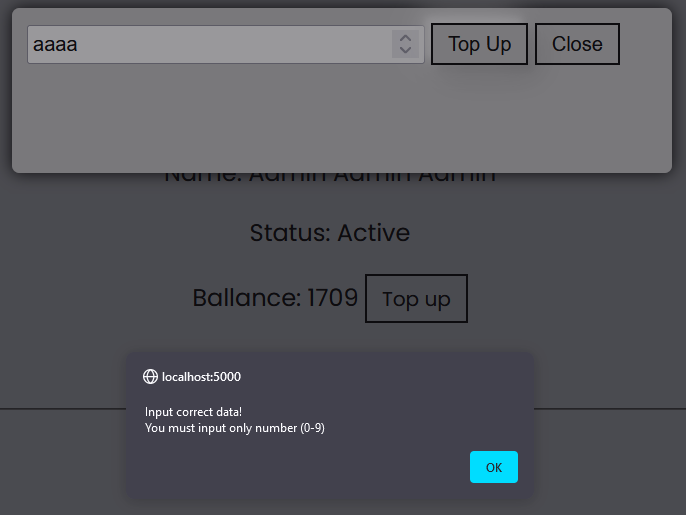


Рисунок 4.7 – Сообщение об ошибке при вводе некорректных данных

Со страницы профиля или по ссылке /Call пользователь может перейти на страницу, с которой можно совершить звонок, при загрузке страница содержит ссылку на главную страницу, выпадающий список с номерами пользователя, кнопку для начала вызова и два поля куда можно ввести номер, с которого будет совершён звонок и номер на который будет совершён звонок, на них наложены ограничения по длине – длина введённого номера не может превышать 6 символов. Страница для совершения звонков продемонстрирована на рисунке 4.8.

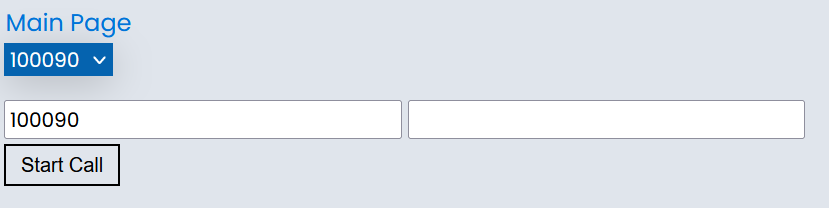


Рисунок 4.8 – Страница для совершения звонка

После нажатия на кнопку Start Call блокируются поля для ввода номеров, текст кнопки меняется на «End Call» и в окно статуса звонка выводится сообщение: «Your call in process», по завершению звонка поля для ввода номеров вновь доступны, текст кнопки возвращается, а пользователь получает сообщение о стоимости звонка и остатке на балансе – рисунок 4.9.

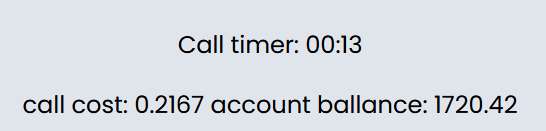


Рисунок 4.9 – Сообщения о статусе звонка после его совершения

В случае, если пользователь введёт некорректные данные в статусе звонка будет выведено соответствующее сообщение, а звонок не будет начат – рисунок 4.10.



Рисунок 4.10 – Сообщение об ошибке, неправильно введён номере получателя

Если пользователь авторизовался с привилегиями администратора, ему доступен переход на страницу администратора, на которой ему доступна информация о количестве объектов в базе данных и инструментарий администратора:

* Вывод информации из таблиц NUMBERS, CALLS, TARIFFS, USERS;
* Выдача прав администратора пользователям, смена их активности, удаление записи о пользователе;
* Импорт, экспорт таблицы CALLS из и в файл формата XML;
* Удаление номеров, закреплённых за пользователем;
* Добавление, редактирование и удаление информации о тарифах.

Окно администратора представлено на рисунке 4.11.

