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»
Специализация Программирование интернет-приложений

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:

Проектирование базы данных «Интернет провайдер»

Выполнил студент Олексюк Андрей Викторович
(Ф.И.О.)

Руководитель проекта асс. Потапенко К.С.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты асс. Патапенко К.С.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер асс. Патапенко К.С.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой _____

Минск 2017

Оглавление

Введение

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

Основные функции СУБД:

– определение структуры создаваемой базы данных, ее инициализация и проведение начальной загрузки;

– предоставление пользователям возможности манипулирования данными (выборка необходимых данных, выполнение вычислений, разработка интерфейса ввода/вывода, визуализация);

– обеспечение логической и физической независимости данных;

– защита логической целостности базы данных;

– защита физической целостности;

– управление полномочиями пользователей на доступ к базе данных;

– синхронизация работы нескольких пользователей;

– управление ресурсами среды хранения;

– поддержка деятельности системного персонала.

– Обычно современная СУБД содержит следующие компоненты:

– ядро, которое отвечает за управление данными во внешней и оперативной памяти и журнализацию;

– процессор языка базы данных, обеспечивающий оптимизацию запросов на извлечение и изменение данных и создание, как правило, машинно-независимого исполняемого внутреннего кода;

– подсистему поддержки времени исполнения, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД.

СУБД существует огромное множество: Oracle, MS SQL Server, Microsoft Access, MySql и так далее. В данной работе будет использовано решение от компании MS SQL Server. А так же для генерации данных был использован JavaScript.

1. Разработка модели базы данных

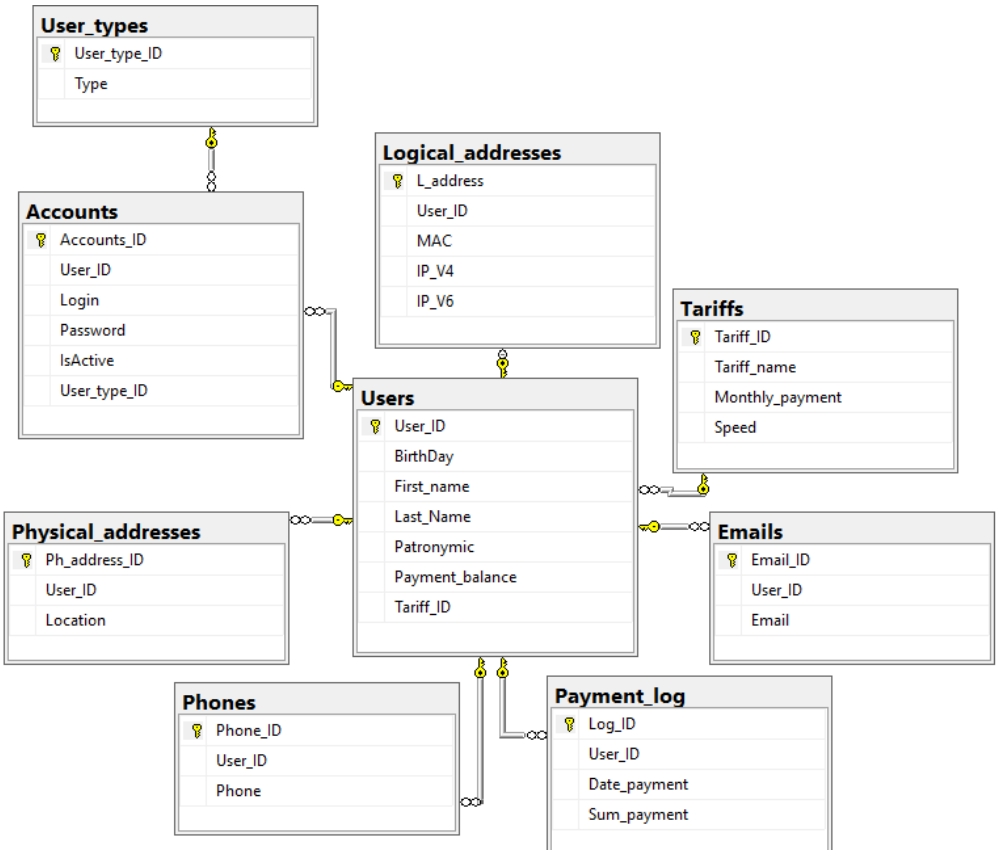


Рисунок 1.1 – Диаграмма базы данных

Для базы данных интернет провайдера были разработаны 9 таблиц. Диаграмма связей таблиц для необходимой базы данных представлена на рисунке 1.1.

Таблица Users, содержащая пользователей БД:

- User_ID – уникальный идентификатор;
- BirthDay – день рождения пользователя;
- First_name – имя пользователя;
- Last_name – фамилия пользователя;
- Patronymic – отчество пользователя;
- Payment_balance – текущий баланс;
- Tariff_ID – идентификатор тарифа.

Таблица Accounts, предназначенная для хранения аккаунтов пользователей:

- Accounts_ID – уникальный идентификатор;
- User_ID – идентификатор пользователя;
- Login;
- Password;
- IsActive – показывает доступен ли трафик пользователю в данный момент;
- User_type_ID – идентификатор типа пользователя.

Таблица User_types, предназначена для хранения типов пользователей и содержит:

- User_type_ID – уникальный идентификатор:
- Type – тип пользователя.

Таблица Logical_addresses хранит сетевые адреса пользователей и содержит:

- L_address – уникальный идентификатор;
- User_ID – идентификатор пользователя;
- MAC;
- IP_V4;
- IP_V6.

Таблица Physical_addresses содержит физический адрес пользователей и содержит поля:

- Ph_address_ID - уникальный идентификатор;
- User_ID – идентификатор пользователя;
- Location – поле типа geography в котором хранится точка в эллиптических координатах местонахождения пользователя.

Таблица Phones хранит номера телефонов пользователей и содержит:

- Phone_ID;
- User_ID;
- Phone.

