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 Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Разработка базы данных с использованием технологии шифрования и маскирования для гостиничного комплекса

Выполнил студент Савчук Алёна Михайловна

(Ф.И.О.)

Руководитель проекта асс. Жигаровская С.А.

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

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

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

Консультант: асс. Жигаровская С.А.

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

Нормоконтролер: асс. Жигаровская С.А.

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

Курсовой проект защищен с оценкой

Минск 2018

# Введение

Пояснительная записка курсового проекта состоит из 28 страниц, 16 рисунков, 3 приложений, 3 источников литературы.

Основная цель курсового проекта: проектирование базы данных для гостиничного комплекса и изучения технологии шифрования и маскирования данных.

Пояснительная записка состоит из введения, 6 глав, заключения.

В первом разделе рассматривается архитектура базы данных.

Во втором разделе предоставлена информация о разработанных объектах базы данных.

В третьем разделе описываются процедуры импорта и экспорта таблиц в формат xml.

В четвертом разделе описывается тестирование производительности базы данных.

Пятый раздел содержит информацию о примененной технологии.

Шестой раздел содержит руководство пользователя для разработанного клиентского приложения.

В заключении описывается результат работы над курсовым проектом, выполненные цели.

Оглавление

[Введение 2](#_Toc532254011)

[Постановка задачи 4](#_Toc532254012)

[1. Разработка модели базы данных 5](#_Toc532254013)

[2. Разработка необходимых объектов 6](#_Toc532254014)

[2.1. Таблицы 6](#_Toc532254015)

[2.2. Процедуры 7](#_Toc532254016)

[2.3. Функции 7](#_Toc532254017)

[2.4. Временные таблицы 7](#_Toc532254018)

[2.5. Триггеры 7](#_Toc532254019)

[3. Описание процедур импорта и экспорта 9](#_Toc532254020)

[3.1. Процедура экспорта 9](#_Toc532254021)

[3.2. Процедура импорта 9](#_Toc532254022)

[4. Тестирование производительности 10](#_Toc532254023)

[4.1 Описание типов индексов 10](#_Toc532254024)

[4.2 Пример использования индексов в базе данных 10](#_Toc532254025)

[5. Описание технологии 13](#_Toc532254026)

[5.1. Описание 13](#_Toc532254027)

[5.2. Реализация шифрования 13](#_Toc532254028)

[5.3. Реализация маскирования 13](#_Toc532254029)

[6. Руководство пользователя 16](#_Toc532254030)

[6.1. Приложение для пользователя user 16](#_Toc532254031)

[6.2. Приложение для пользователя receptionist 17](#_Toc532254032)

[Заключение 18](#_Toc532254033)

[Литература 19](#_Toc532254034)

[Приложение А 20](#_Toc532254035)

[Приложение Б 21](#_Toc532254036)

[Приложение В 27](#_Toc532254037)

# Постановка задачи

Целью курсового проекта является проектирование и разработка базы данных гостиничного комплекса. База данных должна реализовать работу гостиничного комплекса, функциями которого являются: регистрация клиента, добавление клиенту дополнительных услуг (интернет, бассейн, массаж, тренажёрный зал), определение дохода за определённый промежуток времени, защита данных клиента путём шифрования и маскирования.

Система управления базами данных (СУБД) – это комплекс языковых и программных средств, предназначенных для создания, ведения и совместного использования базы данных одним или многими пользователями.

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.

Примеры СУБД: Oracle, MS SQL Server, Microsoft Access, MySql, PostgreSQL, MongoDB и так далее. Данный курсовой проект написан в СУБД MS SQL Server.

# Разработка модели базы данных

Для базы гостиничного комплекса было разработано 5 таблиц. Структура связей представлена на рисунке 1.1. База данных была разработана в СУБД MS SQL Server. Все таблицы подвергнуты процедуре нормализации.

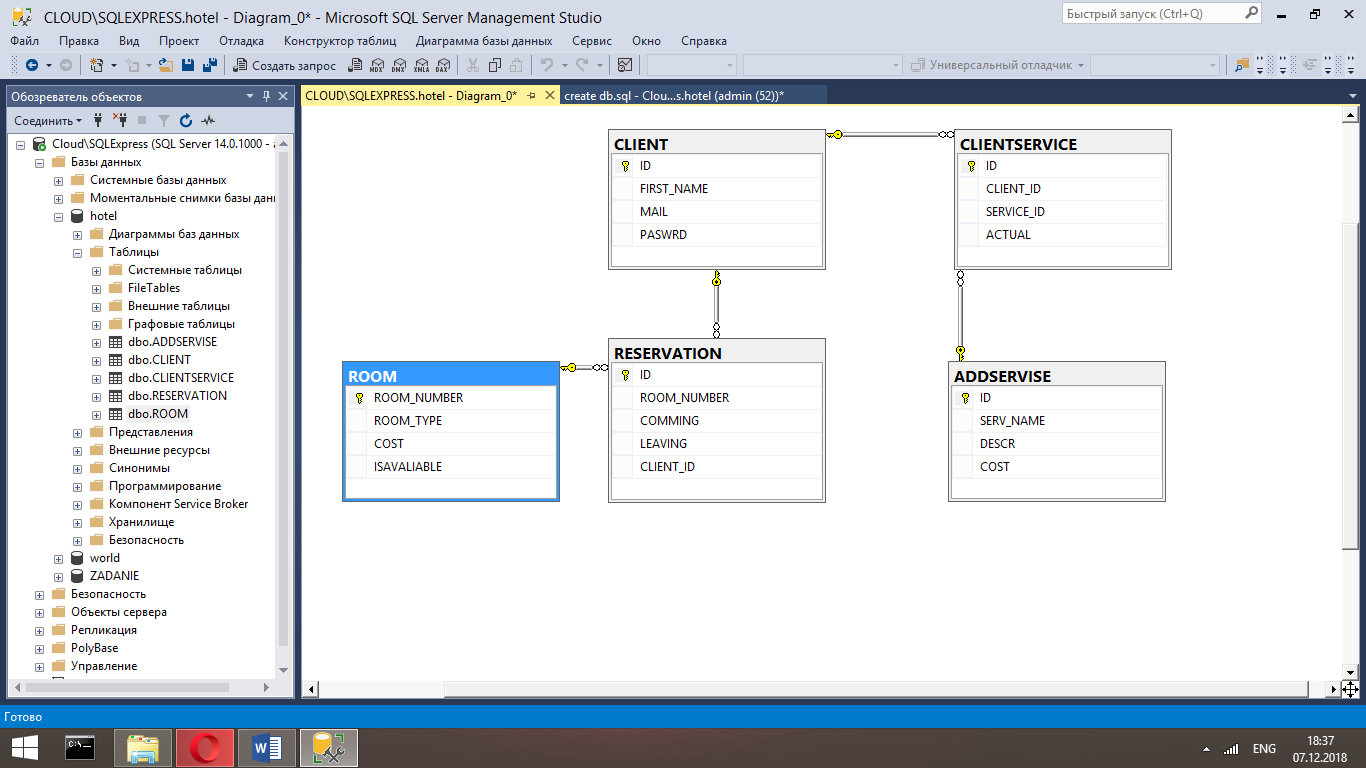


Рисунок 1.1 – Структура базы данных гостиничного комплекса

Таблица ROOM содержит информацию о комнатах гостиницы:

* ROOM\_NUMBER – первичный ключ, идентификатор комнаты;
* ROOM\_TYPE – тип комнаты(«Одноместный», «Двуместный» или «Семейный»);
* COST – цена за сутки;
* ISAVALIABLE – флаг, определяющий доступность комнаты.

Таблица ADDSERVISE содержит информацию о дополнительных услугах, доступных клиенту:

* ID – первичный ключ, идентификатор услуги;
* SERV\_NAME – название услуги;
* DESCR – описание услуги;
* COST – стоимость услуги.

Таблица CLIENTSERVICE предназначена для хранения информации о добавленных клиентом услугах:

* ID – первичный ключ, идентификатор добавленной услуги;
* CLIENT\_ID – внешний ключ, идентификатор клиента;
* SERVICE\_ID – внешний ключ, идентификатор дополнительной услуги;
* ACTUAL – флаг актуальности добавленной услуги.

Таблица CLIENT предназначена для хранения информации о зарегистрированных пользователях:

* ID – первичный ключ, идентификатор клиента;
* FIRST\_NAME – фамилия и имя клиента;
* MAIL – почтовый ящик клиента;
* PASWRD – пароль клиента.

# Разработка необходимых объектов

База данных данного курсового проекта содержит следующие объекты: таблицы, хранимые процедуры, функции, пользователи, а также временную таблицу и триггер. Подробное описание представлено в главе 2.

## Таблицы

База данных гостиничного комплекса содержит 5 таблиц, которые описаны в главе 1. Cкрипты для создания таблиц находятся в приложении А.

Пользователь базы данных – это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации. При проектировании базы данных понадобилось 3 пользователя. Каждый пользователь наделен необходимыми для него правами, для работы с базой данных гостиничного комплекса.

Пользователь admin является администратором, который имеет полный доступ к базе данных. Только admin имеет право создавать объекты базы данных hotel и управлять доступом.

Пользователь reseptionist наделен правами портье, который может получать доступ к базе данных и выполнять следующие процедуры и функции:

* CreateBillTable\_sp – создание временной таблицы с информацией о пользователе и его счетах за комнату и дополнительные услуги;
* DeterminateIncome\_sp – определение дохода за определённый промежуток времени;
* ExportUsers\_sp – экспорт клиентов в формат xml;
* GetClientBill – определение счёта клиента;
* GetServiceCost – скалярная функция для подсчёта общей суммы за все дополнительные услуги клиента;
* SelectFromUsers – функция, возвращающая табличное значение для получения всех зарегистрированных пользователей;
* SetClientServivesIrrelevant – для установки флага ACTUAL в 0;
* SetClientRoosAvaliable – для установки флага для комнаты, которую покинул клиент в 1.

Пользователь user наделён правами клиента и может обращаться к базе данных через следующие процедуры:

* AddServeces\_sp – добавление дополнительной услуги;
* get\_current\_health\_pat – получение информации о текущем состоянии здоровья;
* BookRoom\_sp – бронирование номера;
* GetAvaliableRooms\_sp – получение всех не занятых на данный момент комнат;
* IsUserRegistered – скалярная функция для определения регистрации пользователя;
* Login\_sp – авторизация пользователя;
* RegisterUser – регистрация клиента.

## Процедуры

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

Все процедуры, использованные в курсовом проекте, находятся в приложении Б.

## Функции

Функции бывают двух видов: встроенные (для работы с датами, числами, строками и т.п.) и пользовательские.

Определяемая пользователем функция представляет собой подпрограмму Transact-SQL или среды CLR, которая принимает параметры, выполняет действия, такие как сложные вычисления, а затем возвращает результат этих действий в виде значения. Возвращаемое значение может быть скалярным значением или таблицей.

Все функции, использованные в курсовом проекте, находятся в приложении В.

