

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
Постановка задачи.....	4
1. Разработка модели базы данных	5
2. Разработка необходимых объектов.....	6
2.1. Таблицы	6
2.2. Процедуры.....	7
2.3. Функции.....	7
2.4. Временные таблицы	7
2.5. Триггеры.....	7
3. Описание процедур импорта и экспорта	9
3.1. Процедура экспорта	9
3.2. Процедура импорта.....	9
4. Тестирование производительности	10
4.1 Описание типов индексов	10
4.3 Пример использования индексов в базе данных	10
5. Описание технологии	13
5.1. Описание	13
5.2. Реализация шифрования	13
5.3. Реализация маскирования.....	13
6. Руководство пользователя	16
6.1. Приложение для пользователя user	16
6.2. Приложение для пользователя receptionist	17
Заключение	18
Литература	19
Приложение А	20
Приложение Б	21
Приложение В.....	27

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.

1. 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.

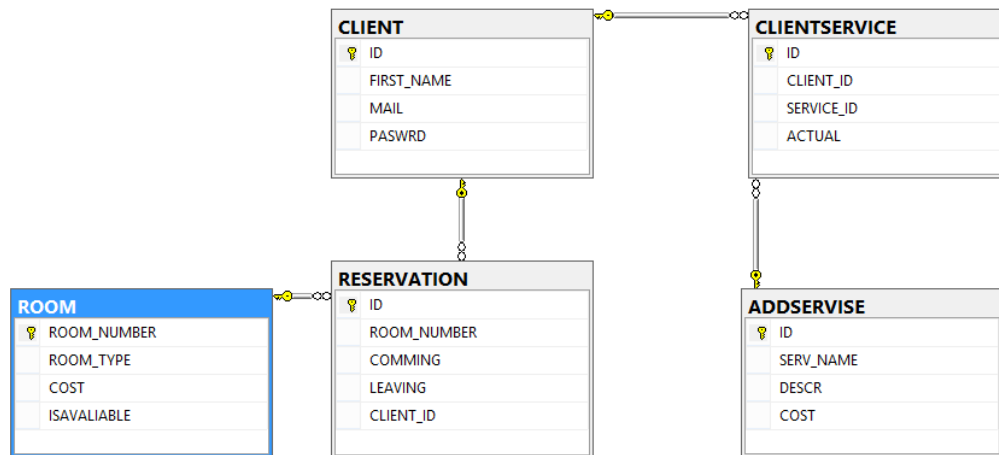


Рисунок 1.1 – Структура базы данных гостиничного комплекса

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;
- **ISAVAILABLE** – флаг, определяющий доступность комнаты.

Таблица **ADDSERVICE** 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.

2. 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.

2.1. Таблицы

База данных гостиничного комплекса содержит 5 таблиц, которые описаны в главе 1. Скрипты 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.

2.2. 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.

2.3. 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.

2.4. 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.

```
CREATE TABLE ##bill
(
    client_id int,
    room_cost money,
    services_cost money,
    dat datetime
)
```

Рисунок 2.1 – Временная таблица для хранения промежуточных вычислений

2.5. 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.

```
CREATE OR ALTER TRIGGER ReservationInsert
ON RESERVATION
AFTER INSERT
AS
UPDATE hotel.dbo.ROOM SET ISAVAILABLE = 0
WHERE ROOM_NUMBER = (SELECT ROOM_NUMBER FROM RESERVATION WHERE ID = @@IDENTITY)
```

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.

```
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
```

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.

```
8 ALTER PROCEDURE [dbo].[RegisterUser]
9     @request XML
10 AS
11 DECLARE
12     @firstname NVARCHAR(30),
13     @mail VARCHAR(50),
14     @paswrд varchar(20),
15     @IsExist SMALLINT,
16     @PassphraseEnteredByUser nvarchar(128) = 'A little learning is a dangerous thing!',
17     @r TINYINT
18 DECLARE temp_cursor cursor local read_only for
19 SELECT
20     CLIENT.value('FIRST_NAME[1]', 'nvarchar(30)'),
21     CLIENT.value('MAIL[1]', 'varchar(50)'),
22     CLIENT.value('PASWRD[1]', 'varchar(20)')
23 FROM
24     @request.nodes('/CLIENT') col(CLIENT)
25 BEGIN
26     OPEN temp_cursor;
27     FETCH NEXT FROM temp_cursor INTO @firstname, @mail, @paswrд;
28     SET @IsExist= dbo.IsUserRegistered_f(@mail)
29     IF @IsExist <> 0
30         PRINT 'Choose another mail'
31
```

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 построении 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 некластеризованных 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.