Таблица Payment_log предназначена для хранения истории платежей каждого пользователя:

- Log_ID – уникальный идентификатор;
- User_ID – идентификатор пользователя;
- Date_payment – время платежа;
- Sum_payment – сумма платежа.

Таблица Emails предназначена для хранения почты пользователей:

- Email_ID – уникальный идентификатор;
- User_ID – идентификатор пользователя;
- Email – почта пользователя.

Таблица Tariffs предназначена для хранения тарифов и содержит:

- Tariff_ID – уникальный идентификатор;
- Tariff_name – название тарифа;
- Monthly_payment – месячный платеж;
- Speed – предоставляемая скорость.

Скрипты для создания всех таблиц базы данных представлены в Приложении А.

2. Разработка необходимых объектов

2.1. Таблицы

Таблицы являются основой любой базы данных, именно в них хранится вся информация. При проектировании базы данных было создано 9 таблиц, которые подробно описаны ранее в разделе 1, а SQL-скрипты для их создания находятся в Приложении А.

2.2. Процедуры

Использование хранимых процедур позволяет ограничить либо вообще исключить непосредственный доступ пользователей к таблицам базы данных, оставив пользователям только разрешения на выполнение хранимых процедур, обеспечивающих косвенный и строго регламентированный доступ к данным. Листинги некоторых хранимых процедур представлены в приложении Б.

Всего было разработано 26 процедур:

- AddAdmin – процедура для добавления администратора;
- AddUser – процедура для добавления администратора;
- CheckPay – процедура для проведения ежедневной оплаты определенного пользователя;
- CheckUserName – процедура для проверки ФИО пользователя на уникальность;
- DailyPayment – процедура для проведения ежедневной оплаты у всех пользователей;
- DistanceFromUsers – процедура для получения расстояния между двумя пользователями;
- NearestUsers – нахождение семи ближайших пользователей
- GetAllUsers – процедура для получения всех пользователей, так же были разработаны процедуры выборки пользователей по логину (GetUserByLogin), почте (GetUserByEmail), MAC (GetUserByMAC), ФИО (GetUserByName);
- GetPaymentLog – процедура для получения истории платежей пользователя;
- GetStatusAccount – процедура для проверки статуса пользователя;
- InsertTariffs – процедура для добавления тарифа;
- InsertUserType –процедура для добавления типа пользователя;
- TariffsDelete – процедура для удаления тарифа;
- TariffsToXML – процедура для экспорта тарифов в файл *.xml;
- TariffsFromXML – процедура для импорта тарифов из файла *.xml;
- UpdateUserTariff – процедура для изменения тарифа пользователя;
- UpdateAccount – процедура для изменения пароля пользователя;

- UpdateLocation – процедура для изменения геоданных пользователя;
- UpdatePhpne – процедура для изменения номера телефона пользователя;
- UpdateTariff – процедура для изменения тарифов;
- UserPay – процедура для проведения оплаты пользователя

Все скрипты хранимых процедур приложены в отдельных файлах в корне директории прилагаемого диска.

3. Описание процедур импорта и экспорта данных

3.1. Процедура импорта данных из XML-файла

Для преобразования XML-данных в строки таблицы предназначена функция OPENXML. Она принимает три входных параметра: дескриптор, сформированный системной хранимой процедурой с именем SP_XML_PREPAREDOCUMENT; выражение XPATH и целое положительное число, определяющее режим работы функции.

Процедура SP_XML_PREPAREDOCUMENT должна быть выполнена до SELECT-запроса, применяющего OPENXML. Процедура принимает в качестве входного параметра XML-документ (в формате строки) и возвращает дескриптор, который впоследствии применяется функцией OPENXML.

Выражение XPATH, принимаемое функцией OPENXML в качестве второго параметра, предназначено для выбора требуемых данных из исходного (введенного процедурой SP_XML_PREPAREDOCUMENT) XML-документа. Последний, третий, параметр функции OPENXML указывает на тип преобразования (режим). Пример реализации процедуры представлен на рисунке 3.1.1.

```
IF OBJECT_ID('dbo.TariffsFromXML') IS NOT NULL
BEGIN
    DROP PROC dbo.TariffsFromXML
END
GO
CREATE PROC dbo.TariffsFromXML
AS
    DECLARE @x xml
    SELECT @x = P
    FROM OPENROWSET (BULK 'D:\WORK\Oracle\курсовой\InternetProvider\XML\Tariffs.xml', SINGLE_BLOB) AS Tariffs(P)
    DECLARE @hdoc int
    EXEC sp_xml_preparedocument @hdoc OUTPUT, @x
    SELECT *
    FROM OPENXML(@hdoc, '/Tariffs/Tariff', 2)
    WITH(
        Tariff_ID int,
        Tariff_name nvarchar(50),
        Monthly_payment real,
        Speed smallint
        )
```

Рисунок 3.1.1. – Процедура импорта select запроса в XML

3.2. Процедура экспорта данных в XML-файл

Для преобразования результата SELECT-запроса в формат XML в операторе SELECT применяется секция FOR XML. При этом можно использовать один из четырех режимов: RAW, AUTO, PATH и EXPLICIT.

По умолчанию в режиме RAW в результате SELECT-запроса создается XML-фрагмент, состоящий из последовательности элементов с именем **row**. Каждый элемент **row** соответствует строке результирующего набора, имена его атрибутов совпадают с именами столбцов результирующего набора (из списка SELECT), а значения атрибутов равны их значениям.

Результат, полученный в режиме AUTO для простых SELECT-запросов, похож на результат, полученный в режиме RAW. Основное отличие – в качестве имени элемента, соответствующего строке исходной таблицы, используется ее имя

Режим PATH позволяет разработчику наиболее полным образом управлять процессом формирования XML-структуры. Каждый столбец конфигурируется независимо с помощью заданного в формате XPATH имени псевдонима этого столбца.