## Временные таблицы

Временные таблицы удобны при обработке данных — особенно во время преобразования, где промежуточные результаты являются временными. В хранилище данных SQL временные таблицы существуют на уровне сеанса. Их можно просмотреть только в сеансе, в котором они были созданы. После выхода из сеанса они автоматически удаляются. Временные таблицы позволяют оптимизировать производительность.

Временная таблица, созданная для хранения промежуточных вычислений счёта клиента представлена на рисунке 2.2.

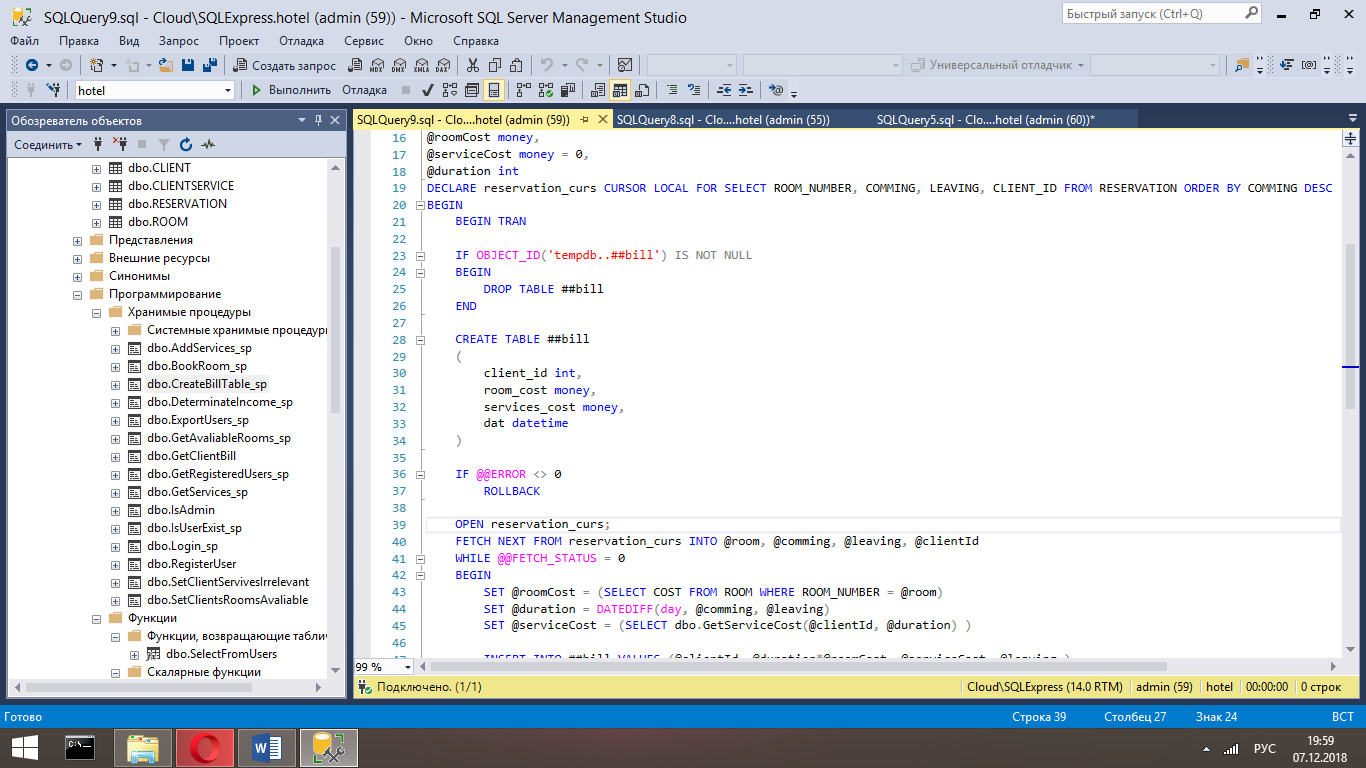


Рисунок 2.1 – Временная таблица для хранения промежуточных вычислений

## Триггеры

Триггеры представляют специальный тип хранимой процедуры, которая вызывается автоматически при выполнении определенного действия над таблицей или представлением, в частности, при добавлении, изменении или удалении данных, то есть при выполнении команд INSERT, UPDATE, DELETE.

В курсовой работе был использован триггер типа INSERT. Скрипт для создания триггера приведён на рисунке 2.2.

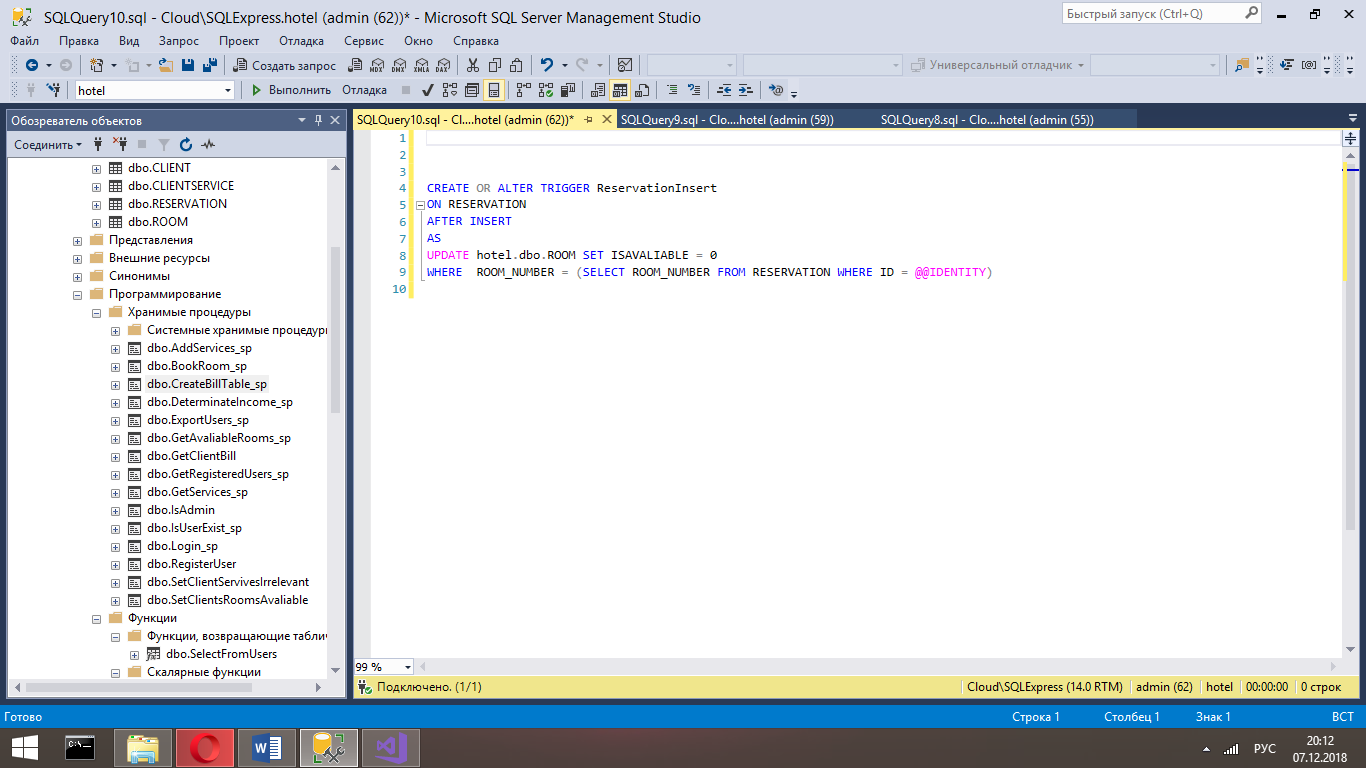


Рисунок 2.2 – Скрипт создания триггера

# 3. Описание процедур импорта и экспорта

В курсовой работе реализованы процедуры экспорта данных в xml файл из базы данных и импорта данных из xml файла в базу данных.

**3.1. Процедура экспорта**

В данной курсовой работе я реализовала импорт с помощью FOR XML функции и расширенной хранимой процедуры xp\_cmdshell для сохранения результата в файл.

Код создания процедуры экспорта данных таблицы в xml формат представлен на рисунке 3.1.

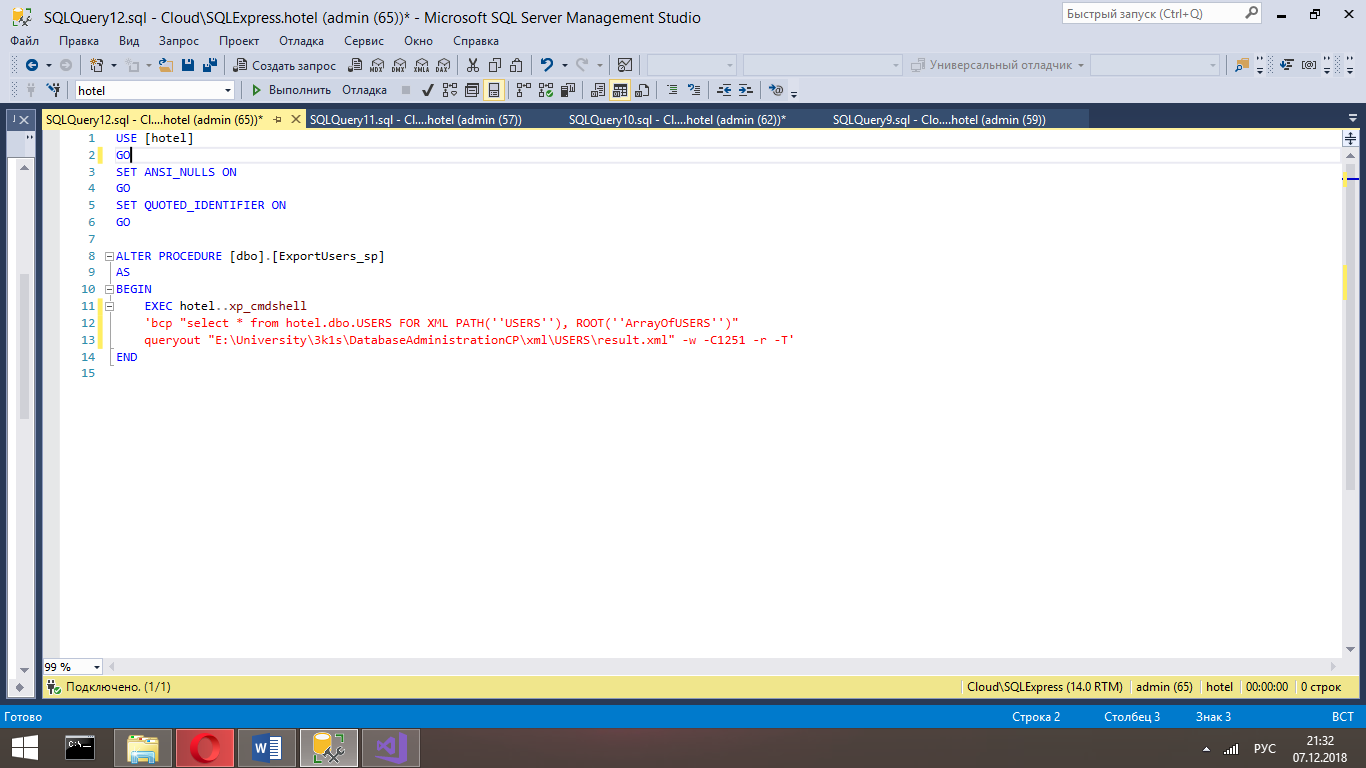


Рисунок 3.1 – Процедура экспорта

Процедура экспорта применяется для таблиц ADDSERVICE и ROOM.