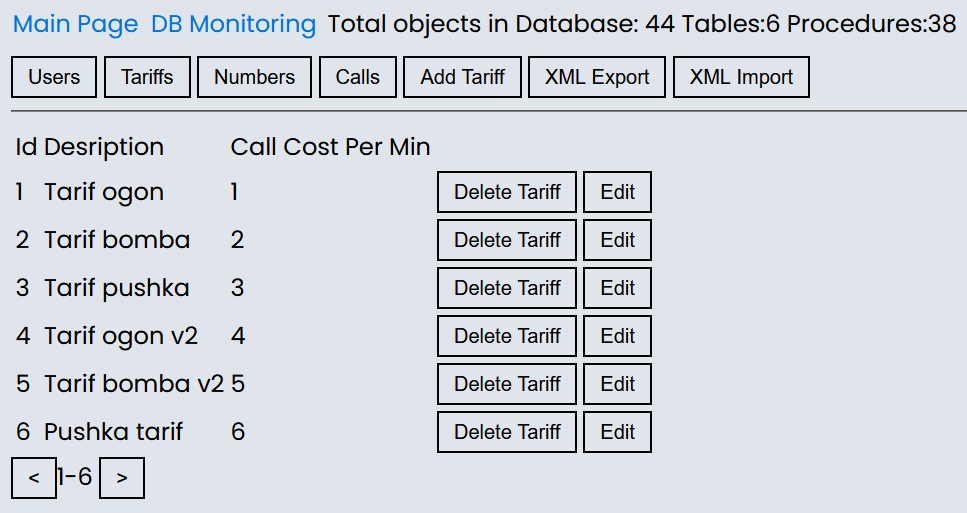


Рисунок 4.11 – Страница администратора

Со страницы администратора со стандартным инструментарием можно осуществить переход на страницу, которая отвечает за использование функционала реализованной технологии – средства мониторинга базы данных.

На странице мониторинга, помимо ссылок на возврат к обычной странице администратора и главной, администратору доступны функции сбора статистической информации и журнала логирования – рисунок 4.12.

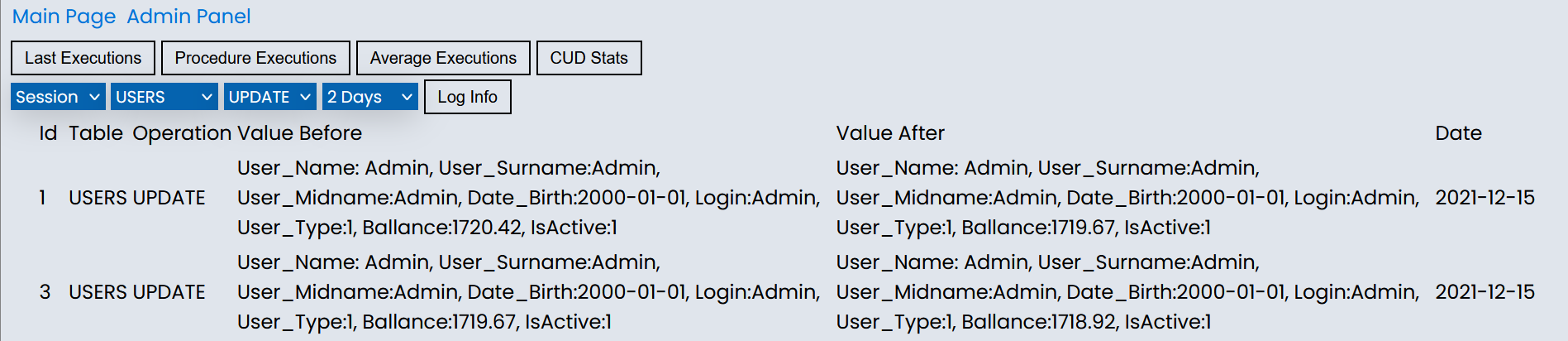


Рисунок 4.12 – Страница мониторинга с информацией из журнала логирования

Функционал журнала логирования позволяет администратору вывести информацию об изменениях проведённых в базе данных за определённый промежуток времени, над конкретной таблицей или определённой операцией.