Так же можно построить корневой элемент с помощью ROOT, указав в его параметре имя корневого элемента.

Реализация процедуры экспорта данных в XML представлена на рисунке 3.2.1.

```
IF OBJECT_ID('dbo.TariffsToXML') IS NOT NULL
BEGIN
    DROP PROC dbo.TariffsToXML
END
GO
CREATE PROC dbo.TariffsToXML
AS
    SELECT
        Tariff_ID,
        Tariff_name,
        Monthly_payment,
        Speed
    FROM Tariffs
    FOR XML PATH('Tariff'), ROOT('Tariffs')
```

Рисунок 3.2.1. – Скрипт разрешения выполнения команд в cmd

4. Технология Spatial Data

Spatial data представляет собой типы данных для обработки и хранения данных о физическом расположении объектов и фигур.

SQL Server поддерживает два пространственных типа данных: *geometry* и *geography*.

Geometry – пространственный тип данных хранящий данные в евклидовой системе координат.

Geography – прострранственный тип данных, предназначенный для хранения данных в сферических координатах

Как и *geometry* так и *geography* представляют возможности для хранения следующих фигур:

- Point – в geometry представляет собой единое место, где X представляет координату x создаваемого экземпляра Point, а Y - координату Y создаваемого экземпляра Point. SRID представляет идентификатор пространственной ссылки экземпляра geometry, который необходимо вернуть. В geography представляет единичное расположение, где Lat представляет широту, а Long - долготу. Значения широты и долготы измеряются в градусах. Значения широты всегда находятся в интервале [-90, 90]. Все значения, находящиеся вне этого диапазона, вызывают исключение. Значения долготы всегда находятся в интервале [-180, 180];
- LineString – является одномерным объектом, представляющим последовательность точек и соединяющих их линейных сегментов;
- CircularString – это коллекция, состоящая из нуля или большего количества непрерывных круговых сегментов дуги. Сегмент дуги - это сегмент кривой, определяемый тремя точками на двумерной плоскости; первая точка не может совпадать с третьей. Если все три точки сегмента дуги лежат на одной прямой, сегмент дуги считается линейным сегментом;
- CompoundCurve – это набор из нуля или большего количества непрерывных экземпляров CircularString или LineString геометрического или географического типов. ;
- Polygon – представляет собой двухмерную поверхность, хранимую в виде последовательности точек, определяющих внешнее ограничивающее кольцо, и внутренних колец (последние могут отсутствовать);
- CurvePolygon – является топологически закрытой областью, определенной внешним ограничивающим кольцом, а также нулем или более внутренних колец;
- MultiPoint – представляет собой коллекцию точек;
- MultiLineString – представляет собой коллекцию экземпляров geometry или geographyLineString;

– MultiPolygon – представляет собой коллекцию экземпляров Polygon ;
– GeometryCollection – представляет собой коллекцию экземпляров geometry или geography.

Так же для этих типов данных существует собственный пространственный индекс позволяющий более эффективно выполнять определенные операции со столбцами, содержащими данные типа geometry или geography

В данной курсовой работе используется тип данных *geography* для хранения точки в сферических координата, указывающая на физический адрес пользователя. А так же были разработаны две процедуры использующие метод STDistance для нахождения расстояния между двумя точками:

- DistanceFromUsers – применяется для нахождения расстояния между двумя пользователями;

- NearestUsers – применяется для нахождения семи ближайших пользователей.

Листинги этих процедур представлены в приложении Б вместе с остальными.

5. Резервное копирование и восстановление

Резервное копирование базы данных будет осуществляться в папку BackupInternetProvider на диске D:\ и имя файла будет состоять из даты создания бэкапа.

Листинг исходного кода резервного копирования:

```
DECLARE @pathName NVARCHAR(512)
SET @pathName = 'D:\ BackupInternetProvider\db_backup_' + Convert(varchar(8), GETDATE(), 112) + '.bak'
BACKUP DATABASE [InternetProvider] TO DISK = @pathName WITH NOFORMAT, NOINIT, NAME = N'db_backup', SKIP, NOREWIND, NOUNLOAD, STATS = 10
```

Далее создаем файл расширения .bat, который и будет запускать скрипт резервного копирования.

Листинг этого .bat файла:

```
sqlcmd -S DESKTOP-L5ROMAE -E -i backup.sql
```

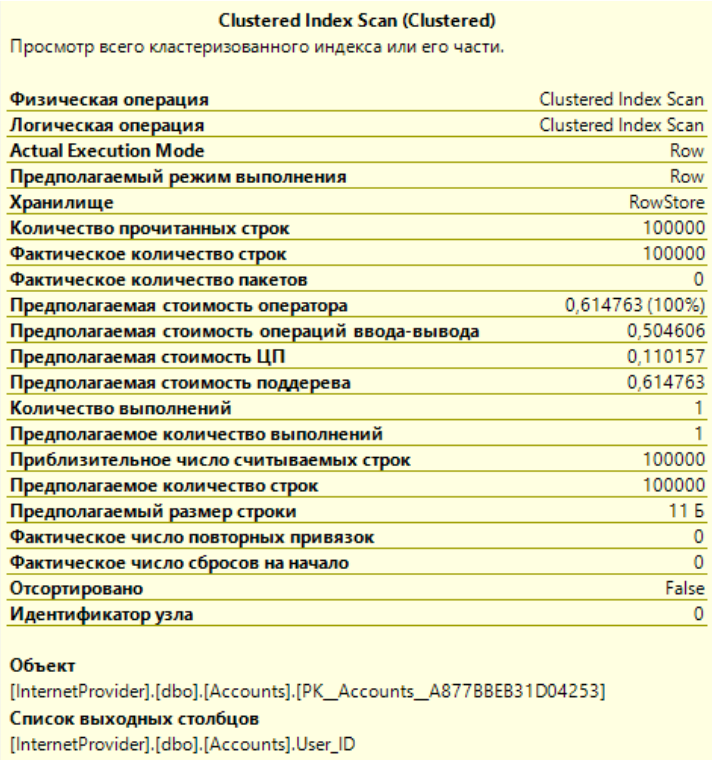
Восстановление базы данных можно осуществить с помощью выполнения .bat файла.