**3.2. Процедура импорта**

При регистрации клиента данные из приложения передают в хранимую процедуру xml, из которого MS SQL Server извлекает данные с помощью курсора.

Код создания процедуры импорта представлен на рисунке 3.2.

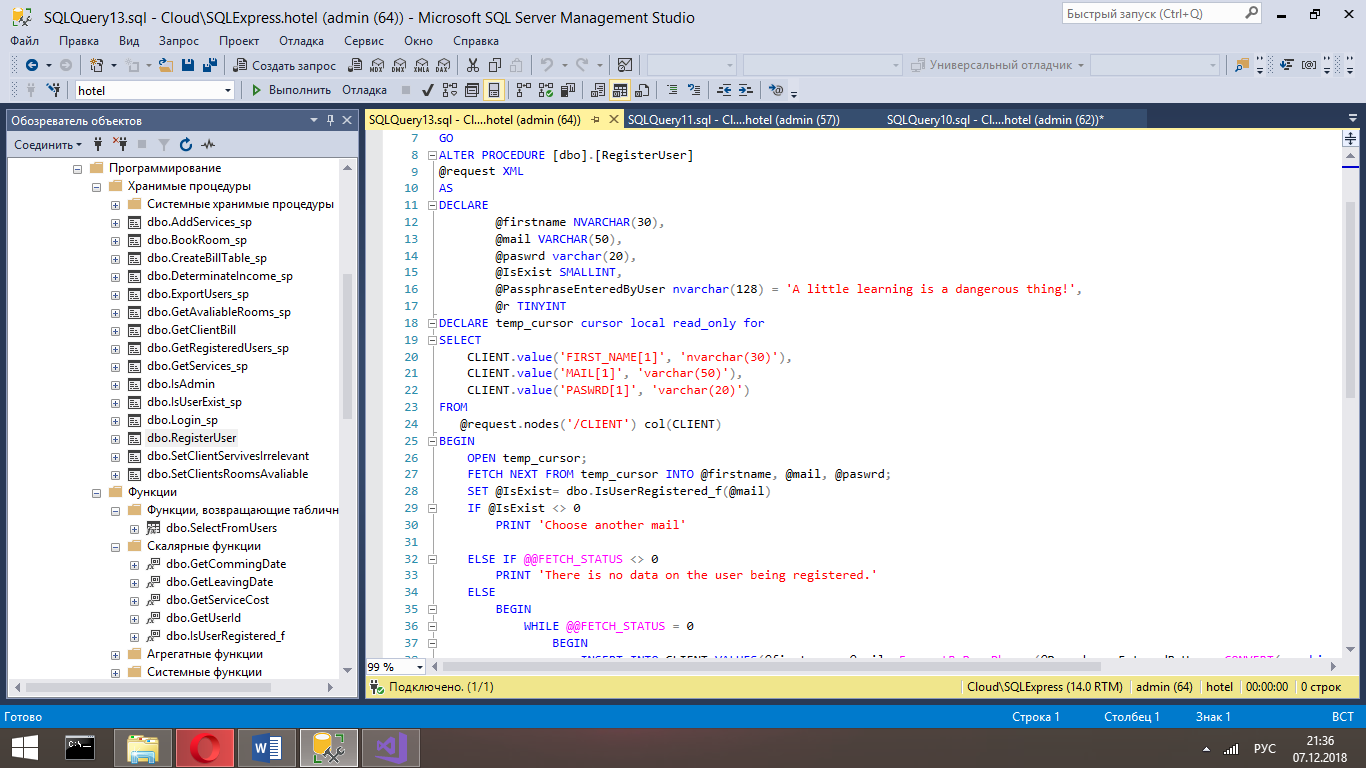


Рисунок 3.2 – Процедура импорта

**4. Тестирование производительности**

Оптимизация запросов — процесс изменения запроса и/или структуры БД с целью уменьшения использования вычислительных ресурсов при выполнении запроса. Один и тот же результат может быть получен СУБД различными способами (планами выполнения запросов), которые могут существенно отличаться как по затратам ресурсов, так и по времени выполнения.

В MS SQL Server оптимизация запросов в основном заключается в построение индексов над таблицами, и изменением плана запроса. Индекс – это объект базы данных, предназначенный для ускорения запросов к данным в таблице базы данных. MSS поддерживает несколько типов индексов: кластеризованные и некластеризованные индексы, уникальные и неуникальные, XML-индексы, пространственные индексы, а также полнотекстовые индексы.

В реляционной СУБД оптимальный план выполнения запроса — это такая последовательность применения операторов реляционной алгебры к исходным и промежуточным отношениям, которое для конкретного текущего состояния БД (её структуры и наполнения) может быть выполнено с минимальным использованием вычислительных ресурсов.

**4.1 Описание типов индексов**

При создании кластеризованного индекса данные индексируемой таблицы располагаются в физическом порядке, соответствующем индексу, и становятся частью кластеризованного индекса. Поэтому кластеризованный индекс для таблицы может быть создан только один.

Некластеризованный индекс – это отдельный объект, имеющий указатели на строки таблицы. Максимальное количество некластерированных индексов для одной таблицы не должно превышать 1000.

При создании индекса максимальная суммарная длина полей допускается равной 900 байт. Причем индекс может быть создан по полям, типы которых допускают большую длину полей, но при этом хранящиеся данные обязательно должны быть меньше.

В реляционной СУБД оптимальный план выполнения запроса — это такая последовательность применения операторов реляционной алгебры к исходным и промежуточным отношениям, которое для конкретного текущего состояния БД (её структуры и наполнения) может быть выполнено с минимальным использованием вычислительных ресурсов.

При создании индекса указывается один или несколько столбцов таблицы, по значениям которых будет построен и поддерживаться индекс. Индекс представляет собой структуру памяти, организованную в виде сбалансированного дерева.

**4.2 Пример использования индексов в базе данных**

В базе данных данного курсового проекта в каждой таблице находится поле с уникальным значением поля типа int, соответственно каждая таблица содержит кластеризованный индекс. База данных содержит множество процедур с выборкой содержащий оператор WHERE сравнивающий строки. Даже при незначительном заполнении базы данных (около 100000 записей в таблице) время выборки начинает занимать существенное время. Ниже на рисунке 4.1 представлена карта запроса при выборке в таблице CLIENTSERVICE.

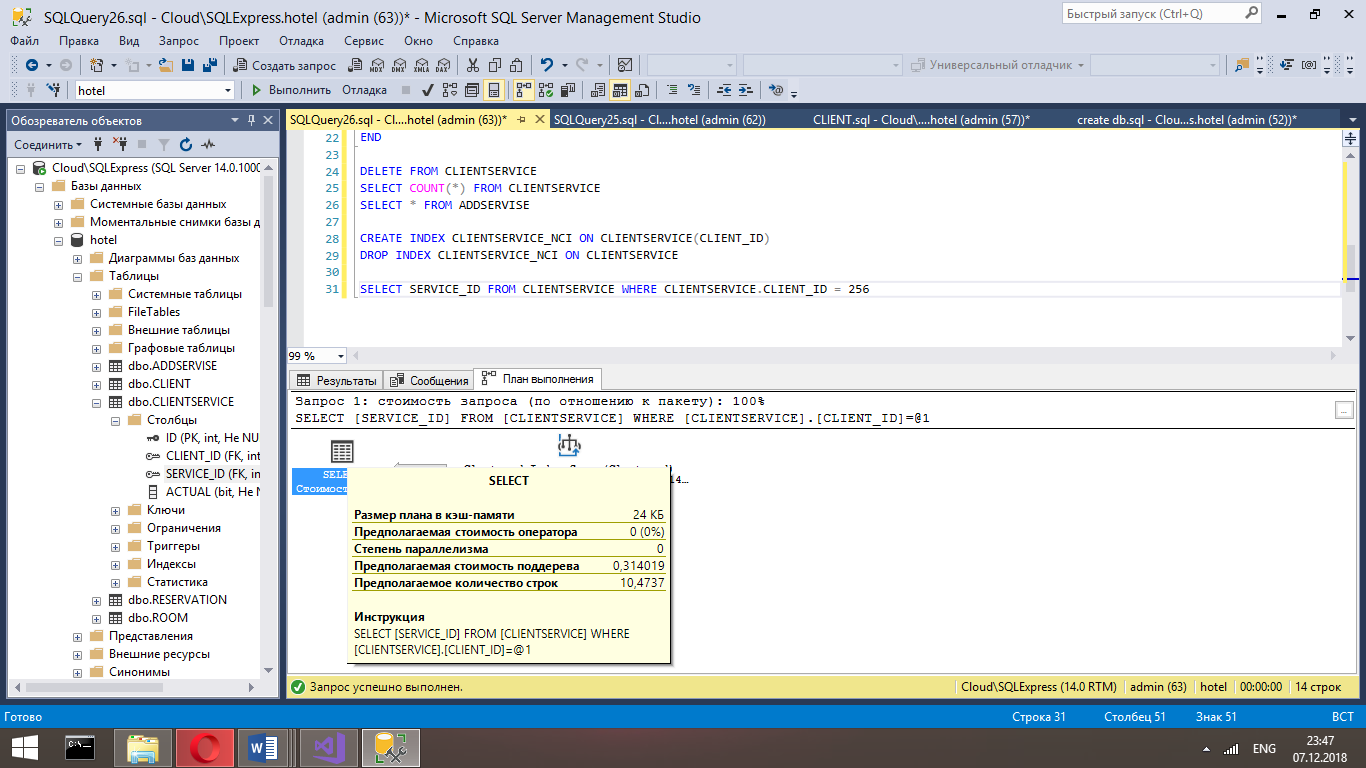


Рисунок 4.1. Карта запроса без оптимизации

Как можно увидеть, стоимость запроса равна 0,314019, что достаточно много. Так как в секции WHERE используется поле CKIENT\_ID, создадим на его основе некластеризованный индекс. Новый результат выполнения запроса можно увидеть на рисунке 4.2.