Помимо функций журнала логирования администратор может получить следующую информацию:

* Последние вызовы в базе данных
* Количество вызовов каждой процедуры
* Процедуры с наибольшим временем выполнения
* Информацию о соотношении CUD-операций

Наиболее полезной информацией для администратора является информация о средней продолжительности выполнения процедур – рисунок 4.13, основываясь на ней он может принять решение об оптимизации запросов, которые дольше всего выполняются, запросы выполняющиеся более одной секунды будут отмечены красным цветом, что является ярким сигналом о чрезмерно длинном времени их выполнения.

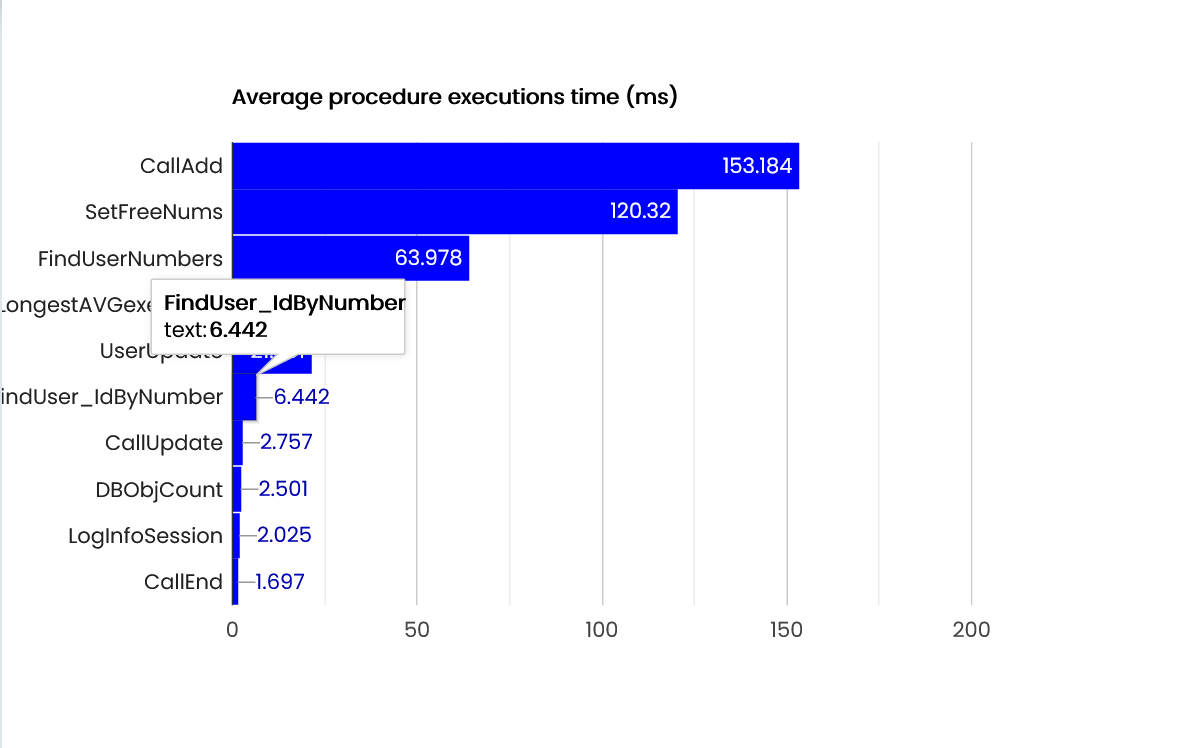


Рисунок 4.13 – Отображение среднего времени выполнения

Не менее полезной информацией может стать и соотношение CUD-операций, которое является отражением загруженности базы данных – рисунок 4.14.