6. Тестирование

6.1 Тестирование производительности базы данных

Для тестирования производительности была взята за основу таблица Accounts так как в основном процедуры используют имя пользователя для получения его ID.

Изначально таблица таблица Accounts имеет 100003 записи. Планы запросов к таблице Accounts представлены на рисунках 6.1.1.



Clustered Index Scan (Clustered)

Просмотр всего кластеризованного индекса или его части.

Физическая операция	Clustered Index Scan
Логическая операция	Clustered Index Scan
Actual Execution Mode	Row
Предполагаемый режим выполнения	Row
Хранилище	RowStore
Количество прочитанных строк	100000
Фактическое количество строк	100000
Фактическое количество пакетов	0
Предполагаемая стоимость оператора	0,614763 (100%)
Предполагаемая стоимость операций ввода-вывода	0,504606
Предполагаемая стоимость ЦП	0,110157
Предполагаемая стоимость поддерева	0,614763
Количество выполнений	1
Предполагаемое количество выполнений	1
Приблизительное число считываемых строк	100000
Предполагаемое количество строк	100000
Предполагаемый размер строки	11 Б
Фактическое число повторных привязок	0
Фактическое число сбросов на начало	0
Отсортировано	False
Идентификатор узла	0

Объект
[InternetProvider].[dbo].[Accounts].[PK__Accounts__A877BBEB31D04253]
Список выходных столбцов
[InternetProvider].[dbo].[Accounts].User_ID

Рисунок 6.1.1 – Оценка запроса к таблице Accounts без некластеризованного индекса

После проведения первоначальной оценки был построен некластеризованный индекс к таблице Accounts по столбцу User_ID и проведена оценка такого же SELECT-запроса к таблице Accounts. Результаты, полученные во время оценки, представлены на рисунке 6.1.2.

Просмотр индекса (NonClustered)

Просмотр всего некластеризованного индекса или его части.

Физическая операция	Просмотр индекса
Логическая операция	Index Scan
Actual Execution Mode	Row
Предполагаемый режим выполнения	Row
Хранилище	RowStore
Количество прочитанных строк	100000
Фактическое количество строк	100000
Фактическое количество пакетов	0
Предполагаемая стоимость операций ввода-вывода	0,132755
Предполагаемая стоимость оператора	0,242912 (100%)
Предполагаемая стоимость ЦП	0,110157
Предполагаемая стоимость поддерева	0,242912
Количество выполнений	1
Предполагаемое количество выполнений	1
Предполагаемое количество строк	100000
Приблизительное число считываемых строк	100000
Предполагаемый размер строки	11 Б
Фактическое число повторных привязок	0
Фактическое число сбросов на начало	0
Отсортировано	False
Идентификатор узла	0

Объект
[InternetProvider].[dbo].[Accounts].[NonClusteredIndex-20171218-124436]
Список выходных столбцов
[InternetProvider].[dbo].[Accounts].User_ID

Рисунок 6.1.2 – Оценка запроса к таблице с построенным некластеризованным индексом

По результатам проведённых оценок до и после построения некластеризованного индекса, можно сделать вывод, что после создания индекса получили, что стоимость операций ввода-вывода и поддерева уменьшились почти в два раза

Таким образом, постройка индекса к таблице была более чем оправдана, так как мы получили прирост производительности в 2 раза, при обращении к столбцу User_ID.

6.2. Тестирование процедур.

В процедурах были предусмотрены исключительные ситуации

```
DECLARE @res int
exec @res = dbo.AddUser 'Andrey',
                        'Oleksyuk',
                        'Victorovich',
                        '22.10.1997',
                        'tariff1',
                        '(29) 111-2233',
                        'andreyoleksyuk@gmail.com',
                        'AA:BB:CC:DD:EE:FF',
                        '127.0.0.1',
                        'FFFF:FFFF:FFFF:FFFF:FFFF:F
                        '10041',
                        '12345',
                        53.932864,
                        27.428590
print @res
```

Сообщения | План выполнения

```
50001
Пользователь с таким именем уже существует.
0
```