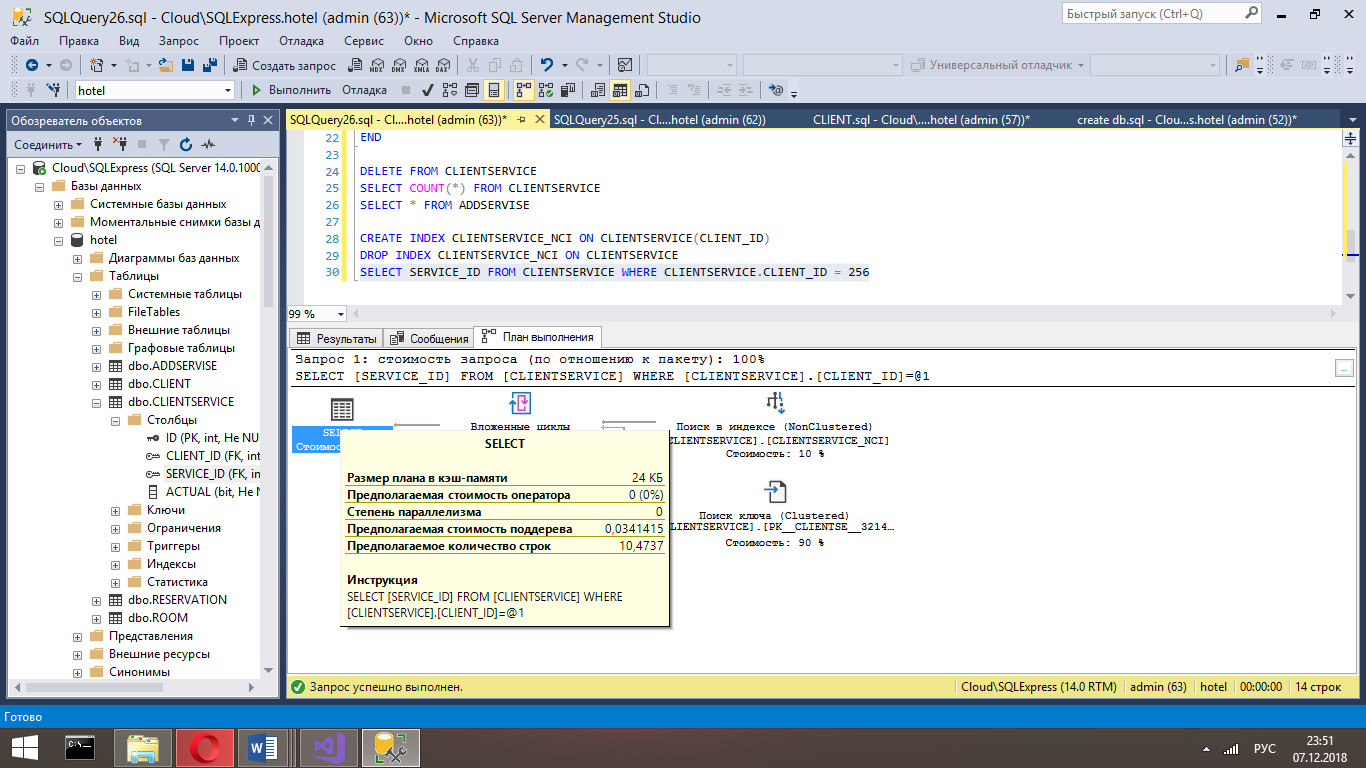


Рисунок 4.2. Карта запроса с некластеризованным индексом

Как можно заметить стоимость запроса значительно уменьшилась (до 0,0341415). Построение индексов для таблиц весьма эффективна для операций выборки с условием.

1. **Описание технологии**
   1. **Описание**

Актуальная проблема из мира информационной безопасности – обеспечить сохранность данных. Есть ситуации, в которых даже при наличии серьезной защиты системы, сохранность данных оказывается под большим вопросом.

Маскировка данных — это способ защиты конфиденциальной информации от несанкционированного доступа путём замены исходных данных фиктивными данными или произвольными символами. Правило маскирования можно задать для столбца в таблице, чтобы замаскировать данные в этом столбце. Доступны четыре типа маскирования: по умолчанию, Email, случайные, пользовательская строка.

Шифрование — использование технологии шифрования для преобразования информации, хранящейся в базе данных (БД), в шифротекст, что делает ее прочтение невозможным для лиц, не обладающих ключами шифрования

Несомненно, между маскированием и шифрованием много сходства. А оптимальной защиты данных можно добиться, только если использовать их совместно. Для шифрования большую роль играет обратимость процесса. Для маскирования обратимость – недостаток. Ни при каких условиях пользователь не должен видеть первоначальную информацию, которая была замаскирована.

* 1. **Реализация шифрования**

Шифрование данных происходит с использованием парольной фразы с использованием алгоритма TRIPLE DES и 128-битного ключа. Из парольной фразы создается симметричный ключ.

Реализация технологии шифрования показана на рисунках 5.1 и 5.2.

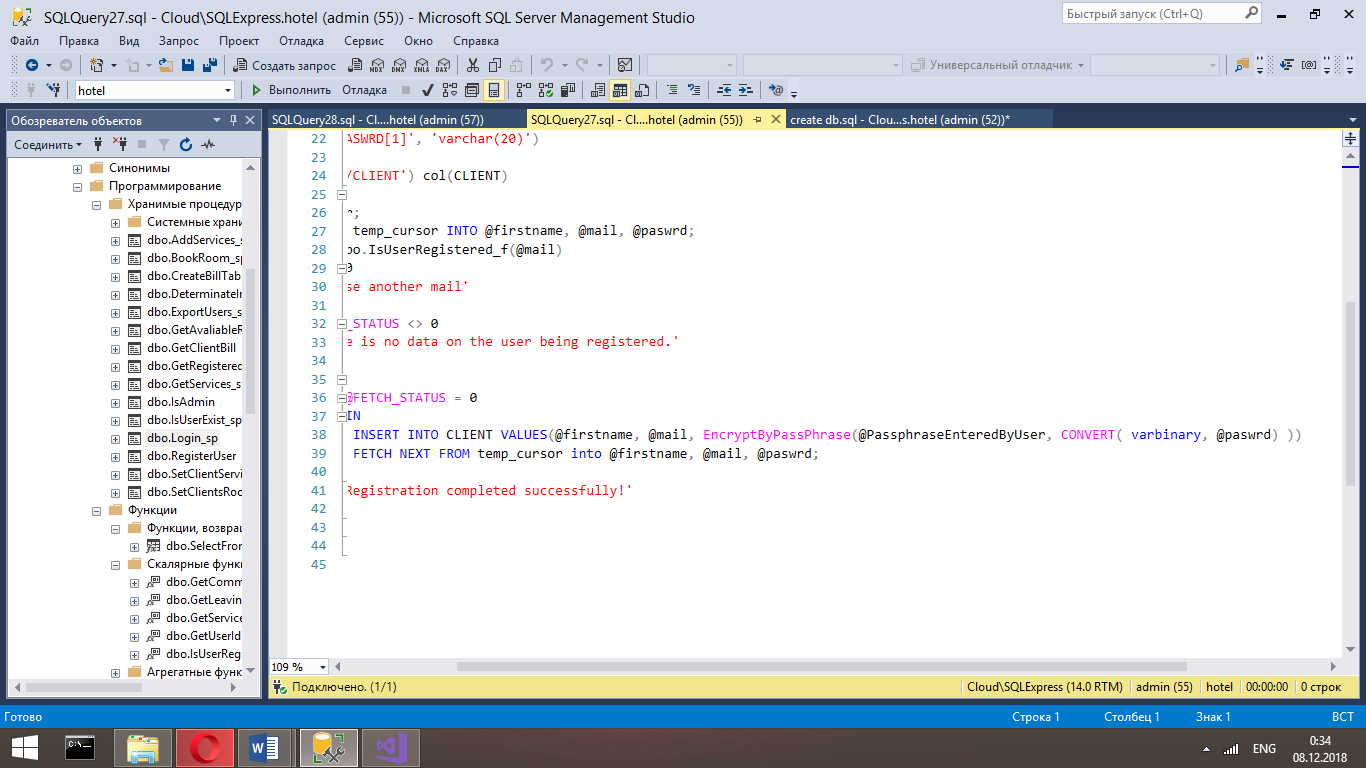


Рисунок 5.1. Шифрование пароля

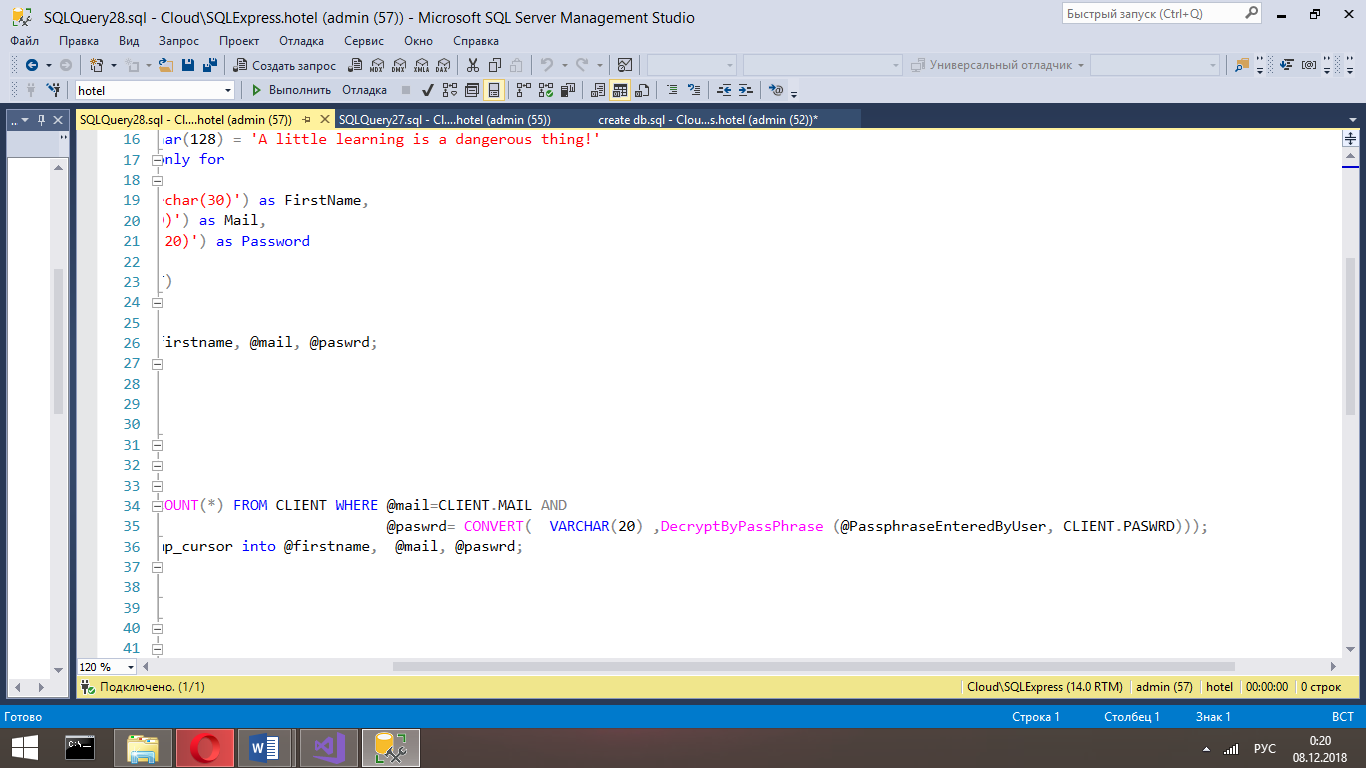


Рисунок 5.2. Дешифрование пароля

* 1. **Реализация маскирования**

Для реализации маскирования в таблице CLIENT при создании я указала, что столбец MAIL будет замаскирован. Это можно увидеть на рисунке 5.3.