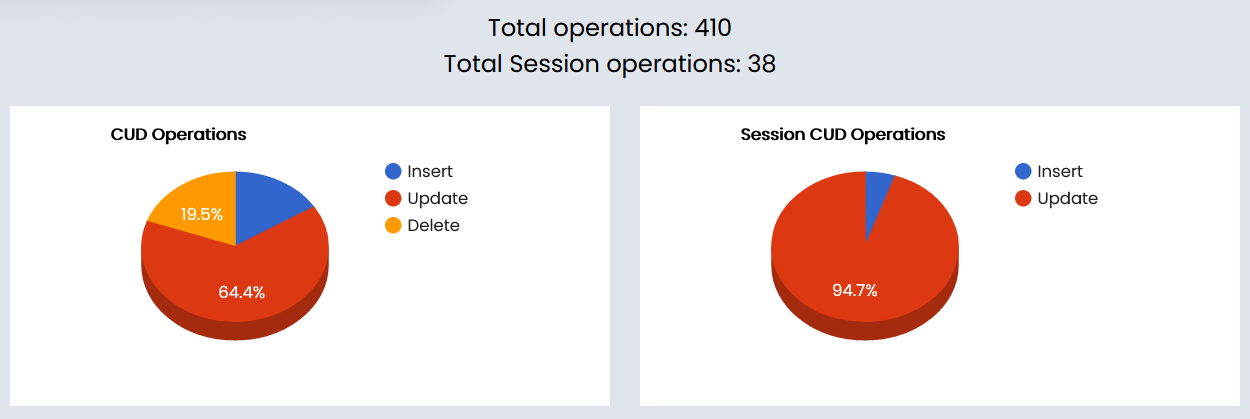


Рисунок 4.14 – Соотношение CUD-операций

# Заключение

Целю курсового проекта являлось проектирование с последующей программной реализации базы данных и демонстрационного приложения. В ходе архитектурного проектирования приложения были использованы не только давно заложенные постулаты проектирования, но и применены современные инструменты и тенденции. Проект включал в себя не только определение необходимого функционала, требуемого операторами сотовой связи, но внедрение понятного интерфейса.

Для разработки базы данных использовались программные возможности MS SQL Server и языков SQL, Transact/SQL, реализация которой описана в пояснительной записке.

Разработка демонстрационного веб-приложения «Оператор сети сотовой связи» велась на платформе Node.js с использованием Express.js, в разработанном веб-приложении использованы возможности, современных фреймворков, языка JavaScript и языка разметки HTML в связке в CSS.

В реализованной программе выполнены все цели, поставленные в начале проектирования курсового проекта, а именно:

– управление базой данных за счёт реализации функций администрирования, в частности просмотр информации о зарегистрированных абонентах;

– просмотр клиентом информации о доступных тарифах с последующим получением номера с привязкой к выбранному тарифу;

– просмотр информации о номерах, зарегистрированных на пользователя, включая привязанные тарифы, просмотр звонков, совершённых пользователем;

– просмотр пользователем информации о своём аккаунте и доступной информации о других пользователях.

Тестирование программы показало, что она работает корректно и выполняет все изложенные выше функции.

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

# Список используемой литературы

1. Node.js v11.15.0 Documentation [Электронный ресурс] / OpenJS Foundation. – Режим доступа: https://nodejs.org/docs/latest-v11.x/api. – Дата доступа: 13.11.2021.

2. Express 4.x API [Электронный ресурс] / StrongLoop, Inc. – Режим доступа: https://expressjs.com/en/4x/api.html. – Дата доступа: 13.11.2021.

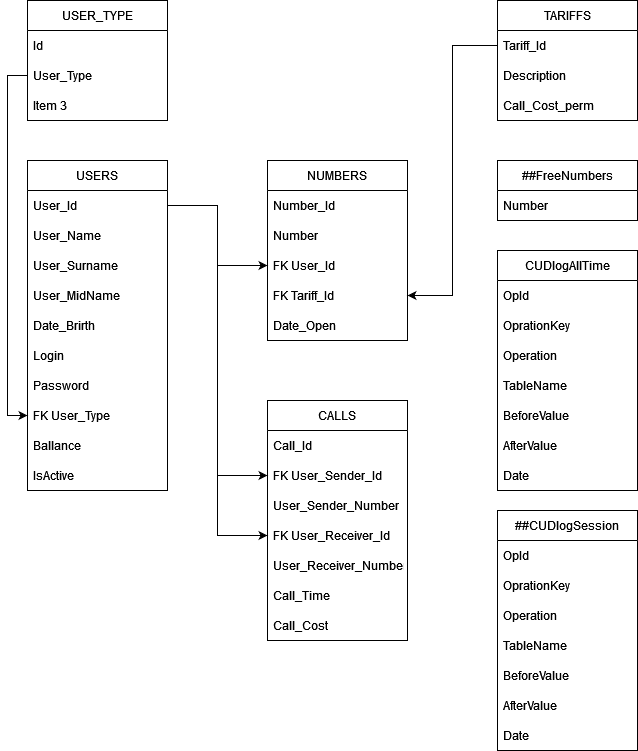
3. Mssql [Электронный ресурс] / OpenJS Foundation. – Режим доступа: https://www.npmjs.com/package/mssql. – Дата доступа: 13.11.2021.