Рисунок 6.2.1 - Попытка добавить пользователя уже существующего в БД

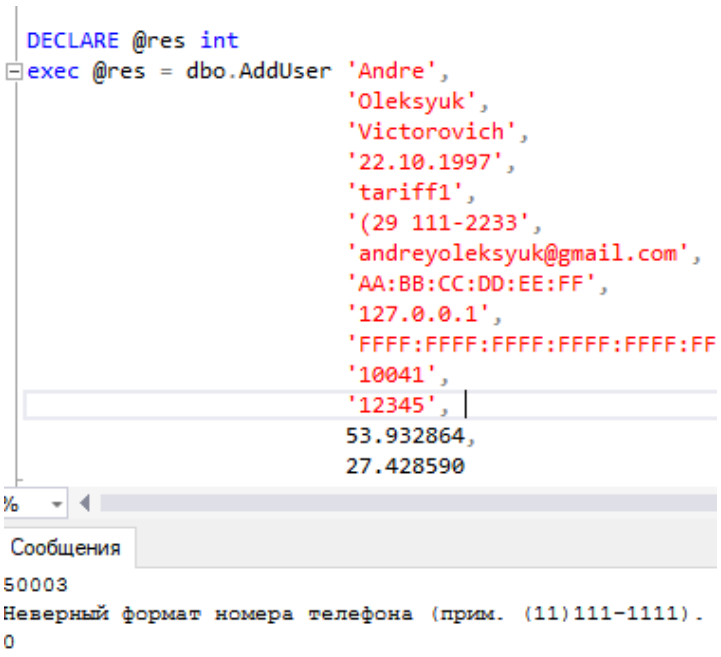


Рисунок 6.2.2 - Неверный формат номера телефона

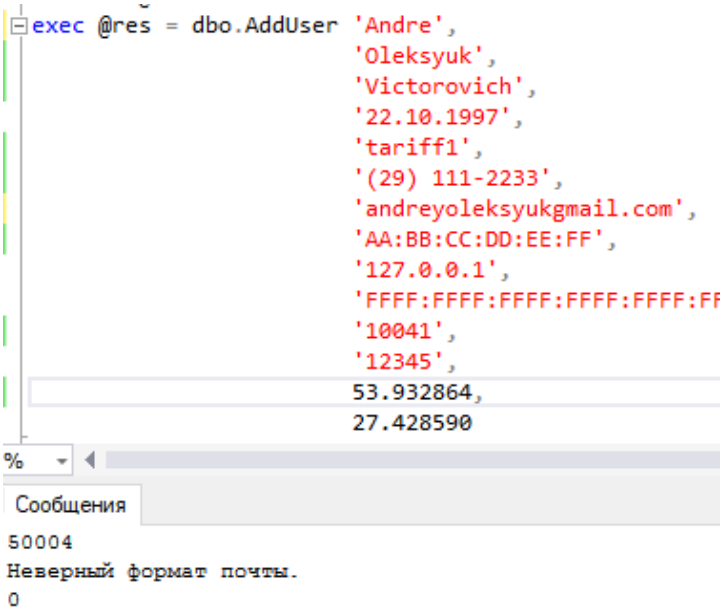


Рисунок 6.2.3 – Неверный формат почты

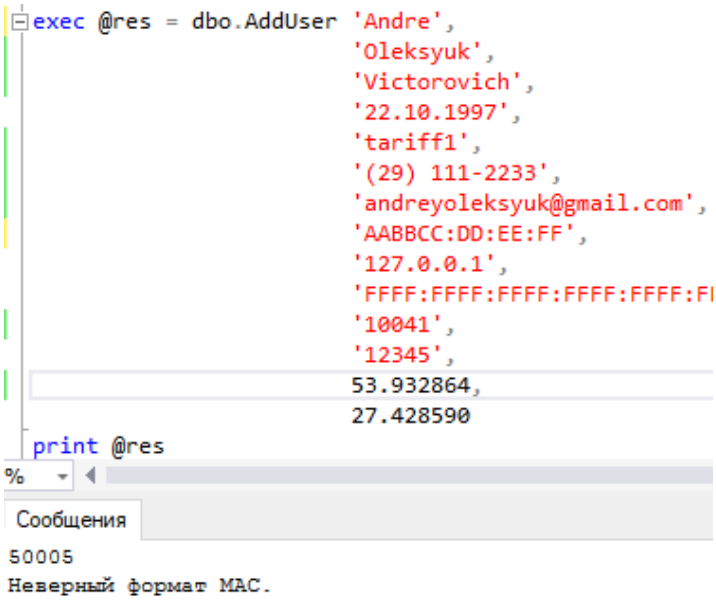


Рисунок 6.2.4 – Задан неверный формат MAC адреса

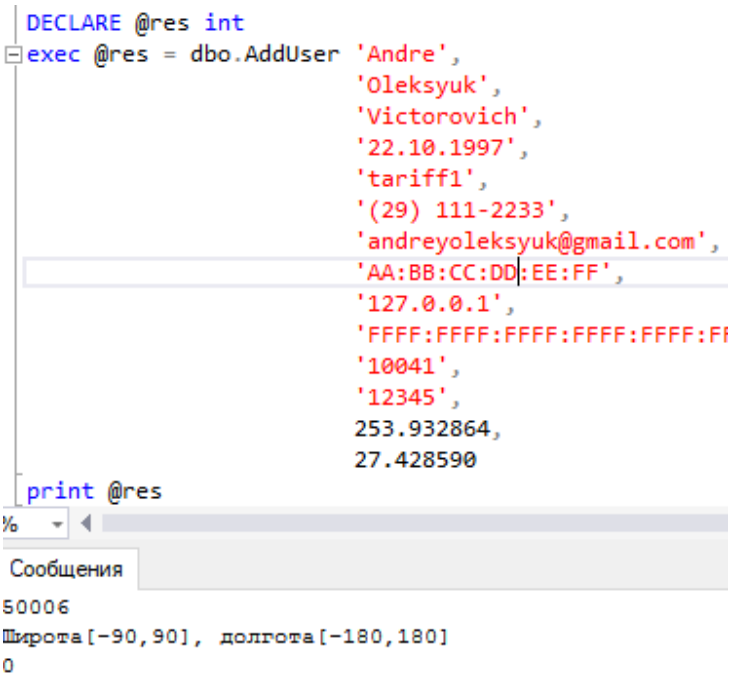


Рисунок 6.2.5 – Задана неверно широта

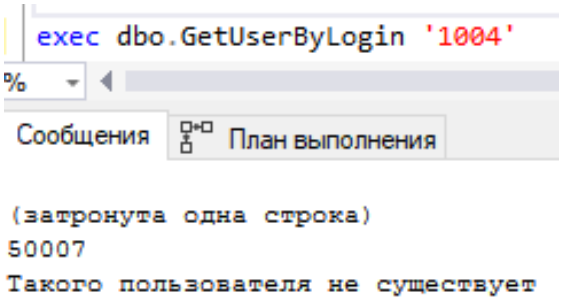


Рисунок 6.2.6 – попытка найти несуществующего пользователя

Заключение

В данном курсовом проекте была разработана база данных для интернет провайдера. Также были разработаны процедуры для взаимодействия с базой данных. Помимо этого, было настроено резервное копирование бд на случай сбоя работы, а также задействована технология SpatialData.
В соответствии с полученным результатом, можно сказать, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