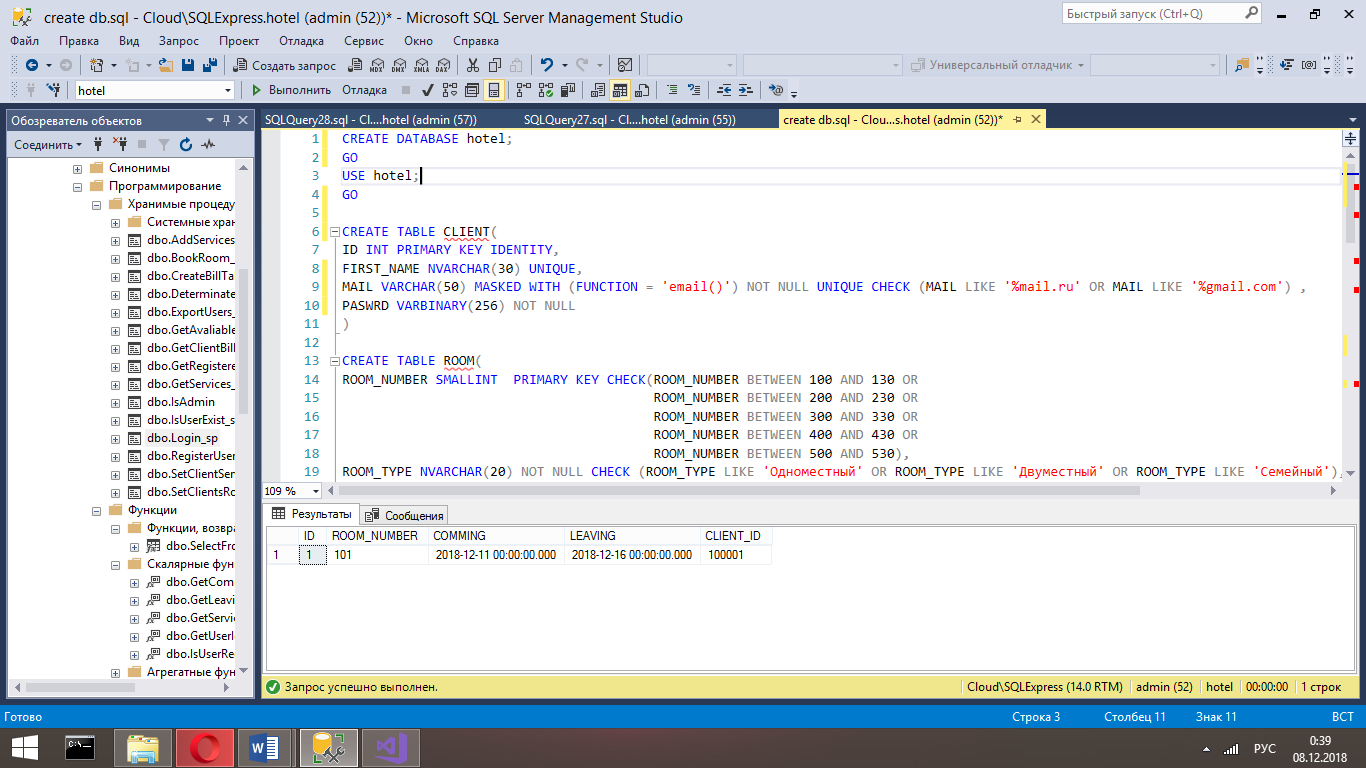


Рисунок 5.3. Маскирование почты клиента

Пользователь, от которого была реализована данная технология будет видеть данные в из первоначальном виде. Это можно увидеть на рисунке 5.4.

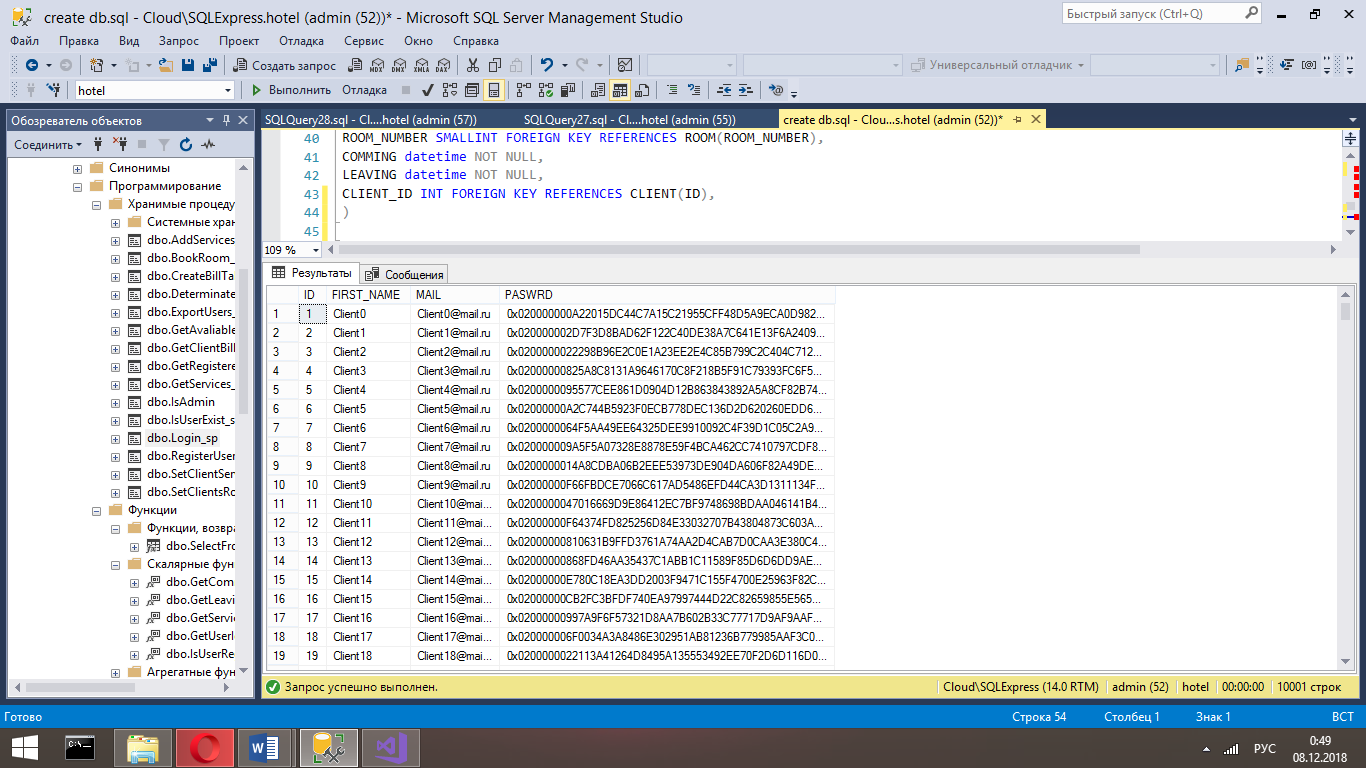


Рисунок 5.4. Просмотр содержимого от создателя таблицы

Другие пользователи, имеющие право SELECT на данную таблицу, будут видеть маскированные данные. Однако если предоставить разрешение UNMASK, то возможность просматривать немаскированные данные другим пользователям будет доступна.

Это продемонстрировано на рисунке 5.5.

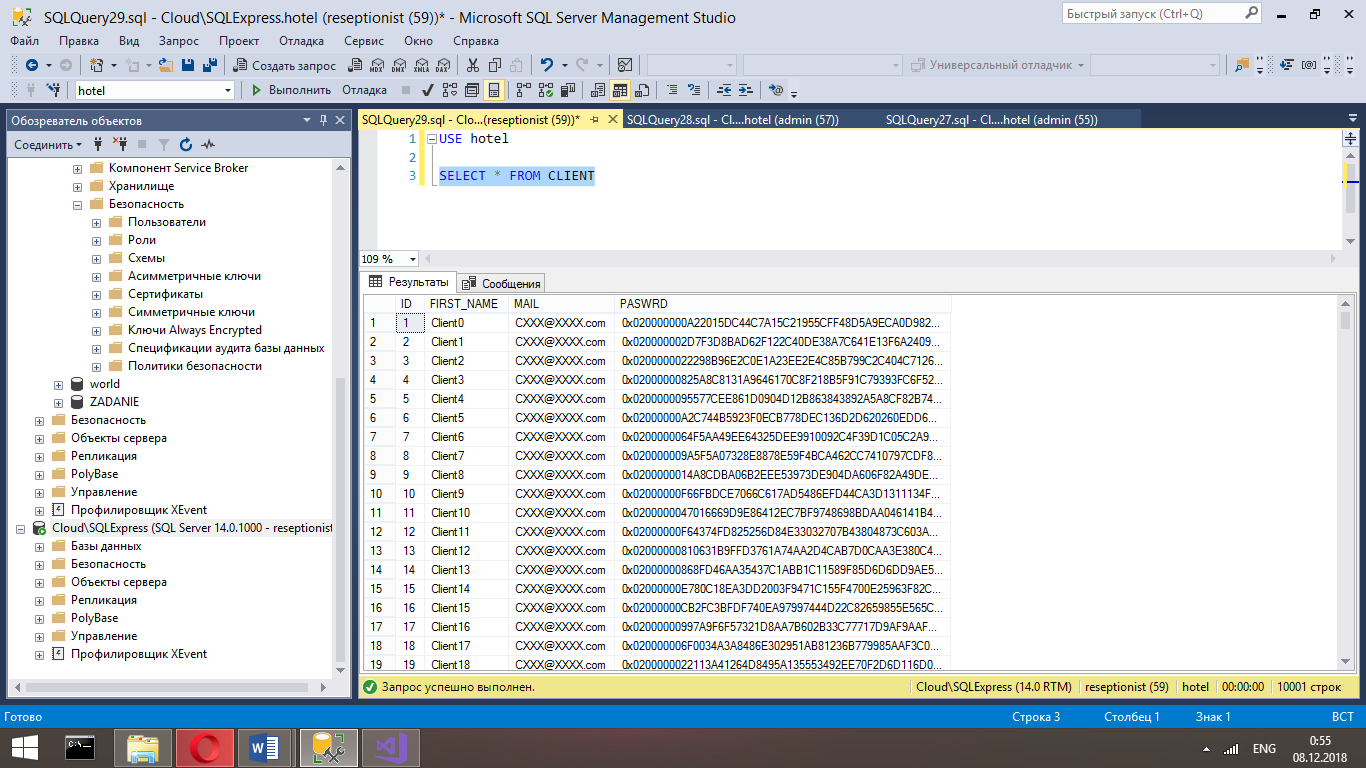


Рисунок 5.5. Просмотр содержимого от другого пользователя

1. **Руководство пользователя**

Пользовательское приложение предоставляет собой desktop приложение для взаимодействия с базой данных. Оно было реализовано с помощью языка C# с использованием технологии WPF. Исходный код программы прилагается на электронном носителе.