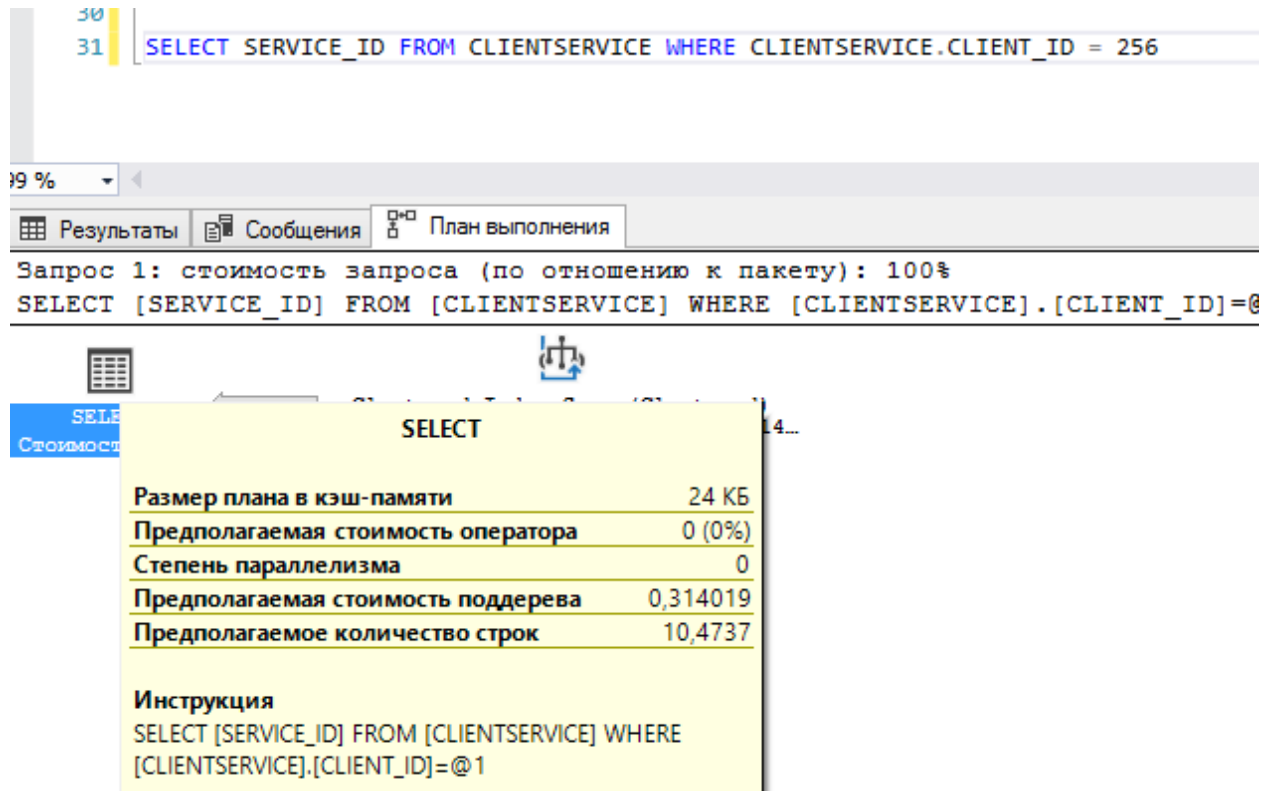


Рисунок 4.1. Карта запроса без оптимизации

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

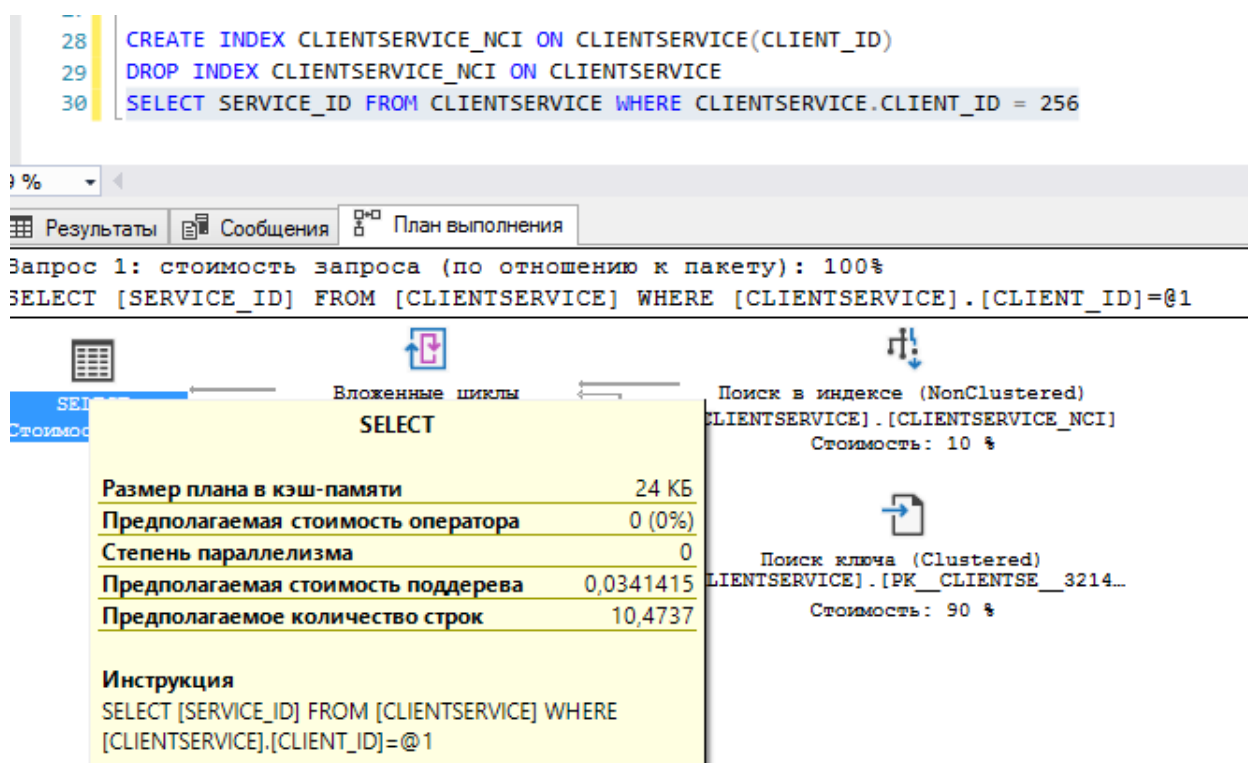


Рисунок 4.2. Карта запроса с некластеризованным индексом

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

5. Описание технологии

5.1.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.

5.2.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.

```
SQL> INSERT INTO CLIENT VALUES(@firstname, @mail, EncryptByPassPhrase(@PassphraseEnteredByUser, CONVERT( varbinary, @paswrд ) ) )  
SQL> FETCH NEXT FROM temp_cursor into @firstname, @mail, @paswrд;
```

Рисунок 5.1. Шифрование пароля

```
SQL> DECLARE @mail=CLIENT.MAIL AND  
SQL> @paswrд= CONVERT( VARCHAR(20) ,DecryptByPassPhrase (@PassphraseEnteredByUser, CLIENT.PASWRD));  
SQL> UPDATE CLIENT SET PASWRD=@paswrд, @mail, @paswrд;
```

Рисунок 5.2. Дешифрование пароля

5.3.Реализация маскирования

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

```
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
)
```

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.