Список использованной литературы

1. developers.google [Электронный ресурс] – Режим доступа: https://developers.google.com/maps/documentation/javascript/geocoding– Дата доступа: 30.11.2017.
2. В.В. Смелов, Л. С. Мороз , Microsoft SQL Server 2008: основы Transact-SQL / В.В. Смелов, Л. С. Мороз – Минск: БГТУ, 2014. – 129 с.
3. msdn.microsoft [Электронный ресурс] – Режим доступа: https://msdn.microsoft.com

Приложение А

```
IF OBJECT_ID('dbo.Tariffs') IS NOT NULL
BEGIN
    DROP TABLE dbo.Tariffs
END
GO
CREATE TABLE dbo.Tariffs(
       Tariff_ID int IDENTITY(1,1) PRIMARY KEY,
       Tariff_name nvarchar(50) NOT NULL UNIQUE,
       Monthly_payment real NOT NULL,
       Speed smallint NOT NULL,
);

IF OBJECT_ID('dbo.Users') IS NOT NULL
BEGIN
    DROP TABLE dbo.Users
END
GO
CREATE TABLE dbo.Users(
       User_ID int IDENTITY(1,1) PRIMARY KEY,
       BirthDay date NOT NULL,
       First_name nvarchar(50) NOT NULL,
       Last_Name nvarchar(50) NOT NULL,
       Patronymic nvarchar(50) NOT NULL,
       Payment_balance real,
       Tariff_ID int,
       FOREIGN KEY (Tariff_ID) REFERENCES dbo.Tariffs(Tariff_ID)
);

IF OBJECT_ID('dbo.Phones') IS NOT NULL
BEGIN
    DROP TABLE dbo.Phones
END
GO
CREATE TABLE dbo.Phones(
       Phone_ID int IDENTITY(1,1) PRIMARY KEY,
       User_ID int NOT NULL,
       Phone nchar(15) NOT NULL,
       FOREIGN KEY (User_ID) REFERENCES dbo.Users(User_ID)
);

IF OBJECT_ID('dbo.Emails') IS NOT NULL
BEGIN
    DROP TABLE dbo.Emails
END
GO
CREATE TABLE dbo.Emails(
       Email_ID int IDENTITY(1,1) PRIMARY KEY,
       User_ID int NOT NULL,
       Email varchar(50) NOT NULL UNIQUE,
       FOREIGN KEY (User_ID) REFERENCES dbo.Users(User_ID)
);

IF OBJECT_ID('dbo.Physical_addresses') IS NOT NULL
BEGIN
    DROP TABLE dbo.Physical_addresses
END
GO
CREATE TABLE dbo.Physical_addresses(
       Ph_address_ID int IDENTITY(1,1) PRIMARY KEY,
       User_ID int NOT NULL,
       Location geography,
       FOREIGN KEY (User_ID) REFERENCES dbo.Users(User_ID)
```

```
);

IF OBJECT_ID('dbo.Logical_addresses') IS NOT NULL
BEGIN
    DROP TABLE dbo.Logical_addresses
END
GO
CREATE TABLE dbo.Logical_addresses(
       L_address int IDENTITY(1,1) PRIMARY KEY,
       User_ID int NOT NULL,
       MAC nchar(17) NOT NULL,
       IP_V4 nchar(15) NOT NULL,
       IP_V6 nchar(39) NOT NULL,
       FOREIGN KEY (User_ID) REFERENCES dbo.Users(User_ID)
);

IF OBJECT_ID('dbo.Payment_log') IS NOT NULL
BEGIN
    DROP TABLE dbo.Payment_log
END
GO
CREATE TABLE dbo.Payment_log(
       Log_ID int IDENTITY(1,1) PRIMARY KEY,
       User_ID int NOT NULL,
       Date_payment datetime NOT NULL,
       Sum_payment real NOT NULL,
       FOREIGN KEY (User_ID) REFERENCES dbo.Users(User_ID)
);

IF OBJECT_ID('dbo.User_types') IS NOT NULL
BEGIN
    DROP TABLE dbo.User_types
END
GO
CREATE TABLE dbo.User_types(
       User_type_ID tinyint IDENTITY(1,1) PRIMARY KEY,
       Type nvarchar(50) NOT NULL
);

IF OBJECT_ID('dbo.Accounts') IS NOT NULL
BEGIN
    DROP TABLE dbo.Accounts
END
GO
CREATE TABLE dbo.Accounts(
       Accounts_ID int IDENTITY(1,1) PRIMARY KEY,
       User_ID int NOT NULL,
       Login nvarchar(50) NOT NULL UNIQUE,
       Password nvarchar(50) NOT NULL,
       IsActive bit NOT NULL,
       User_type_ID tinyint NOT NULL,
       FOREIGN KEY (User_ID) REFERENCES dbo.Users(User_ID),
       FOREIGN KEY (User_type_ID) REFERENCES dbo.User_types(User_type_ID)
);
```