* 1. **Приложение для пользователя user**

При запуске приложения пользователь попадает на начальное окно, где он может зарегистрироваться или войти в систему под ранее зарегистрированным пользователем. Окно авторизации, представлено на рисунке 6.1.

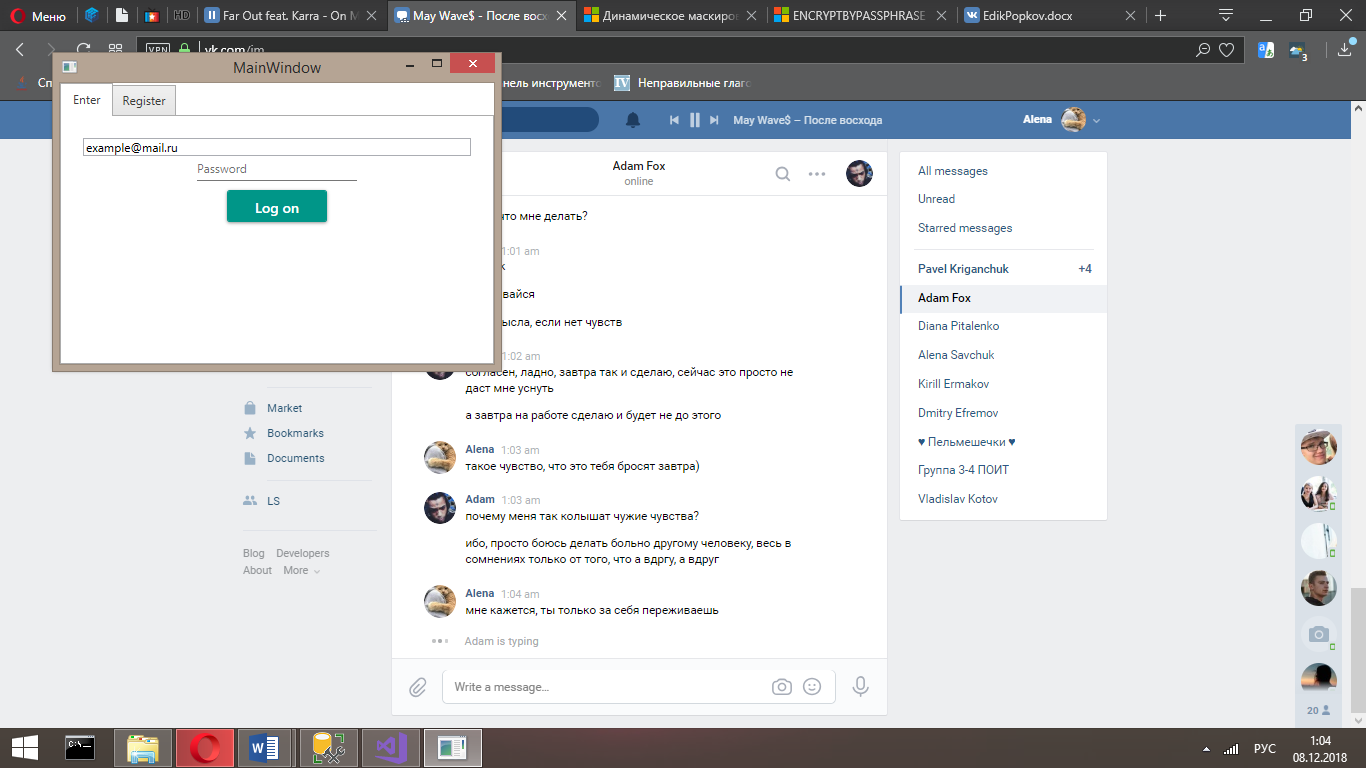


Рисунок 6.1 – Окно авторизации и регистрации

Далее пользователю предоставляется возможность забронировать комнату и добавить дополнительные услуги.

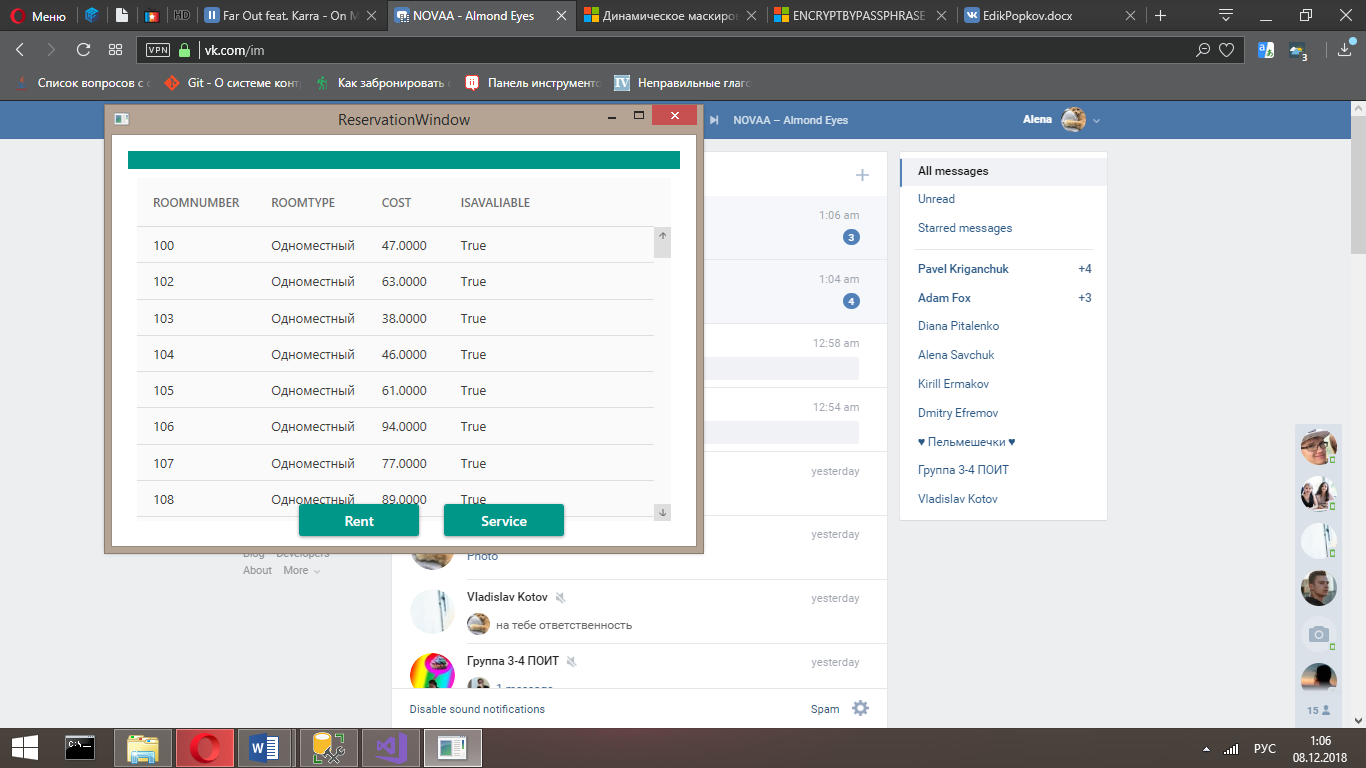


Рисунок 6.2 – Окно для бронирования номера

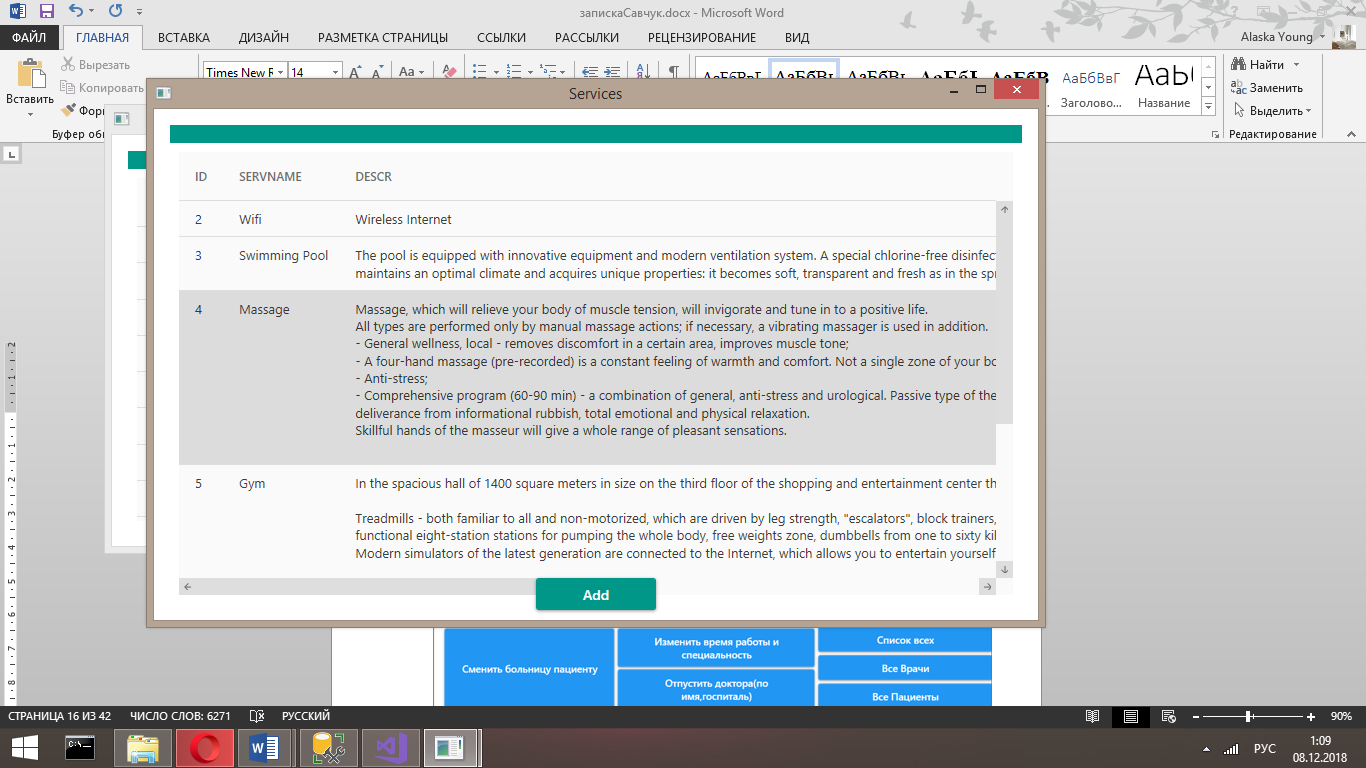


Рисунок 6.3 – Окно для добавления дополнительных услуг

* 1. **Приложение для пользователя receptionist**

Главное окно для портье предоставляет подсчитать доход гостиничного комплекса за определённый промежуток времени и определить счёт для определённого клиента. Результат будет выводится в левой части окна. Пример подсчёта дохода за определённый промежуток времени представлен на рисунке 6.4.