Результаты		Сообщения		
	ID	FIRST_NAME	MAIL	PASWRD
1	1	Client0	Client0@mail.ru	0x020000000A22015DC44C7A15C21955CFF48D5A9ECA0D982...
2	2	Client1	Client1@mail.ru	0x0200000002D7F3D8BAD62F122C40DE38A7C641E13F6A2409...
3	3	Client2	Client2@mail.ru	0x02000000022298B96E2C0E1A23EE2E4C85B799C2C404C712...
4	4	Client3	Client3@mail.ru	0x020000000825A8C8131A9646170C8F218B5F91C79393FC6F5...
5	5	Client4	Client4@mail.ru	0x02000000095577CEE861D0904D12B863843892A5A8CF82B74...
6	6	Client5	Client5@mail.ru	0x020000000A2C744B5923F0ECB778DEC136D2D620260EDD6...
7	7	Client6	Client6@mail.ru	0x02000000064F5AA49EE64325DEE9910092C4F39D1C05C2A9...
8	8	Client7	Client7@mail.ru	0x0200000009A5F5A07328E8878E59F4BCA462CC7410797CDF8...
9	9	Client8	Client8@mail.ru	0x02000000014A8CDBA06B2EEE53973DE904DA606F82A49DE...
10	10	Client9	Client9@mail.ru	0x020000000F66FBDCE7066C617AD5486EFD44CA3D1311134F...
11	11	Client10	Client10@mai...	0x02000000047016669D9E86412EC7BF9748698BDAA046141B4...
12	12	Client11	Client11@mai...	0x020000000F64374FD825256D84E33032707B43804873C603A...
13	13	Client12	Client12@mai...	0x020000000810631B9FFD3761A74AA2D4CAB7D0CAA3E380C4...
14	14	Client13	Client13@mai...	0x020000000868FD46AA35437C1ABB1C11589F85D6D6DD9AE...
15	15	Client14	Client14@mai...	0x020000000E780C18EA3DD2003F9471C155F4700E25963F82C...
16	16	Client15	Client15@mai...	0x020000000CB2FC3BFDF740EA97997444D22C82659855E565...
17	17	Client16	Client16@mai...	0x020000000997A9F6F57321D8AA7B602B33C77717D9AF9AAF...
18	18	Client17	Client17@mai...	0x0200000006F0034A3A8486E302951AB81236B779985AAF3C0...
19	19	Client18	Client18@mai...	0x02000000022113A41264D8495A135553492EE70F2D6D116D0...

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.

Результаты		Сообщения		
	ID	FIRST_NAME	MAIL	PASWRD
1	1	Client0	CXXX@XXXX.com	0x020000000A22015DC44C7A15C21955CFF48D5A9ECA0D982...
2	2	Client1	CXXX@XXXX.com	0x0200000002D7F3D8BAD62F122C40DE38A7C641E13F6A2409...
3	3	Client2	CXXX@XXXX.com	0x02000000022298B96E2C0E1A23EE2E4C85B799C2C404C7126...
4	4	Client3	CXXX@XXXX.com	0x020000000825A8C8131A9646170C8F218B5F91C79393FC6F52...
5	5	Client4	CXXX@XXXX.com	0x02000000095577CEE861D0904D12B863843892A5A8CF82B74...
6	6	Client5	CXXX@XXXX.com	0x020000000A2C744B5923F0ECB778DEC136D2D620260EDD6...
7	7	Client6	CXXX@XXXX.com	0x02000000064F5AA49EE64325DEE9910092C4F39D1C05C2A9...
8	8	Client7	CXXX@XXXX.com	0x0200000009A5F5A07328E8878E59F4BCA462CC7410797CDF8...
9	9	Client8	CXXX@XXXX.com	0x02000000014A8CDBA06B2EEE53973DE904DA606F82A49DE...
10	10	Client9	CXXX@XXXX.com	0x020000000F66FBDCE7066C617AD5486EFD44CA3D1311134F...
11	11	Client10	CXXX@XXXX.com	0x02000000047016669D9E86412EC7BF9748698BDAA046141B4...
12	12	Client11	CXXX@XXXX.com	0x020000000F64374FD825256D84E33032707B43804873C603A...
13	13	Client12	CXXX@XXXX.com	0x020000000810631B9FFD3761A74AA2D4CAB7D0CAA3E380C4...
14	14	Client13	CXXX@XXXX.com	0x020000000868FD46AA35437C1ABB1C11589F85D6D6DD9AE5...
15	15	Client14	CXXX@XXXX.com	0x020000000E780C18EA3DD2003F9471C155F4700E25963F82C...
16	16	Client15	CXXX@XXXX.com	0x020000000CB2FC3BFDF740EA97997444D22C82659855E565C...
17	17	Client16	CXXX@XXXX.com	0x020000000997A9F6F57321D8AA7B602B33C77717D9AF9AAF...
18	18	Client17	CXXX@XXXX.com	0x0200000006F0034A3A8486E302951AB81236B779985AAF3C0...
19	19	Client18	CXXX@XXXX.com	0x02000000022113A41264D8495A135553492FF70F2D6D116D0