4. Monitoring performance by using the Query Store [Электронный ресурс] / Microsoft. – Режим доступа: <https://docs.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store?view=sql-server-ver15>. – Дата доступа: 04.12.2021.

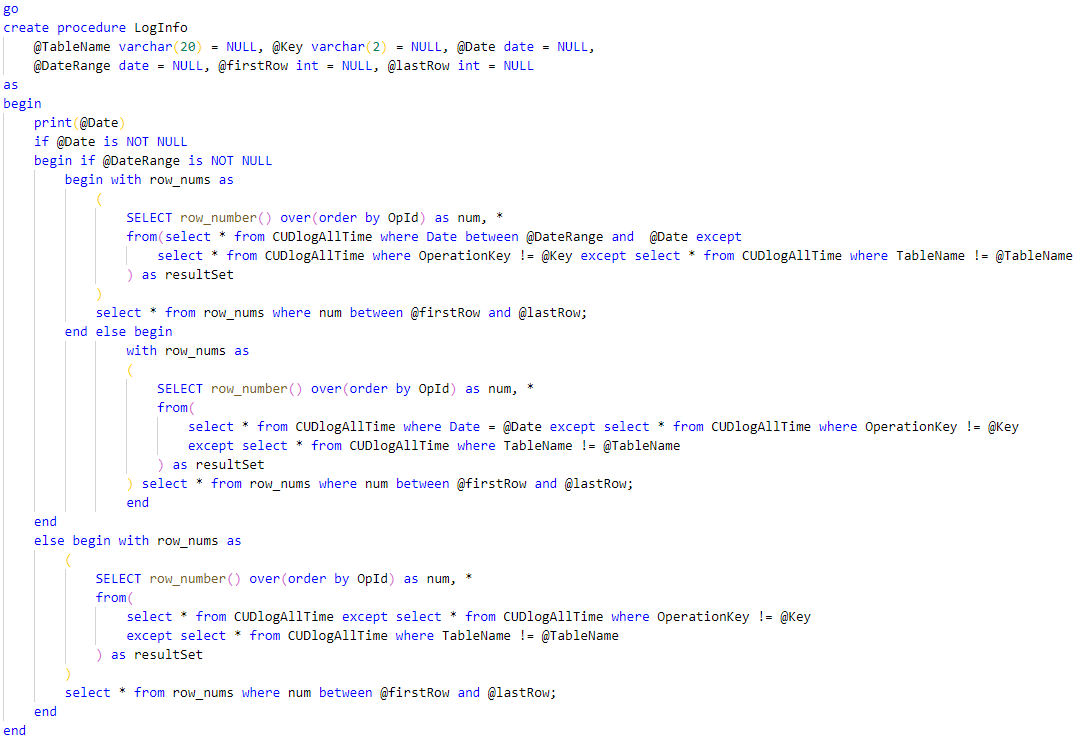
5. Query Store Catalog Views (Transact-SQL) [Электронный ресурс] / Microsoft. – Режим доступа: <https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/query-store-catalog-views-transact-sql?view=sql-server-ver15>. – Дата доступа: 09.12.2021.

6. Best practices with Query Store [Электронный ресурс] / Microsoft. – Режим доступа: https://docs.microsoft.com/en-us/sql/relational-databases/performance/best-practice-with-the-query-store?view=sql-server-ver15. – Дата доступа: 17.12.2021.

# Приложение А



# Приложение Б



# Приложение В