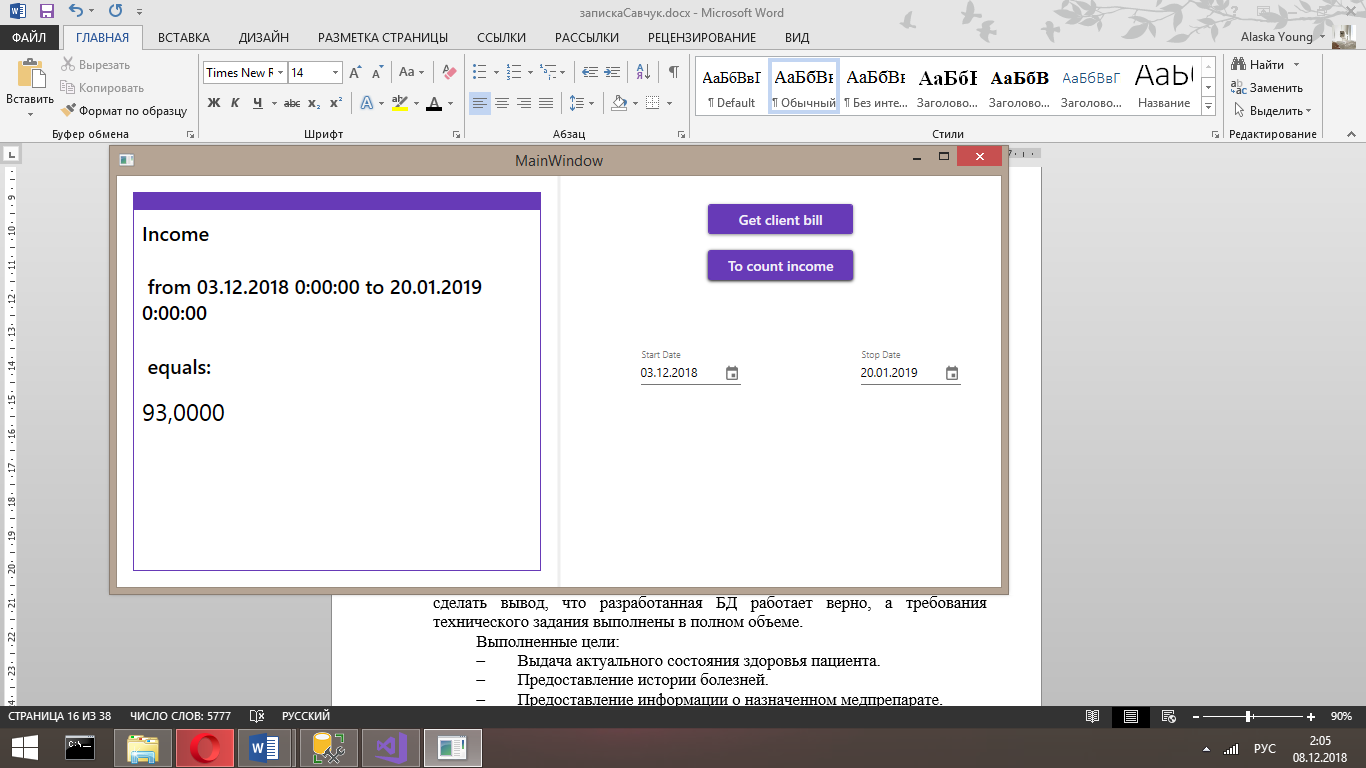


Рисунок 6.4 – Выполнение подсчёта дохода за определённый промежуток времени

**Заключение**

В данном курсовом проекте была разработана база данных для гостиничного комплекса с использованием технологии шифрования и маскирования для гостиничного комплекса.

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

Выполненные цели:

* регистрация клиента;
* добавление клиенту дополнительных услуг (интернет, бассейн, массаж, тренажёрный зал);
* шифрование и маскирование данных клиента;
* определение дохода за определённый промежуток времени;
* Так же были проведены импорт и экспорт данных формата xml.

# Литература

1. В.В. Смелов, Л. С. Мороз Microsoft SQL Server 2008: основы Transact-SQL

2. Документация Microsoft [Электронный ресурс] – режим доступа

https://docs.microsoft.com/ru-ru/sql/relational-databases/database-features?view=sql-server-2017 Дата доступа 20.11.2018

3. METANIT.COM (MS SQL Server и T-SQL) [Электронный ресурс] – режим доступа -  [https://metanit.com/sql/sqlserver/1.1.php Дата доступа – 15.11.201](https://docs.microsoft.com/ru-ru/sql/relational-databases/import-export/examples-of-bulk-import-and-export-of-xml-documents-sql-server%20Дата%20доступа%20–%2007.12.017)8.

# Приложение А

Описание создания таблиц базы данных

CREATE DATABASE hotel;

GO

USE hotel;

GO

CREATE TABLE CLIENT(

ID INT PRIMARY KEY IDENTITY,

FIRST\_NAME NVARCHAR(30) UNIQUE,

MAIL VARCHAR(50) MASKED WITH (FUNCTION = 'email()') NOT NULL UNIQUE CHECK (MAIL LIKE '%mail.ru' OR MAIL LIKE '%gmail.com') ,

PASWRD VARBINARY(256) NOT NULL

)

CREATE TABLE ROOM(

ROOM\_NUMBER SMALLINT PRIMARY KEY CHECK(ROOM\_NUMBER BETWEEN 100 AND 130 OR

ROOM\_NUMBER BETWEEN 200 AND 230 OR

ROOM\_NUMBER BETWEEN 300 AND 330 OR

ROOM\_NUMBER BETWEEN 400 AND 430 OR

ROOM\_NUMBER BETWEEN 500 AND 530),

ROOM\_TYPE NVARCHAR(20) NOT NULL CHECK (ROOM\_TYPE LIKE 'Одноместный' OR ROOM\_TYPE LIKE 'Двуместный' OR ROOM\_TYPE LIKE 'Семейный'),

COST MONEY NOT NULL,

ISAVALIABLE BIT

)

CREATE TABLE ADDSERVISE(

ID INT PRIMARY KEY IDENTITY,

SERV\_NAME VARCHAR(30) NOT NULL UNIQUE,

DESCR TEXT NULL,

COST MONEY NOT NULL

)

CREATE TABLE CLIENTSERVICE(

ID INT PRIMARY KEY IDENTITY,

CLIENT\_ID INT FOREIGN KEY REFERENCES CLIENT(ID),

SERVICE\_ID INT FOREIGN KEY REFERENCES ADDSERVISE(ID),

ACTUAL BIT NOT NULL

)

CREATE TABLE RESERVATION(

ID INT PRIMARY KEY IDENTITY,

ROOM\_NUMBER SMALLINT FOREIGN KEY REFERENCES ROOM(ROOM\_NUMBER),

COMMING datetime NOT NULL,

LEAVING datetime NOT NULL,

CLIENT\_ID INT FOREIGN KEY REFERENCES CLIENT(ID),

)

# Приложение Б

Описание процедур

ALTER PROCEDURE [dbo].[AddServices\_sp]

@service VARCHAR(30),

@client int,

@coming datetime,

@leaving datetime

AS

DECLARE

@servId int,

@n int

BEGIN

SET @n = (SELECT COUNT(\*) FROM RESERVATION WHERE RESERVATION.CLIENT\_ID = @client

AND RESERVATION.COMMING = @coming

AND RESERVATION.LEAVING = @leaving)

IF @n <> 0

BEGIN

SET @servId = (SELECT ID FROM ADDSERVISE WHERE ADDSERVISE.SERV\_NAME = @service )

INSERT INTO CLIENTSERVICE VALUES (@client, @servId, 1)

END

END

----------------------------------------------------------------------------------------- ALTER PROCEDURE [dbo].[BookRoom\_sp]

@room\_number SMALLINT,

@client\_id INT,

@come datetime,

@leave datetime

AS

DECLARE

@n int

BEGIN TRY

SET @n =(SELECT COUNT(\*) FROM RESERVATION WHERE ROOM\_NUMBER = @room\_number)

IF (@come < @leave AND

@come > GETDATE() AND

@n = 0)

INSERT INTO RESERVATION VALUES (@room\_number, @come, @leave, @client\_id)

END TRY

BEGIN CATCH

PRINT 'Error ' + CONVERT(VARCHAR, ERROR\_NUMBER()) + ':' + ERROR\_MESSAGE()

END CATCH

-----------------------------------------------------------------------------------------

ALTER PROCEDURE [dbo].[CreateBillTable\_sp]

AS

DECLARE

@room smallint,

@comming datetime,

@leaving datetime,

@clientId int,

@roomCost money,

@serviceCost money = 0,

@duration int

DECLARE reservation\_curs CURSOR LOCAL FOR SELECT ROOM\_NUMBER, COMMING, LEAVING, CLIENT\_ID FROM RESERVATION ORDER BY COMMING DESC

BEGIN

BEGIN TRAN

IF OBJECT\_ID('tempdb..##bill') IS NOT NULL

BEGIN

DROP TABLE ##bill

END

CREATE TABLE ##bill

(

client\_id int,

room\_cost money,

services\_cost money,

dat datetime

)

IF @@ERROR <> 0

ROLLBACK

OPEN reservation\_curs;

FETCH NEXT FROM reservation\_curs INTO @room, @comming, @leaving, @clientId

WHILE @@FETCH\_STATUS = 0

BEGIN

SET @roomCost = (SELECT COST FROM ROOM WHERE ROOM\_NUMBER = @room)

SET @duration = DATEDIFF(day, @comming, @leaving)

SET @serviceCost = (SELECT dbo.GetServiceCost(@clientId, @duration) )

INSERT INTO ##bill VALUES (@clientId, @duration\*@roomCost, @serviceCost, @leaving )

IF @@ERROR <> 0

ROLLBACK

SET @servicecost = 0

FETCH NEXT FROM reservation\_curs INTO @room, @comming, @leaving, @clientId

END

CLOSE reservation\_curs

IF @@ERROR <> 0

ROLLBACK

COMMIT

END

-----------------------------------------------------------------------------------------

ALTER PROCEDURE [dbo].[DeterminateIncome\_sp]

@start datetime,

@end datetime,

@income money out

AS

DECLARE

@client int,

@b money

BEGIN

BEGIN TRANSACTION

EXEC CreateBillTable\_sp

IF @@ERROR<>0

ROLLBACK

DECLARE

bill\_curs CURSOR LOCAL STATIC FOR SELECT client\_id FROM ##bill WHERE dat >= @start and dat <= @end