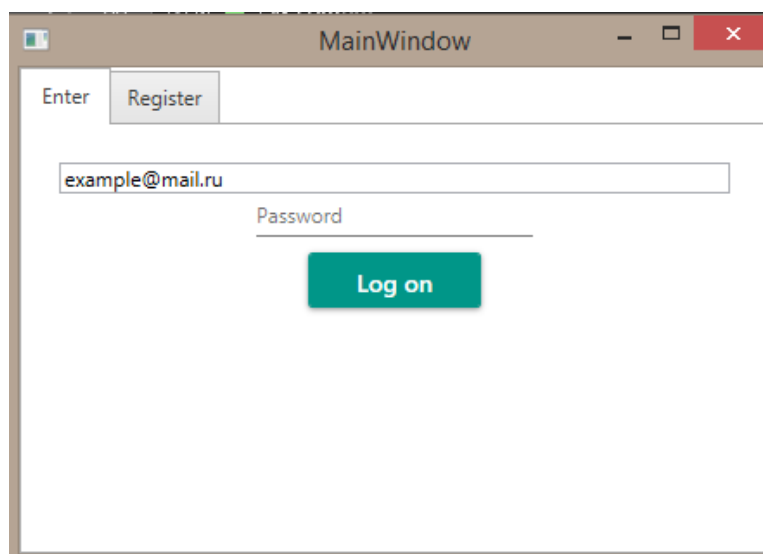
Рисунок 5.5. Просмотр содержимого от другого пользователя

6. 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.

6.1. Приложение для пользователя user

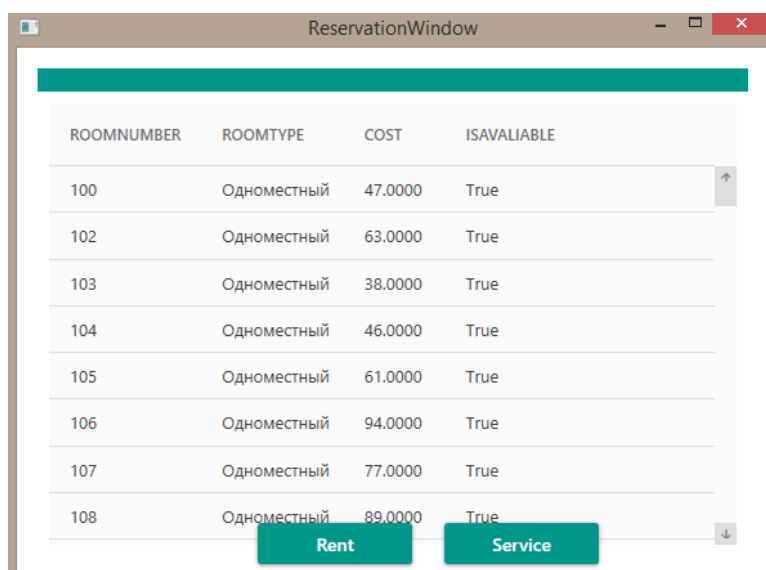
При запуске приложения пользователь попадает на начальное окно, где он может зарегистрироваться или войти в систему под ранее зарегистрированным пользователем. Окно авторизации, представлено на рисунке 6.1.



The screenshot shows a window titled 'MainWindow'. At the top, there are two buttons: 'Enter' and 'Register'. Below them is a text input field containing 'example@mail.ru'. Underneath the email field is a label 'Password' followed by a password input field. At the bottom center, there is a green button labeled 'Log on'.