Приложение Б

```
IF OBJECT_ID('dbo.AddUser') IS NOT NULL
BEGIN
    DROP PROC dbo.AddUser
END
GO
CREATE PROC dbo.AddUser    @First_name nvarchar(50),
                                        @Last_name nvarchar(50),
                                        @Patronymic nvarchar(50),
                                        @BirthDay date,
                                        @Tariff_name nvarchar(50),
                                        @Phone nchar(15),
                                        @Email nvarchar(50),
                                        @MAC nchar(17),
                                        @IP_V4 nchar(15),
                                        @IP_V6 nchar(39),
                                        @Login nvarchar(50),
                                        @Password nvarchar(50),
                                        @Latitude float,
                                        @Longitude float
AS
       BEGIN TRY
              SET NOCOUNT ON

              SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
              BEGIN TRAN

              IF(RTRIM(@Phone) NOT LIKE '(%) [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]')
                            THROW 50003,
                     'Неверный формат номера телефона (прим. (11)111-1111).', 1;
              IF(@Email NOT LIKE '%@%.%')
                     THROW 50004, 'Неверный формат почты.', 1;
              IF(@MAC
       NOT LIKE '[0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F]:[0-F][0-F]')
                     THROW 50005, 'Неверный формат MAC.', 1;
              IF(@Latitude NOT BETWEEN -90 AND 90)
                     THROW 50006, 'Широта[-90,90], долгота[-180,180]', 1;
              IF(@Longitude NOT BETWEEN -180 AND 180)
                     THROW 50006, 'Широта[-90,90], долгота[-180,180]', 1;

              DECLARE @count int;
              DECLARE @User_ID int;
              exec @count = dbo.CheckUserName @First_name, @Last_name, @Patronymic;
              IF(@count > 0)
                     THROW 50001, 'Пользователь с таким именем уже существует.', 1;

              DECLARE @Tariff_ID int;
              SELECT @Tariff_ID = dbo.Tariffs.Tariff_ID
                     FROM dbo.Tariffs
                     WHERE dbo.Tariffs.Tariff_name = @Tariff_name;
              IF(@Tariff_ID is null)
                     THROW 50002, 'Не найден такой тариф.', 1;

              INSERT INTO dbo.Users (BirthDay, First_name, Last_Name,
                                          Patronymic, Tariff_ID, Payment_balance)
                     SELECT @BirthDay, @First_name, @Last_Name, @Patronymic, @Tariff_ID,
0;

              SET @User_ID = IDENT_CURRENT('Users');

              INSERT INTO dbo.Phones(User_ID, Phone)
                     SELECT @User_ID, @Phone;
```

```
		INSERT INTO dbo.Emails(User_ID, Email)
			SELECT @User_ID, @Email;

		INSERT INTO dbo.Logical_addresses(User_ID, MAC, IP_V4, IP_V6)
			SELECT @User_ID, @MAC, @IP_V4, @IP_V6;

		DECLARE @User_type_ID tinyint
		SELECT @User_type_ID = User_type_ID
			FROM dbo.User_types
			WHERE dbo.User_types.Type = 'user'
		IF(@User_type_ID is NULL)
			THROW 50008, 'Перед добавлением пользователя добавьте тип пользова-
теля user', 1;

		INSERT INTO dbo.Accounts(User_ID, Login, Password, IsActive, User_type_ID)
			SELECT @User_ID, @Login, @Password, 0, @User_type_ID;

		DECLARE @g geography;
		SET @g = geography::Point(@Latitude, @Longitude, 4326);
		INSERT INTO dbo.Physical_addresses(User_ID, Location)
			SELECT @User_ID, @g;

		COMMIT TRAN
		RETURN @User_ID;

	END TRY
	BEGIN CATCH
		print error_number()
		print error_message()
		rollback tran
		return 0
	END CATCH

IF OBJECT_ID('dbo.CheckUserName') IS NOT NULL
BEGIN
    DROP PROC dbo.CheckUserName
END
GO
CREATE PROC dbo.CheckUserName	@First_name nvarchar(50),
						@Last_name nvarchar(50),
						@Patronymic nvarchar(50)
AS BEGIN
	SET NOCOUNT ON
	DECLARE @count int
	DECLARE ChekName CURSOR LOCAL STATIC FOR
		SELECT First_name, Last_name, Patronymic
		FROM dbo.Users
		WHERE dbo.Users.First_name = @First_name AND
		dbo.Users.Last_Name = @Last_name AND
		dbo.Users.Patronymic = @Patronymic

	OPEN ChekName
	SET @count = @@CURSOR_ROWS
	CLOSE ChekName
	DEALLOCATE ChekName;
	return @count
END

IF OBJECT_ID('dbo.UserPay') IS NOT NULL
BEGIN
    DROP PROC dbo.UserPay
END
GO
CREATE PROCEDURE dbo.UserPay	@Login nvarchar(50),
						@Summ real
```

```
AS
	BEGIN TRY
		SET NOCOUNT ON

		SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
		BEGIN TRAN
			DECLARE @User_ID int
			DECLARE @ResBalance real

			SELECT @User_ID = dbo.Accounts.User_ID
			FROM dbo.Accounts
			WHERE dbo.Accounts.Login = @Login
			IF(@@ROWCOUNT = 0)
				THROW 50007, 'Такого пользователя не существует', 1;

			SELECT @ResBalance = dbo.Users.Payment_balance
			FROM dbo.Users
			WHERE dbo.Users.User_ID = @User_ID

			UPDATE dbo.Users
			SET dbo.Users.Payment_balance = @ResBalance + @Summ
			WHERE dbo.Users.User_ID = @User_ID

			INSERT INTO dbo.Payment_log(User_ID, Date_payment, Sum_payment)
			values(@User_ID, GETDATE(), @Summ)

			COMMIT TRAN
		RETURN 1
	END TRY
	BEGIN CATCH
		print error_number()
		print error_message()
		rollback tran
		return 0
	END CATCH


IF OBJECT_ID('dbo.CheckPay') IS NOT NULL
BEGIN
    DROP PROC dbo.CheckPay
END
GO
CREATE PROC dbo.CheckPay @Login nvarchar(50)
AS
	BEGIN TRY
		SET NOCOUNT ON

		SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
		BEGIN TRAN
		DECLARE @User_ID int