BEGIN

SET @income = 0

OPEN bill\_curs

FETCH NEXT FROM bill\_curs INTO @client

WHILE @@FETCH\_STATUS = 0

BEGIN

EXEC GetClientBill @client, @b out

SET @income = @income + @b

FETCH NEXT FROM bill\_curs INTO @client

END

COMMIT

CLOSE bill\_curs

END

IF @@ERROR<>0

ROLLBACK

END

---------------------------------------------------------------------------------------

ALTER PROCEDURE [dbo].[ExportUsers\_sp]

AS

BEGIN

EXEC hotel..xp\_cmdshell 'bcp "select \* from hotel.dbo.USERS FOR XML PATH(''USERS''), ROOT(''ArrayOfUSERS'')" queryout "E:\University\3k1s\DatabaseAdministrationCP\xml\USERS\result.xml" -w -C1251 -r -T'

END

-----------------------------------------------------------------------------------------

ALTER PROCEDURE [dbo].[GetAvaliableRooms\_sp]

AS

BEGIN

EXEC hotel..xp\_cmdshell 'bcp "select \* from hotel.dbo.ROOM WHERE ISAVALIABLE = 1 FOR XML PATH(''ROOM''), ROOT(''ArrayOfROOM'')" queryout "E:\University\3k1s\DatabaseAdministrationCP\xml\ROOM\result.xml" -w -C1251 -r -T'

END

--------------------------------------------------------------

ALTER PROCEDURE [dbo].[GetClientBill]

@clientId int,

@bill money out

AS

DECLARE

@roomCost money,

@serviceCost money = 0

BEGIN

BEGIN TRANSACTION

EXEC CreateBillTable\_sp

IF @@ERROR<>0

ROLLBACK

DECLARE clientbill\_curs cursor LOCAL for SELECT room\_cost, services\_cost FROM ##bill WHERE client\_id = @clientId ORDER BY dat DESC

BEGIN

OPEN clientbill\_curs

FETCH NEXT FROM clientbill\_curs INTO @roomCost, @serviceCost

SET @bill = @roomCost + @serviceCost

END

CLOSE clientbill\_curs

COMMIT

END

-------------------------------------------------------------------------------

ALTER PROCEDURE [dbo].[GetRegisteredUsers\_sp]

AS

BEGIN

SELECT \* FROM CLIENT

END

---------------------------------------------------------------------------------

ALTER PROCEDURE [dbo].[GetServices\_sp]

AS

BEGIN

EXEC hotel..xp\_cmdshell 'bcp "select \* from hotel.dbo.ADDSERVISE FOR XML PATH(''ADDSERVISE''), ROOT(''ArrayOfADDSERVISE'')" queryout "E:\University\3k1s\DatabaseAdministrationCP\xml\ADDSERVISE\result.xml" -w -C1251 -r -T'

END

---------------------------------------------------------------------------------

ALTER PROCEDURE [dbo].[IsUserExist\_sp]

@mail VARCHAR(50),

@r TINYINT OUT

AS

DECLARE

@n SMALLINT

BEGIN TRY

SET @n = (SELECT COUNT(\*) FROM CLIENT WHERE CLIENT.MAIL=@mail)

IF @n <> 0

SET @r = 1

ELSE

SET @r = 0

END TRY

BEGIN CATCH

PRINT 'Error ' + CONVERT(VARCHAR, ERROR\_NUMBER()) + ':' + ERROR\_MESSAGE()

END CATCH;

-------------------------------------------------------------------------------------

ALTER PROCEDURE [dbo].[Login\_sp]

@request XML,

@r BIT = 0 out

AS

DECLARE

@firstname NVARCHAR(30),

@mail VARCHAR(50),

@paswrd VARCHAR(20),

@PassphraseEnteredByUser nvarchar(128) = 'A little learning is a dangerous thing!'

DECLARE temp\_cursor cursor local read\_only for

SELECT

CLIENT.value('FIRST\_NAME[1]', 'nvarchar(30)') as FirstName,

CLIENT.value('MAIL[1]', 'varchar(50)') as Mail,

CLIENT.value('PASWRD[1]', 'varchar(20)') as Password

FROM

@request.nodes('/CLIENT') col(CLIENT)

BEGIN TRY

OPEN temp\_cursor;

FETCH NEXT FROM temp\_cursor INTO @firstname, @mail, @paswrd;

IF @@FETCH\_STATUS <> 0

PRINT 'Empty login data!!!'

ELSE

DECLARE @n SMALLINT

BEGIN

while @@FETCH\_STATUS = 0

begin

SET @n = ( SELECT COUNT(\*) FROM CLIENT WHERE @mail=CLIENT.MAIL AND

@paswrd= CONVERT( VARCHAR(20) ,DecryptByPassPhrase (@PassphraseEnteredByUser, CLIENT.PASWRD)));

fetch next from temp\_cursor into @firstname, @mail, @paswrd;

IF @n = 1

BREAK

end

IF @n = 1

BEGIN

SET @r = 1

END

ELSE

PRINT 'Wrong login data!!!'

END

END TRY

BEGIN CATCH

PRINT 'Error ' + CONVERT(VARCHAR, ERROR\_NUMBER()) + ':' + ERROR\_MESSAGE()

END CATCH;

---------------------------------------------------------------------------------------

ALTER PROCEDURE [dbo].[RegisterUser]

@request XML

AS

DECLARE

@firstname NVARCHAR(30),

@mail VARCHAR(50),

@paswrd varchar(20),

@IsExist SMALLINT,

@PassphraseEnteredByUser nvarchar(128) = 'A little learning is a dangerous thing!',

@r TINYINT

DECLARE temp\_cursor cursor local read\_only for

SELECT

CLIENT.value('FIRST\_NAME[1]', 'nvarchar(30)'),

CLIENT.value('MAIL[1]', 'varchar(50)'),

CLIENT.value('PASWRD[1]', 'varchar(20)')

FROM

@request.nodes('/CLIENT') col(CLIENT)

BEGIN

OPEN temp\_cursor;

FETCH NEXT FROM temp\_cursor INTO @firstname, @mail, @paswrd;

SET @IsExist= dbo.IsUserRegistered\_f(@mail)

IF @IsExist <> 0

PRINT 'Choose another mail'

ELSE IF @@FETCH\_STATUS <> 0

PRINT 'There is no data on the user being registered.'

ELSE

BEGIN

WHILE @@FETCH\_STATUS = 0

BEGIN

INSERT INTO CLIENT VALUES(@firstname, @mail, EncryptByPassPhrase(@PassphraseEnteredByUser, CONVERT( varbinary, @paswrd) ))

FETCH NEXT FROM temp\_cursor into @firstname, @mail, @paswrd;

END

PRINT 'Registration completed successfully!'

END

END

----------------------------------------------------------------------------------------

ALTER PROCEDURE [dbo].[SetClientServivesIrrelevant]

AS

DECLARE

clientServ\_curs CURSOR FOR SELECT CLIENT\_ID FROM RESERVATION WHERE

DATENAME(day, LEAVING) = DATENAME(day, GETDATE()) AND

DATENAME(month, LEAVING) = DATENAME(month, GETDATE()) AND

DATENAME(year, LEAVING) = DATENAME(year, GETDATE())

DECLARE

@client int

BEGIN

OPEN clientServ\_curs

FETCH NEXT FROM clientServ\_curs INTO @client

WHILE @@FETCH\_STATUS = 0

BEGIN

UPDATE CLIENTSERVICE SET ACTUAL = 0

WHERE CLIENT\_ID = @client

FETCH NEXT FROM clientServ\_curs INTO @client

END

CLOSE clientServ\_curs

END

----------------------------------------------------------------------------------------

ALTER PROCEDURE [dbo].[SetClientsRoomsAvaliable]

AS

DECLARE

@day int,

@month int,

@year int,

@date datetime,

@room smallint

DECLARE curs cursor local for

SELECT RESERVATION.ROOM\_NUMBER, RESERVATION.LEAVING FROM RESERVATION

BEGIN

SET @day = DAY(GETDATE())

SET @month = MONTH(GETDATE())

SET @year = YEAR(GETDATE())

OPEN curs

FETCH NEXT FROM curs INTO @room, @date

BEGIN

WHILE @@FETCH\_STATUS = 0

BEGIN

IF @day = DAY(@date) AND @month = MONTH(@date) AND @year = YEAR(@date)

UPDATE ROOM SET ISAVALIABLE = 1 WHERE ROOM\_NUMBER = @room

FETCH NEXT FROM curs INTO @room, @date

END

END

END

**Приложение В**

Функции

ALTER FUNCTION [dbo].[SelectFromUsers]

()

RETURNS TABLE

AS

RETURN (SELECT \* FROM CLIENT )

-----------------------------------------------------------------------------------------

ALTER FUNCTION [dbo].[GetCommingDate]

(

@id int

)

RETURNS datetime

AS

BEGIN

RETURN (SELECT TOP 1 COMMING FROM RESERVATION WHERE RESERVATION.CLIENT\_ID = @id ORDER BY COMMING DESC)

END

---------------------------------------------------------------------------------------------------

ALTER FUNCTION [dbo].[GetLeavingDate]

(

@id int

)

RETURNS datetime

AS

BEGIN

RETURN (SELECT TOP 1 LEAVING FROM RESERVATION WHERE RESERVATION.CLIENT\_ID = @id ORDER BY COMMING DESC)

END

-----------------------------------------------------------------------------------------

ALTER FUNCTION [dbo].[GetServiceCost]

(

@clientId int,

@duration int

)

RETURNS money

AS

BEGIN

DECLARE

@serviceCost money = 0,

@serviceId int

DECLARE curs\_services CURSOR LOCAL FOR SELECT SERVICE\_ID FROM CLIENTSERVICE WHERE CLIENTSERVICE.CLIENT\_ID = @clientId

OPEN curs\_services

FETCH NEXT FROM curs\_services INTO @serviceId

WHILE @@FETCH\_STATUS = 0

BEGIN

IF @serviceId = 2

SET @serviceCost = @serviceCost + @duration\*(SELECT COST FROM ADDSERVISE WHERE ADDSERVISE.ID = @serviceId)

ELSE

SET @serviceCost = @serviceCost + (SELECT COST FROM ADDSERVISE WHERE ADDSERVISE.ID = @serviceId)

FETCH NEXT FROM curs\_services INTO @serviceId

END

CLOSE curs\_services

RETURN @serviceCost

END

----------------------------------------------------------------------------------------------------

ALTER FUNCTION [dbo].[GetUserId]

(

@mail VARCHAR(50)

)

RETURNS int

AS

BEGIN

RETURN (SELECT ID FROM CLIENT WHERE CLIENT.MAIL = @mail)

END

-----------------------------------------------------------------------------------------

ALTER FUNCTION [dbo].[IsUserRegistered\_f]

(

@mail VARCHAR(50)

)

RETURNS int

AS

BEGIN

RETURN (SELECT COUNT(\*) FROM CLIENT WHERE CLIENT.MAIL=@mail)

END