Рисунок 6.1 – Окно авторизации и регистрации

Далее пользователю предоставляется возможность забронировать комнату и добавить дополнительные услуги.



The screenshot shows a window titled 'ReservationWindow'. It contains a table with the following columns: ROOMNUMBER, ROOMTYPE, COST, and ISAVAILABLE. The table lists 8 rooms, all of which are 'Одноместный' (Single) and have 'ISAVAILABLE' set to 'True'. Below the table, there are two green buttons: 'Rent' and 'Service'.

ROOMNUMBER	ROOMTYPE	COST	ISAVAILABLE
100	Одноместный	47.0000	True
102	Одноместный	63.0000	True
103	Одноместный	38.0000	True
104	Одноместный	46.0000	True
105	Одноместный	61.0000	True
106	Одноместный	94.0000	True
107	Одноместный	77.0000	True
108	Одноместный	89.0000	True

Рисунок 6.2 – Окно для бронирования номера

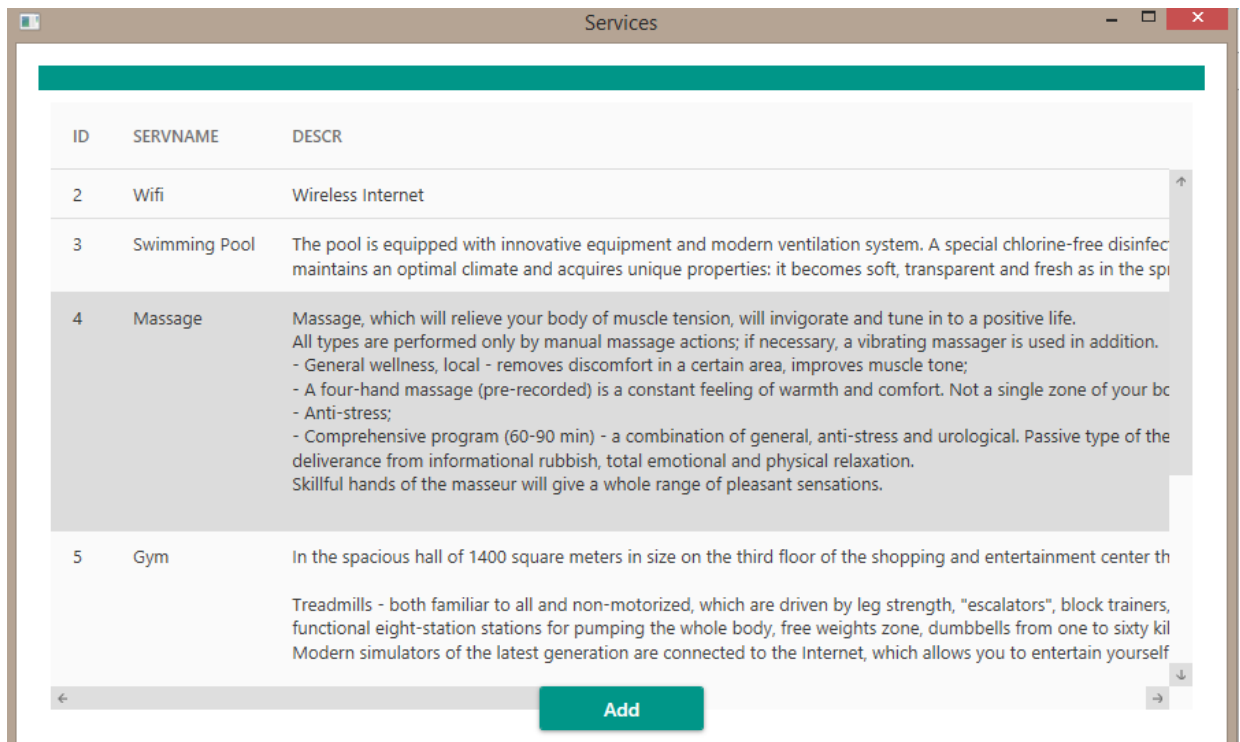


Рисунок 6.3 – Окно для добавления дополнительных услуг

6.2. 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 выводиться в левой части окна. Пример подсчёта дохода за определённый промежуток времени представлен на рисунке 6.4.