		DECLARE @ResBalance real
		DECLARE @Monthly_payment real


		SELECT @User_ID = dbo.Accounts.User_ID
		FROM dbo.Accounts
		WHERE dbo.Accounts.Login = @Login
		IF(@@ROWCOUNT = 0)
			THROW 50007, 'Такого пользователя не существует', 1;

		SELECT @ResBalance = dbo.Users.Payment_balance
		FROM dbo.Users
		WHERE dbo.Users.User_ID = @User_ID

		SELECT @Monthly_payment = dbo.Tariffs.Monthly_payment
```

```
		FROM dbo.Users
		INNER JOIN dbo.Tariffs ON dbo.Users.Tariff_ID = dbo.Tariffs.Tariff_ID
		WHERE dbo.Users.User_ID = @User_ID

		IF(@ResBalance-@Monthly_payment / 30 < 0)
			UPDATE dbo.Accounts SET dbo.Accounts.IsActive = 0 WHERE dbo.Ac-
counts.Login = @Login
		ELSE
			UPDATE dbo.Accounts SET dbo.Accounts.IsActive = 1 WHERE dbo.Ac-
counts.Login = @Login

		UPDATE dbo.Users
		SET dbo.Users.Payment_balance = @ResBalance-@Monthly_payment / 30
		WHERE dbo.Users.User_ID = @User_ID

		COMMIT TRAN
		RETURN 1
	END TRY
	BEGIN CATCH
		print error_number()
		print error_message()
		ROLLBACK TRAN
		RETURN 0
	END CATCH


IF OBJECT_ID('dbo.DailyPayment') IS NOT NULL
BEGIN
    DROP PROC dbo.DailyPayment
END
GO
CREATE PROC dbo.DailyPayment
AS
	BEGIN TRY
		SET NOCOUNT ON

		SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
		BEGIN TRAN

		DECLARE getAllUsersCursor CURSOR
		FOR SELECT dbo.Users.User_ID, dbo.Tariffs.Monthly_payment, dbo.Users.Pay-
ment_balance
			FROM dbo.Tariffs
			INNER JOIN dbo.Users ON dbo.Tariffs.Tariff_ID = dbo.Users.Tariff_ID
			INNER JOIN dbo.Accounts ON dbo.Accounts.User_ID = dbo.Users.User_ID
				INNER JOIN dbo.User_types ON dbo.Accounts.User_type_ID =
	dbo.User_types.User_type_ID
			AND dbo.User_types.Type != 'admin'
		OPEN getAllUsersCursor
			IF(@@CURSOR_ROWS = 0)
				THROW 50008, 'Пользователей нет.', 1;
			DECLARE @User_ID int,
					@Monthly_payment real,
					@Payment_balance real
			FETCH NEXT FROM getAllUsersCursor INTO @User_ID, @Monthly_payment,
@Payment_balance

			WHILE @@FETCH_STATUS = 0
			BEGIN
			--------------------
			------- LOOP -------
				IF(@Payment_balance-@Monthly_payment / 30 < 0)
					UPDATE dbo.Accounts SET dbo.Accounts.IsActive = 0 WHERE
					dbo.Accounts.User_ID = @User_ID
				ELSE
```

```
                                UPDATE dbo.Accounts SET dbo.Accounts.IsActive = 1 WHERE
                                dbo.Accounts.User_ID = @User_ID

                        UPDATE dbo.Users
                                SET dbo.Users.Payment_balance = @Payment_balance-
                                @Monthly_payment / 30
                                WHERE dbo.Users.User_ID = @User_ID

                                FETCH NEXT FROM getAllUsersCursor INTO @User_ID, @Monthly_pay-
                                ment, @Payment_balance
                        ------- END -------
                        ------------------
                        END
                CLOSE getAllUsersCursor
                DEALLOCATE getAllUsersCursor;

                COMMIT TRAN
                RETURN 1
        END TRY
        BEGIN CATCH
                print error_number()
                print error_message()
                ROLLBACK TRAN
                RETURN 0
        END CATCH

GO
IF OBJECT_ID('dbo.DistanceFromUsers') IS NOT NULL
BEGIN
    DROP PROC dbo.DistanceFromUsers
END
GO
CREATE PROC dbo.DistanceFromUsers @Login1 nvarchar(50),
                                                                @Login2 nvarchar(50)
AS
BEGIN TRY
        DECLARE @User_ID1 int, @User_ID2 int;
        DECLARE @loc1 geography, @loc2 geography;

        SELECT @User_ID1 = dbo.Accounts.User_ID
        FROM dbo.Accounts
        WHERE dbo.Accounts.Login = @Login1
        IF(@User_ID1 is null)
                THROW 50007, 'Первого пользователя не существует', 1;

        SELECT @User_ID2 = dbo.Accounts.User_ID
        FROM dbo.Accounts
        WHERE dbo.Accounts.Login = @Login2
        IF(@User_ID1 is null)
                THROW 50007, 'Второго пользователя не существует', 1;

        SELECT @loc1 = dbo.Physical_addresses.Location
        FROM dbo.Physical_addresses
        WHERE dbo.Physical_addresses.User_ID = @User_ID1

        SELECT @loc2 = dbo.Physical_addresses.Location
        FROM dbo.Physical_addresses
        WHERE dbo.Physical_addresses.User_ID = @User_ID2

        SELECT @loc1.STDistance(@loc2)[Distanse(m)];
        return @loc1.STDistance(@loc2);
END TRY
BEGIN CATCH
                print error_number()
                print error_message()
```

```
            return null
END CATCH
GO


IF OBJECT_ID('dbo.NearestUsers') IS NOT NULL
BEGIN
    DROP PROC dbo.NearestUsers
END
GO
CREATE PROC dbo.NearestUsers      @Login nvarchar(50)
AS
BEGIN TRY
       DECLARE @User_ID int
       DECLARE @loc geography

       SELECT @User_ID = dbo.Accounts.User_ID
       FROM dbo.Accounts
       WHERE dbo.Accounts.Login = @Login
       IF(@User_ID is null)
              THROW 50007, 'Такого пользователя не существует', 1;


       SELECT @loc = dbo.Physical_addresses.Location
       FROM dbo.Physical_addresses
       WHERE dbo.Physical_addresses.User_ID = @User_ID

       SELECT TOP(7) dbo.Accounts.Login, Physical_addresses.Location.ToString()
       FROM dbo.Accounts
       INNER JOIN dbo.Physical_addresses ON dbo.Accounts.User_ID = dbo.Physical_ad-
dresses.User_ID
       WHERE Location.STDistance(@loc) IS NOT NULL
       ORDER BY Location.STDistance(@loc)

END TRY
BEGIN CATCH
            print error_number()
            print error_message()
END CATCH
```