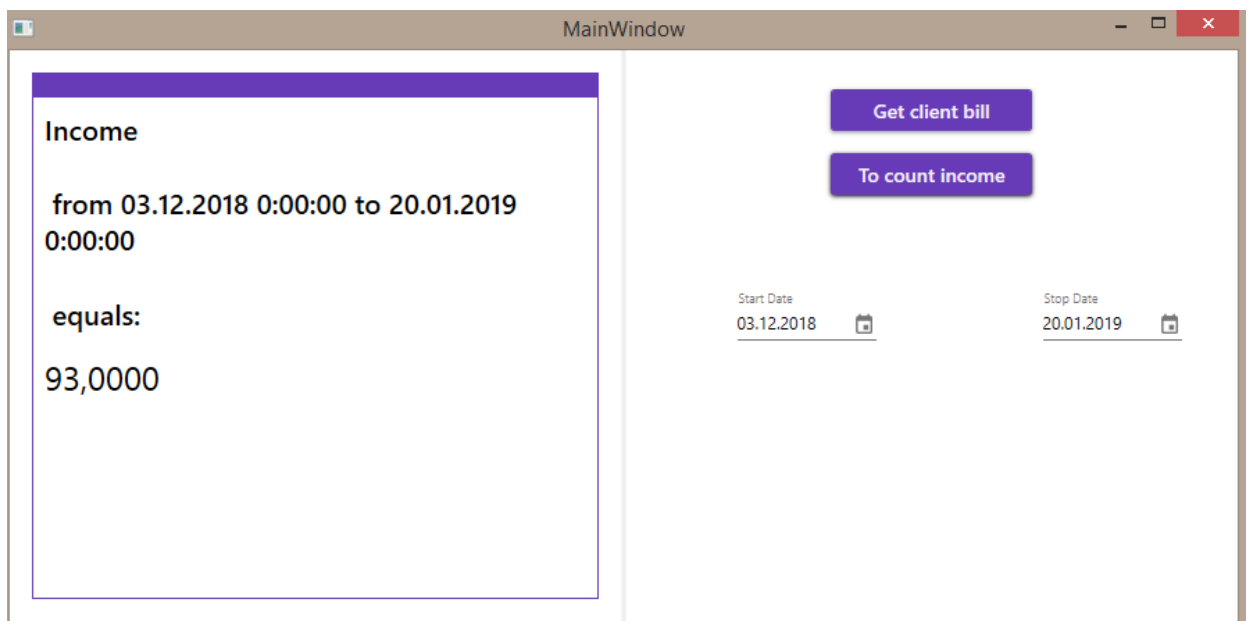


Рисунок 6.4 – Выполнение подсчёта дохода за определённый промежуток времени

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 – 15.11.2018.

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,
ISAVAILABLE BIT
)
```

```
CREATE TABLE ADDSERVICE(
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 ADDSERVICE(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 ADDSERVICE WHERE ADDSERVICE.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
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 ISAVAILABLE = 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.ADDSERVICE FOR XML
PATH(''ADDSERVICE''), ROOT(''ArrayOfADDSERVICE'')" queryout
"E:\University\3k1s\DatabaseAdministrationCP\xml\ADDSERVICE\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),
    @paswrn 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, @paswrn;
    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
                    @paswrn= CONVERT( VARCHAR(20) ,DecryptByPassPhrase
                    (@PassphraseEnteredByUser, CLIENT.PASWRD)));
                fetch next from temp_cursor into @firstname, @mail,
                @paswrn;
                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),

```



```

        @paswrđ 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, @paswrđ;
    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, @paswrđ) ))
                FETCH NEXT FROM temp_cursor into @firstname, @mail,
@paswrđ;
            END
            PRINT 'Registration completed successfully!'
        END
    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 ISAVAILABLE = 1 WHERE ROOM_NUMBER =
                        @room
                FETCH NEXT FROM curs INTO @room, @date
            END
        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
ADDSERVICE WHERE ADDSERVICE.ID = @serviceId)
        ELSE
            SET @serviceCost = @serviceCost + (SELECT COST FROM ADDSERVICE WHERE
ADDSERVICE.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

```